UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

INCORPORANDO DADOS ESPACIAIS VAGOS EM DATA WAREHOUSES GEOGRÁFICOS: A PROPOSTA DO TIPO ABSTRATO DE DADOS VAGUEGEOMETRY

ANDERSON CHAVES CARNIEL

ORIENTADOR: PROF. DR. RICARDO RODRIGUES CIFERRI

São Carlos – SP

Outubro/2014

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

INCORPORANDO DADOS ESPACIAIS VAGOS EM DATA WAREHOUSES GEOGRÁFICOS: A PROPOSTA DO TIPO ABSTRATO DE DADOS VAGUEGEOMETRY

ANDERSON CHAVES CARNIEL

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação, área de concentração: Engenharia de Software, Banco de Dados e Interação Humano Computador.
Orientador: Prof. Dr. Ricardo Rodrigues Ciferri

São Carlos – SP

Outubro/2014

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

C289id

Carniel, Anderson Chaves.
Incorporando dados espaciais vagos em data warehouses geográficos : a proposta do tipo abstrato de dados vaguegeometry / Anderson Chaves Carniel. -- São Carlos : UFSCar, 2014.
138 f.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2014.

1. Banco de dados. 2. *Data warehouse*. 3. Dados espaciais vagos. 4. Tipos abstratos de dados (Computação). I. Título.

CDD: 005.74 (20ª)

Universidade Federal de São Carlos

Centro de Ciências Exatas e de Tecnologia

Programa de Pós-Graduação em Ciência da Computação

"Incorporando Dados Espaciais Vagos em Data Warehouses Geográficos: a Proposta do Tipo Abstrato de Dados VagueGeometry"

Anderson Chaves Carniel

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação

Membros da Banca:

Prof. Dr. Ricardo Rodrigues Ciferri
(Orientador - DC/UFSCar)

Prof. Dr. Renato Bueno
(DC/UFSCar)

Prof. Dr. Valéria Cesário Times
(UFPE)

São Carlos
Outubro/2014

AGRADECIMENTOS

Aos meus queridos pais, que me apoiaram em todos os momentos acreditando em mim e por virem a São Carlos comigo para me apoiar ainda mais.

À minha querida namorada Dany, pelo amor, carinho e afeto, compreensão e apoio.

Ao meu orientador Prof. Dr. Ricardo Rodrigues Ciferri do DC/UFSCar e à Profa. Dra. Cristina Dutra de Aguiar Ciferri do ICMC/USP pelo grande apoio e confiança na recepção deste aluno de mestrado e as orientações realizadas.

Ao Prof. Dr. Markus Schneider pela orientação e recepção deste aluno no período de estágio no exterior.

A todas as pessoas envolvidas neste trabalho, que de algum modo me ajudaram com maior ou menor intensidade. Obrigado pelas palavras acolhedoras e amizade.

Ao projeto Nº 3280/2010 intitulado "Modelagem Dimensional com Aspectos Geográficos do Programa Observatório da Educação", para o qual tive bolsa de estudos no período de junho a julho de 2012.

Ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) pela bolsa de mestrado no mês de agosto de 2012.

À Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP) pela bolsa de mestrado sob o número de processo 2012/12299-8, com início em setembro de 2012 e pela bolsa BEPE sob o número de processo 2013/19633-3 no período de fevereiro a junho de 2014.

RESUMO

Um *data warehouse* é uma solução para a organização e o armazenamento de dados multidimensionais relacionados à tomada de decisão estratégica em empresas, constituindo um banco de dados histórico, volumoso, orientado ao assunto e não volátil. Um *data warehouse geográfico* (DWG) adicionalmente ao armazenamento de dados convencionais (tais como dados numéricos e alfanuméricos), armazena dados espaciais como atributos em tabelas de dimensão ou como medidas em tabelas de fatos, ou seja, armazena dados representados por meio de geometrias. Pontos, linhas e polígonos são exemplos de dados espaciais armazenados. Enquanto estes dados armazenados atualmente em DWGs são *crisp*, ou seja, possuem localização, interiores e fronteiras bem definidas, aplicações geográficas têm demandado o armazenamento de dados espaciais vagos, os quais possuem localização, interiores ou fronteiras incertas. Esta pesquisa de mestrado visa incorporar dados espaciais vagos em DWGs. Mais especificamente, foi proposto e implementado um novo tipo abstrato de dados (TAD), denominado VagueGeometry, para representar dados espaciais vagos no Sistema Gerenciador de Banco de Dados (SGBD) PostgreSQL com a extensão espacial PostGIS. A proposta do TAD VagueGeometry engloba a proposta de uma forma de armazenamento interna para os dados espaciais vagos, os quais são complexos e podem possuir diversas partes disjuntas. Isso também inclui a definição de operações para manipular objetos espaciais vagos, tais como os relacionamentos topológicos vagos e seus operadores. Avaliações experimentais foram conduzidas para medir o desempenho do TAD VagueGeometry frente a soluções existentes, tal como a implementação de predicados topológicos vagos reutilizando operações existentes do PostGIS. A proposta do TAD VagueGeometry apresentou reduções no tempo de processamento de predicados topológicos vagos de 81,63% a 90,34% em bancos de dados espaciais e reduções de 92,46% a 95,20% em ambientes de DWG. Este trabalho, portanto, avança no estado da arte em DWGs de forma a preencher essa lacuna existente na literatura. Adicionalmente, modelos *fuzzy* para representação dos dados espaciais vagos também foram estudados e uma proposta preliminar de um TAD, denominado FuzzyGeometry, também foi desenvolvida.

Palavras-chave: Data Warehouse, Data Warehouse Geográfico, Dados Espaciais Vagos, Tipo Abstrato de Dados

ABSTRACT

A data warehouse is a solution for organizing and storing multidimensional data related to decision-making processes in companies, generating a historical, highly voluminous, subject-oriented and nonvolatile database. A geographic data warehouse (GDW) additionally to the conventional data storage (i.e. numeric and alphanumeric data), stores spatial data as attributes in dimension tables or as measures in fact tables, storing data represented by geometries. Points, lines and polygons are examples of spatial data types. While spatial data currently stored in GDWs are crisp, i.e., they have exact location in the space, strict interiors and well-defined boundaries, geographic applications have required the storage of vague spatial data, which have inaccurate location, inexact interiors or uncertain boundaries. This Master's research aims at incorporating vague spatial data to GDWs. More specifically, we propose and implement a new abstract data type (ADT) called VagueGeometry to represent vague spatial data in the Spatial Database Management System (SDBMS) PostgreSQL/PostGIS. The proposal of the ADT VagueGeometry encompasses the issue of physical storage for vague spatial data, which are complex and can have several disjoint parts. It also focuses on definitions of operations to handle vague spatial objects, such as vague topological predicates and its operators. Experimental evaluations were conducted in order to assess the performance of the ADT VagueGeometry in comparison to available solutions, such as implementation of vague topological predicates utilizing existing operations of the PostGIS. The proposed ADT VagueGeometry shown reductions in query processing with vague topological predicates from 81.63% to 90.34% in spatial databases and reductions from 92.46% a 95.20% in GDW environments. This Master's project, therefore, advances in the state of art in GDWs to study this gap in the literature. Additionally, fuzzy models to represent vague spatial data was also studied, and as a result, a preliminary proposal of a ADT, called as FuzzyGeometry, was also developed.

Keywords: Data Warehouse, Geographic Data Warehouse, Vague Spatial Data, Asbtract Data Type

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

9-IM – *9-Intersection Model*

BLOB – *Binary Large Objects*

DE-9IM – *Dimensionally Extended 9-Intersection Model*

DWG – *Data Warehouse Geográfico*

DW – *Data Warehouse*

EFWKB – *Extended Fuzzy Well-Known Binary*

EFWKT – *Extended Fuzzy Well-Known Text*

EWKB – *Extended Well-known Binary*

EWKT – *Extended Well-known Text*

FMBR – *Fuzzy Minimum Bounding Rectangle*

FWKB – *Fuzzy Well-Known Binary*

FWKT – *Fuzzy Well-Known Text*

GML – *Geography Markup Language*

GeoJSON – *Geographic JSON*

JSON – *JavaScript Object Notation*

KML – *Keyhole Markup Language*

MBR – *Qualitative Min-Max model*

OGC – *Open Geospatial Consortium*

OLAP – *On-line Analytical Processing*

PL/pgSQL – *Procedural Language/PostgreSQL*

QMM – *Qualitative Min-Max model*

SGBDE – *Sistema Gerenciador de Banco de Dados Espacial*

SGBD – *Sistema Gerenciador de Banco de Dados*

SIG – *Sistema de Informação Geográfica*

SOLAP – *Spatial On-line Analytical Processing*

SRID – *Spatial Reference System Identifier*

TAD – *Tipo Abstrato de Dados*

VASA – *Vague Spatial Algebra*

WKB – *Well-known Binary*

WKT – *Well-known Text*

XML – *eXtensible Markup Language*

iBLOB – *Intelligence Binary Large Objects*

SUMÁRIO

Capítulo 1

INTRODUÇÃO

Este capítulo detalha o contexto em que este trabalho está inserido, a motivação e os objetivos desta dissertação. Por fim, também descreve a organização desta monografia.

1.1 Contexto

Atualmente, empresas têm requerido cada vez mais a compreensão de seus dados visando melhorar a lucratividade de seus negócios e auxiliar também na tomada de decisão estratégica. Um *data warehouse* (DW) é uma solução para a organização e o armazenamento de dados relacionados à tomada de decisão estratégica, constituindo um banco de dados histórico, volumoso, orientado ao assunto e não volátil (KIMBALL; ROSS, 2002; CIFERRI, 2013). Adicionalmente aos atributos convencionais mantidos no DW, atributos podem armazenar dados espaciais representados por meio de geometrias (por exemplo, um conjunto de pontos no espaço Euclidiano bidimensional), constituindo assim um *data warehouse* geográfico (DWG). Enquanto sobre o DW incidem consultas *Online Analytical Processing* (OLAP) (KIMBALL; ROSS, 2002), sobre o DWG incidem consultas *Spatial* OLAP (SOLAP) que viabilizam a análise multidimensional aliada à análise espacial (MALINOWSKI; ZIMÁNYI, 2008).

No DWG, comumente os dados espaciais têm natureza vetorial e usam tipos de dados geométricos para representar objetos espaciais do mundo real que sejam simples, tais como ponto, linha e polígono, ou complexos, tais como multiponto, multilinha e multipolígono. Esses objetos frequentemente têm sua localização exata no espaço, ou seja, assume-se que suas coordenadas geográficas definem com clareza a posição geográfica do objeto. Além disso, as fronteiras e interiores de regiões (ou seja, polígonos) definem com exatidão o formato da região. Tais objetos são conhecidos como *dados espaciais crisp*. Exemplos de dados espaciais *crisp* são propriedades rurais com suas fronteiras cadastrais bem definidas e países com suas fronteiras políticas bem demarcadas.

Por outro lado, fenômenos geográficos podem ter a localização inexata, fronteiras incertas e/ou interiores incertos. Objetos que possuem ao menos uma dessas características são denominados *objetos espaciais vagos*. Tais objetos espaciais não podem ser representados por dados espaciais *crisp* uma vez que a *vagueza espacial* pode ser negligenciada. Em geral, objetos espaciais vagos contêm partes que realmente pertencem ao fenômeno geográfico e partes que podem ou não pertencer ao fenômeno. Exemplos de dados espaciais vagos, mais especificamente de regiões vagas, são áreas de poluições de ar, oceanos, lagos e habitats de espécies de animais. Por exemplo, devido a diferentes concentrações de poluições no ar em diferentes locais, é difícil mapear com exatidão os limites da poluição no ar. Outro exemplo é o mapeamento de um lago, onde áreas que sofrem precipitações ou alagamentos pode afetar de maneira imprecisa a sua real extensão.

Na literatura, ainda se discute a respeito de padrões para a modelagem de dados espaciais vagos e suas operações, tais como os predicados topológicos envolvendo dados espaciais vagos. Estes padrões consistem por exemplo nos modelos exatos e *fuzzy*. Dessa forma, inexiste o suporte nativo para dados espaciais vagos em Sistemas Gerenciadores de Banco de Dados Espaciais (SGBDE), tais como no PostgreSQL com a extensão espacial PostGIS. Consequentemente, inexiste o suporte para estes tipos de dados em DWG também. Nesse sentido, a pesquisa sobre dados espaciais vagos é cada vez mais importante, uma vez que, o uso de dados espaciais vagos em DWGs é cada vez mais requerido para representar situações comumente encontradas no mundo real.

1.2 Motivação e Objetivos

A principal motivação para o desenvolvimento desta pesquisa em nível de mestrado é fazer com que aplicações do mundo real possam utilizar objetos espaciais vagos para a representação de fenômenos do mundo real que antes não podiam ser representados. Desafios relacionados ao uso de dados espaciais vagos referem-se à definição de tipos abstratos de dados (TADs) que abranja tanto aspectos de armazenamento quanto à manipulação de sua estrutura para processar operações espaciais em SGBDE, assim como em operações SOLAP em DWG. Um TAD é responsável por prover operações de alto nível a usuários finais, sem que os mesmos estejam preocupados na forma segunda a qual os objetos estão armazenados e nos algoritmos para sua manipulação.

Na literatura, trabalhos correlatos que implementam dados espaciais vagos, como os investigados no Capítulo 3, apresentam limitações que motivam o desenvolvimento de novas

pesquisas. Um exemplo de limitação é a falta de suporte de operações para manipulação de dados espaciais vagos, tal como os predicados espaciais vagos. Outra limitação foi apontada por Siqueira (2011) que demonstrou que a indexação de dados espaciais vagos e o processamento destes em tabelas relacionais, sem suporte nativo, não propicia um bom desempenho no processamento de consultas SOLAP. Essas limitações motivam o desenvolvimento deste projeto de mestrado, o qual avança no estado da arte da pesquisa em dados espaciais vagos bem como em DWGs de forma a preencher essa lacuna existente.

Esta pesquisa de mestrado tem como objetivo incorporar dados espaciais vagos em aplicações que utilizam banco de dados espaciais bem como em DWGs. Mais especificamente, é proposto e implementado um novo TAD, denominado VagueGeometry, para representar dados espaciais vagos no SGBD PostgreSQL com a extensão espacial PostGIS. O modelo exato VASA (PAULY; SCHNEIDER, 2010) foi o considerado para a representação dos objetos espaciais vagos do TAD VagueGeometry, devido a sua completude discutida no Capítulo 2. A proposta do TAD VagueGeometry engloba uma forma de armazenamento interna para dados espaciais vagos, os quais são complexos e podem possuir diversas partes disjuntas. Além disso, foi investigado e implementado operações para manipulação de dados espaciais vagos, tais como os predicados topológicos vagos, que foram avaliados experimentalmente em consultas sobre banco de dados espaciais bem como em consultas SOLAP.

Adicionalmente, o modelo *fuzzy* também foi considerado e investigado neste trabalho. É importante enfatizar que este estudo não estava previsto originalmente no projeto e que a inclusão deste estudo ocorreu pela importância do modelo *fuzzy* na modelagem de dados espaciais, o qual vem sendo usado em diversos trabalhos (ALTMAN, 1994; SCHNEIDER, 1999; VERSTRAETE; DE TRÉ; HALLEZ, 2006; SCHNEIDER, 2001; DILO; BY; STEIN, 2007). Como principal vantagem sobre os modelos exatos, a teoria de conjuntos *fuzzy* pode adicionar mais níveis de vagueza na modelagem de objetos espaciais baseados em fenômenos do mundo real. Em outras palavras, para cada ponto contido em um objeto espacial, é possível indicar o quanto ele pertence em um determinado fenômeno. Este valor, o qual pode variar de 0 à 1, é conhecido como o *grau de pertinência* de um ponto em um *objeto espacial fuzzy*. Dessa forma, esta dissertação apresenta a proposta do TAD FuzzyGeometry, para representação de dados espaciais vagos baseada no modelo *fuzzy*.

1.3 Organização da Monografia

Esta monografia está estruturada da seguinte forma. No Capítulo 2 é descrita a fundamentação teórica necessária para o entendimento deste trabalho, tais como as representações de dados espaciais vagos e definições de *data warehouse* geográfico. No Capítulo 3 são destacados trabalhos correlatos e discutidas suas principais limitações. No Capítulo 4 é detalhado a proposta do TAD VagueGeometry. Em seguida, no Capítulo 5, a análise experimental que avalia o desempenho do TAD VagueGeometry é apresentada. Como resultado adicional, no Capítulo 6 é apresentada a proposta do TAD FuzzyGeometry. Por fim, no Capítulo 7 a dissertação é concluída e trabalhos futuros direcionados.

Capítulo 2

FUNDAMENTAÇÃO TEÓRICA

Este capítulo resume conceitos sobre banco de dados espaciais, modelos de dados espaciais vagos e seu uso em data warehouses geográficos. Para os modelos de dados espaciais vagos, são descritos mais detalhadamente os modelos exatos e os modelos fuzzy.

2.1 Considerações Iniciais

Neste capítulo são apresentadas todas as definições necessárias para a compreensão dessa dissertação de mestrado. O capítulo está organizado da seguinte forma. Na Seção 2.2 é discutida a extensibilidade do Sistema Gerenciador de Banco de Dados (SGBD) PostgreSQL, que é usada como base para o desenvolvimento do TAD proposto nesta dissertação. O SGBD PostgreSQL é considerado nesta pesquisa de mestrado por ser de código fonte aberto, gratuito e amplamente usado em pesquisas científicas e na indústria. Na Seção 2.3 são resumidos conceitos de banco de dados espaciais e tipos de dados espaciais *crisp*. Na Seção 2.4 são detalhados conceitos sobre dados espaciais vagos. Modelos exatos para representar dados espaciais vagos são descritos na Seção 2.4.1, enquanto modelos *fuzzy* são descritos na Seção 2.4.2. Na Seção 2.5 são resumidos os conceitos de *data warehouse* e *data warehouse* geográfico. O capítulo é finalizado na Seção 2.6 com as considerações finais.

2.2 Extensibilidade do SGBD PostgreSQL

PostgreSQL é um SGBD extensível com código fonte aberto que permite a adição de novos tipos de dados definidos pelo usuário (POSTGRESQL..., 2014). A primeira forma de especificar um novo tipo de dado é o uso de linguagens de baixo nível, como C, enquanto uma outra forma é por meio de procedimentos armazenados em linguagens, como *Procedural Language/PostgreSQL* (PL/pgSQL). Para isso, é necessário definir várias características sobre o

novo tipo de dado a ser armazenado e manipulado pelo SGBD PostgreSQL: (i) o tamanho em bytes do tipo de dado, o qual pode ser fixo ou variável; (ii) as funções de entrada e saída (*input/output*); (iii) o tipo de armazenamento; (iv) alinhamento dos dados, caso o tipo de dado tenha tamanho variável; e, (v) outras características, por exemplo o delimitador em vetores, modificadores de tipos de dados e a categoria pertencente do tipo de dado. Todas essas características são definidas no comando SQL CREATE TYPE e são detalhadas a seguir.

Primeiramente, é necessário conhecer previamente se o novo tipo de dado terá um tamanho fixo ou um tamanho variável. Para um tipo de dado de tamanho fixo, o tamanho em *bytes* da estrutura interna deve ser informada por meio do comando CREATE TYPE. Caso o tamanho seja variável, o primeiro elemento da estrutura a ser armazenada internamente pelo SGBD deve conter em um inteiro de 4 *bytes* (*int32*) o tamanho do objeto. Além disso, deve ser definido o alinhamento entre os valores da estrutura interna. O alinhamento diz respeito à forma segunda a qual os objetos estarão organizados em disco (por exemplo, a cada 8 *bytes* existe um valor armazenado).

As funções obrigatórias para definição de um novo tipo de dado no SGBD PostgreSQL são as funções de entrada e saída (*input/output*). A função de entrada (*input*) converte a representação textual ASCII externa para a representação interna do tipo de dado, a qual é usada pelos operadores internos definidos pelo novo tipo de dado. Já a função de saída (*output*) realiza a conversão inversa. Opcionalmente, é possível especificar funções binárias de entrada e saída, as quais são definidas como *send* e *receive*. A função *send* converte a representação binária para a representação interna, enquanto a função *receive* realiza a conversão inversa. As operações binárias são em geral mais rápidas do que as operações textuais, porém menos portáveis.

Na sequência, deve ser definida qual estratégia de armazenamento o tipo de dado irá realizar. As estratégias que o SGBD PostgreSQL permite são: (i) *plain*, somente para tipos de dados que têm tamanhos fixos, os quais serão armazenados em formato não comprimido; (ii) *extended*, na qual o valor do objeto a ser armazenado passará por uma compressão e caso o mesmo seja muito grande, ele será movido para fora da tabela principal. A tabela principal, é o local onde o tipo de dado está definido como coluna; (iii) *external*, a qual, de forma contrária à estratégia *extended*, permite que o objeto a ser armazenado seja movido para fora da tabela principal sem comprimi-lo; e, (iv) *main*, a qual une as características das estratégias *extended* e *external*, permitindo a compressão do objeto a ser armazenado, porém desencorajando movê-lo de sua tabela principal. Nessa estratégia talvez os objetos sejam movidos se não houver outro meio, mas eles serão mantidos na tabela principal preferencialmente. Ressalta-se que para tipos de dados com tamanho variável é recomendável o uso da estratégia *extended*, *external* ou *main*.

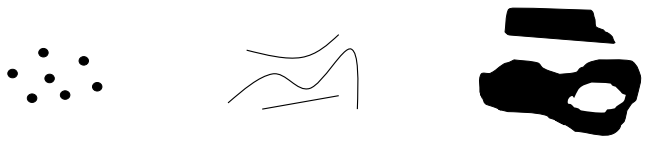
Por fim, podem ser definidas outras características do tipo de dado, tais como modificadores de tipos, delimitadores e a categoria do tipo de dado. Um *modificador de tipo* é especificado por meio de duas funções, denominadas *typmod_input* e *typmod_output*. A função *typmod_input* tem como objetivo armazenar, como um número inteiro, a restrição do modificador de tipo associado a uma coluna de uma tabela de um banco de dados. Por exemplo, para o tipo de dado VARCHAR(50), o valor do *typmod_input* é 50, o qual restringe o número de caracteres de um objeto VARCHAR. Já a função *typmod_output* mostra para o usuário o valor do modificador armazenado no objeto. O delimitador é definido como separador de elementos em um vetor de objetos do novo tipo abstrato de dados. Por fim, a categoria do tipo de dado informa ao SGBD se o tipo de dado é por exemplo, do tipo geométrico, tipo numérico ou tipo de data.

Após a especificação de um novo tipo de dado, são definidas funções (CREATE FUNCTION) em baixo ou alto nível para manipular esse tipo de dado. A partir da versão 9.0 do SGBD PostgreSQL, encontra-se disponível um mecanismo para compilação e definição de uma nova extensão de forma automatizada, denominado PGXS. PGXS é um programa interno para o ambiente Linux do SGBD PostgreSQL que, por meio de alguns arquivos de configuração, compila os arquivos de código fonte necessários e os copia para os diretórios corretos do servidor. A extensão é então efetivada por meio do comando SQL CREATE EXTENSION, a qual carrega a extensão para o banco de dados conectado. Dessa forma, uma extensão pode ser carregada ou excluída (DROP EXTENSION) sem a necessidade de interromper o serviço do SGBD PostgreSQL.

2.3 Banco de Dados Espaciais

Sistemas Gerenciadores de Banco de Dados Espaciais (SGBDEs) e Sistemas de Informações Geográficas (SIGs) comumente usam dados espaciais *crisp* no espaço Euclidiano bidimensional ou tridimensional para representar fenômenos do mundo real. Nesta dissertação são tratados apenas objetos bidimensionais no espaço $\mathbb{R}^2$. Tipos de dados espaciais para pontos *crisp*, linhas *crisp*, e regiões *crisp* (também denominados como polígonos) são propostos para essas representações (GÜTING, 1994; CLEMENTINI; FELICE, 1996; CIFERRI, 2002; SCHNEIDER; BEHR, 2006). Na Figura 2.1 são ilustrados exemplos desses tipos de dados, os quais são caracterizados por sua localização, fronteira, interior e extensão precisamente definidos no espaço. Por exemplo, uma região *crisp* pode representar um país com suas fronteiras políticas bem definidas.

Além disso, operações para manipular objetos espaciais *crisp* são propostos, tal como predicados topológicos *crisp* (e.g., *overlap*), operações geométricas de conjunto (e.g., união



(a) (b) (c)

Figura 2.1: Exemplos de dados espaciais complexos *crisp*, ponto *crisp* (a), linha *crisp* (b) e região (polígono) com ilha e buraco *crisp* (c).

geométrica) e operações numéricas (e.g. área de uma região) (GÜTING, 1994). Em geral, os predicados topológicos são mais investigados na literatura devido a sua importância em consultas espaciais. Um relacionamento topológico entre dois objetos espaciais é descoberto por meio da combinação das intersecções entre bordas, interiores e exteriores de um conjunto de pontos no espaço topológico *crisp*. A borda, interior e exterior de um conjunto de pontos S, representados como ∂S, S° e S^- respectivamente, são mutualmente disjuntos no espaço topológico *crisp*. Intuitivamente, um ponto *crisp* não contém borda, contendo somente interior. A borda de uma linha *crisp* é composta por seus pontos finais, enquanto seu interior é composto pelos pontos entre os pontos finais. A borda de uma região *crisp* é definida como o seu contorno (ou seja, uma linha *crisp*), enquanto seu interior são todos os pontos dentro da região. O exterior de um objeto espacial *crisp* A, é o complemento entre a união espacial da borda e do interior, i.e. $A^- = \mathbb{R}^2 - (A^\circ \cup \partial A)$ (SCHNEIDER; BEHR, 2006).

Dessa forma, combinando a intersecção entre as bordas, interiores e exteriores de dois objetos espaciais A e B, uma matriz 3×3 é obtida. Esta matriz é chamada de *9-Intersection Model* (9-IM) (CLEMENTINI; SHARMA; EGENHOFER, 1994; CLEMENTINI; FELICE, 1996; SCHNEIDER; BEHR, 2006). Cada célula dessa matriz tem um valor em $\{0,1\}$ correspondendo se existe (i.e., valor 1) ou não (i.e., valor 0) uma intersecção. Existem 29 possíveis configurações dessa matriz. Cada configuração simboliza um relacionamento topológico, porém somente oito relacionamentos são comumente utilizados. Esses predicados são *disjoint*, *meet*, *overlap*, *equal*, *inside*, *contains*, *covers* e *coveredBy*. Dessa forma os predicados topológicos são mutualmente exclusivos e completos de acordo com o relacionamento topológico da matriz e considerando o critério de existência ou não de uma intersecção. Sejam A e B dois objetos espaciais *crisp*, a matriz 9-IM é definida como

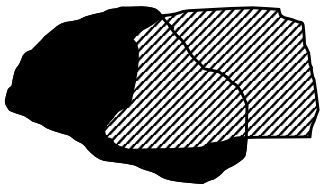
$$\textit{9-IM}(A,B) = \begin{pmatrix} A^\circ \cap B^\circ \neq \varnothing & A^\circ \cap \partial B \neq \varnothing & A^\circ \cap B^- \neq \varnothing \\ \partial A \cap B^\circ \neq \varnothing & \partial A \cap \partial B \neq \varnothing & \partial A \cap B^- \neq \varnothing \\ A^- \cap B^\circ \neq \varnothing & A^- \cap \partial B \neq \varnothing & A^- \cap B^- \neq \varnothing \end{pmatrix}$$

Na Figura 2.2b é mostrado um exemplo de relacionamento topológico entre duas regiões

crisp da Figura 2.2a. Nesta figura é adotada a representação T para o valor 1 e F para o valor 0.

Adicionalmente, a dimensionalidade das intersecções entre as bordas, interiores e exteriores podem também ser consideradas, constituindo assim uma matriz dimensional chamada como *Dimensionally Extended 9-Intersection Model* (DE-9IM) (CLEMENTINI; DI FELICE; OOSTEROM, 1993; CLEMENTINI; DI FELICE, 1995). A dimensionalidade do objeto espacial resultante de uma intersecção pode ser 0 (ponto), 1 (linha) ou 2 (região). Com a matriz do modelo DE-9IM é possível restringir ainda mais um relacionamento topológico e assim criar predicados considerando também a dimensionalidade das intersecções. Um exemplo de uso desse modelo é dado na Figura 2.2c, onde o resultado de sua matriz em forma textual é "212101212". Esta forma textual é utilizada para representar uma matriz no modelo DE-9IM, onde cada elemento corresponde a um respectivo elemento da matriz. Por exemplo, o primeiro elemento da matriz (i.e., o elemento $(0,0)$) é 2, assim, o primeiro elemento da forma textual é "2".

Devido a complexidade dos relacionamentos topológicos, é possível melhorar seu processamento por meio de técnicas de aproximações, tal como o uso de *Minimum Boundary Rectangles* (MBRs). Estas técnicas tem como objetivo eliminar situações onde os predicados topológicos não ocorrem, sem a necessidade de computar o relacionamento topológico sobre os objetos originais. Um MBR é constituído pelas coordenadas mínimas e máximas de x e y, formando assim um retângulo que envolve de forma mínima um objeto espacial. Dessa forma, na maioria das vezes, o cálculo dos relacionamentos topológicos sobre tais retângulos se torna mais eficiente do que no objeto espacial. Em Clementini, Sharma e Egenhofer (1994) é discutido o uso dos relacionamentos entre os MBRs para possivelmente determinar os relacionamentos entre os objetos espaciais. Por exemplo, o relacionamento *disjoint* entre dois MBRs estabelece que os objetos espaciais com certeza são disjuntos e que portanto, não existe a possibilidade de ocorrer outro tipo de relacionamento topológico. Outro exemplo é que, se um MBR não está contido em outro MBR, então não existe a possibilidade de ocorrer o relacionamento *inside*, *contains*, *coveredBy* e *covers* sobre tais objetos.



(a)

$$\begin{pmatrix} T & T & T \\ T & T & T \\ T & T & T \end{pmatrix}$$

(b)

$$\begin{pmatrix} 2 & 1 & 2 \\ 1 & 0 & 1 \\ 2 & 1 & 2 \end{pmatrix}$$

(c)

Figura 2.2: A matriz resultante 9-IM (b) e a matriz resultante DE-9IM (c) do relacionamento topológico de duas regiões *crisp* sobrepostas (a) (região de cor preta com a região de interior tracejado).

Nesse sentido, SIGs e SGBDEs disponibilizam TADs que armazenam e manipulam dados espaciais *crisp* para realizar tais operações. PostGIS (POSTGIS..., 2014) é uma extensão espacial, com código fonte aberto, para o SGBD PostgreSQL, o qual implementa dados espaciais *crisp* e suas operações espaciais baseando-se nas especificações da *Open Geospatial Consortium* (OGC) (OPEN..., 2014). O PostGIS utiliza a extensibilidade do SGBD PostgreSQL (Seção 2.2) para definir dados espaciais *crisp*. Com o PostGIS é possível especificar atributos espaciais do tipo *ponto*, *linha* ou *polígono* (i.e. que representa uma região) em uma tabela relacional do PostgreSQL, os quais podem ser simples ou complexos (e.g. polígonos simples ou polígonos com buraco e ilhas). As operações gerais do PostGIS são discutidas na Seção 3.2 do Capítulo 3.

Contudo, o PostGIS não provê suporte a dados espaciais vagos e nem operações para manipulá-los. Claramente, camadas para tratamentos desses tipos de dados bem como de suas operações são necessárias para o armazenamento e a manipulação de dados espaciais vagos. Tendo em vista essa lacuna, esta pesquisa de mestrado propõe um TAD que implementa e manipula dados espaciais vagos por meio de uma extensão do PostGIS. Nas próximas seções são apresentados conceitos sobre dados espaciais vagos e seus modelos de representações.

2.4 Dados Espaciais Vagos

Um dado espacial vago é um objeto espacial caracterizado com *vagueza espacial*. Um objeto espacial que apresenta vagueza espacial tem ao menos a sua localização, seu interior ou fronteira não bem definidos. Em geral, esses objetos espaciais contêm um núcleo e uma parte hipotética. O núcleo de um objeto espacial vago é a parte exata, ou seja, a parte espacial com localização, fronteira e interior bem conhecidos e definidos. Enquanto a parte hipotética é a parte que demonstra incerteza, ou seja, a parte espacial com localização, fronteiras ou interiores incertos. Na literatura, a representação dos dados espaciais vagos pode ser classificada principalmente nos modelos descritos a seguir.

- Modelos exatos: reusam os conceitos e implementações já existentes de dados espaciais *crisp* (que possuem localizações exatas, com fronteiras e interiores bem definidos) para manipular dados espaciais vagos, por meio de adaptações. Em geral, é considerado aproximações máxima e mínima de uma região e separam as porções do espaço que pertencem à região em: partes espaciais que certamente pertencem à região (núcleo), partes espaciais que possivelmente pertencem à região (parte hipotética) e as partes espaciais que certamente não pertencem à região. Alguns modelos exatos consideram a região do

núcleo como sendo disjunta da parte hipotética.

- Modelos probabilísticos: são baseados em funções de densidade de probabilidade (CHENG; KALASHNIKOV; PRABHAKAR, 2003, 2004; CHENG, 2004; ZINN; BOSCH; GERTZ, 2007) e tratam a incerteza das posições no espaço e medidas dos objetos (por exemplo, a sua área ou perímetro). Em geral, essas técnicas lidam com a expectativa de um evento acontecer no futuro, baseando-se em características já conhecidas. Ressalta-se que, enquanto as funções de densidade de probabilidade são exatas, a localização do objeto é incerta. Além de que, a vagueza espacial considerada é somente no futuro e não no presente. Consultas por abrangência probabilística também já foram estudadas para dados espaciais vagos do tipo ponto (TAO, 2005), bem como consultas relativas aos k-vizinhos mais próximos em bancos de dados espaciais com incerteza baseados em funções de densidade de probabilidade (LI, 2007; YUEN, 2010). Exemplos de aplicações desses modelos incluem a estimativa da temperatura de uma região e a definição do nível de água em um lago após um incidente.

- Modelos *fuzzy*: são baseados na teoria de conjuntos *fuzzy* para tratar a vagueza espacial e descrevem a possibilidade de um determinado ponto pertencer a um objeto espacial, ou do quanto uma afirmação pode ser verdadeira. Porém, não descrevem uma expectativa futura. O conceito de retângulo envolvente mínimo *fuzzy* (*fuzzy minimum bounding rectangle* - FMBR) foi proposto com o intuito de representar regiões de pertinência relativas aos dados espaciais vagos (SOMODEVILLA; PETRY, 2004). A partir das funções de pertinência resultantes, foi definido o conceito de *data warehouse* geográfico *fuzzy* (DAVID; SOMODEVILLA; PINEDA, 2007), o qual também será discutido na Seção 2.5. Adicionalmente, tipos de dados espaciais *fuzzy* que incorporam funções de pertinência para representar a vagueza espacial foram definidos (SCHNEIDER, 1999, 2001; VERSTRAETE; DE TRÉ; HALLEZ, 2006; DILO, 2006a; DILO; BY; STEIN, 2007; SCHNEIDER, 2008, 2014). Dados espaciais do tipo ponto *fuzzy*, linha *fuzzy* e região *fuzzy* foram propostos para esta finalidade. Operações espaciais *fuzzy* também são definidas, tais como operações geométricas de conjunto *fuzzy* (e.g. união geométrica *fuzzy*), relacionamentos topológicos *fuzzy* (e.g. *fuzzy overlap*) e operações numéricas (e.g. área de uma região *fuzzy*).

Os modelos exatos são menos refinados do que os modelos probabilísticos e *fuzzy* para representar os dados espaciais vagos e a sua semântica, porém os modelos probabilísticos e *fuzzy* requerem a definição de funções probabilísticas complexas ou funções de pertinência complexas que sejam adequadas ao problema. Além disso, o custo computacional para implementar as estruturas de dados e os algoritmos para os modelos probabilísticos e *fuzzy* é muito maior,

se comparado aos modelos exatos. Esta dissertação de mestrado investiga prioritariamente os modelos exatos e posteriormente os modelos *fuzzy*, os quais são detalhados nas Seções 2.4.1 e 2.4.2, respectivamente.

2.4.1 Modelos Exatos

Os principais modelos exatos existentes na literatura são: Egg-Yolk (COHN; GOTTS, 1995), QMM (BEJAOUI, 2009, 2010) e VASA (PAULY; SCHNEIDER, 2010). Em geral, os modelos exatos objetivam reutilizar os TADs implementados em SGBDEs para prover suporte a dados espaciais vagos. Isso inclui a definição de operações espaciais vagos sobre dados espaciais vagos a fim de sua utilização em aplicações do mundo real.

O trabalho de Cohn e Gotts (1995) define o modelo Egg-Yolk para a representação de regiões vagas com fronteiras cujos traçados não são bem definidos. Não foram considerados os tipos de dado ponto ou linha, e nem foi abordada a inexatidão de localização de uma região. Uma região com fronteiras vagas é representada por meio de duas (ou mais) sub-regiões concêntricas para indicar níveis de vagueza. A sub-região denominada *gema* representa a extensão (ou seja, área) mínima da região vaga, enquanto que a sub-região denominada *clara* representa a extensão máxima da região vaga. A união das sub-regiões gema e clara resulta no ovo, que representa o dado espacial vago. Todos os pontos contidos na gema certamente pertencem à região vaga, enquanto que todos os pontos fora da clara e da gema certamente não pertencem à região vaga. A área de incerteza, localizada na clara, contém todos os pontos que podem ou não pertencer à região vaga. É nessa área que se encontra a fronteira vaga. Um exemplo de região vaga segundo o modelo Egg-Yolk é exibido na Figura 2.3.

É importante notar que, no modelo Egg-Yolk, além de existir uma área vaga que rodeia os limites da região vaga, essa mesma área pode não possuir fronteiras bem definidas. Extensões ao modelo Egg-Yolk incluem o modelo de ovos mexidos (GUESGEN, 2002) e o modelo *Fuzzy Region Connection Calculus* (SCHOCKAERT, 2008), os quais são auxiliados pela teoria de conjuntos *fuzzy*.

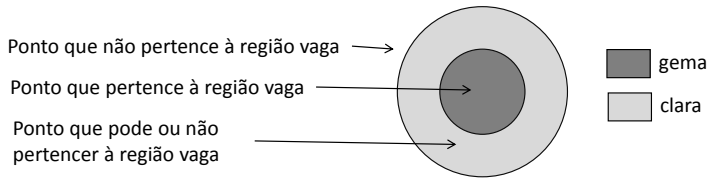


Figura 2.3: Uma região vaga conforme o modelo Egg-Yolk (Adaptada de Cohn e Gotts (1995)).

O modelo QMM (*Qualitative Min-Max model*) de Bejaoui (2009, 2010) define tipos de dados espaciais vagos que descrevem fenômenos com fronteiras cujos traçados não são bem definidos por meio de limites mínimos e máximos. O QMM adota classificações qualitativas para diferenciar *níveis de vagueza* nos objetos espaciais vagos, tais como: *completamente crisp*, *parcialmente vago* e *completamente vago*.

Para representar os dados espaciais vagos, o QMM reusa os princípios do modelo Egg-Yolk, e os estende para lidar com pontos, linhas e regiões. O limite mínimo é uma geometria que se refere à parte que certamente pertence ao fenômeno do mundo real, enquanto o limite máximo define uma parte vaga onde tal fenômeno pode ou não ocorrer. Já o exterior denota onde tal fenômeno certamente não ocorre. Na Figura 2.4 são exemplificados objetos espaciais vagos do tipo ponto e região (polígono) segundo o modelo QMM. Um ponto *crisp* é um objeto com dimensão zero. Já o ponto vago consiste em uma região *crisp*. Desse modo, um ponto vago não possui uma extensão mínima, mas apenas uma extensão máxima. No tratamento de regiões vagas, o modelo QMM estabelece duas regiões *crisp* que definem o limite mínimo e o máximo. Dada uma região A e seus limites A_{min} e A_{max}, se $A_{max} = A_{min}$, então A é uma região *completamente crisp*. Caso A_{max} cubra A_{min}, então a fronteira de A é vaga apenas em alguns pontos (os perímetros de A_{max} e A_{min} possuem intersecção), de modo que A é uma região *parcialmente vaga*. Por fim, se A_{max} contém A_{min}, então A é uma região *completamente vaga*. Não há suporte para pontos e polígonos complexos, nem polígonos com buracos no modelo QMM.

Uma linha *crisp* é um objeto unidimensional constituído de interior e dois pontos como limite. A sua borda são os dois limites e o seu interior é a união dos pontos que conectam esses dois pontos. Já uma linha vaga pode ter quaisquer dos seus pontos como sendo vagos. Além disso, cada componente pode ser *completamente crisp*, *parcialmente vago* ou *completamente vago*. Essas características determinam 9 tipos de linha vaga definidas pelo QMM, conforme a Figura 2.5. Ressalta-se que não há suporte para linhas que contenham auto-intersecções, linhas fechadas e linhas complexas.

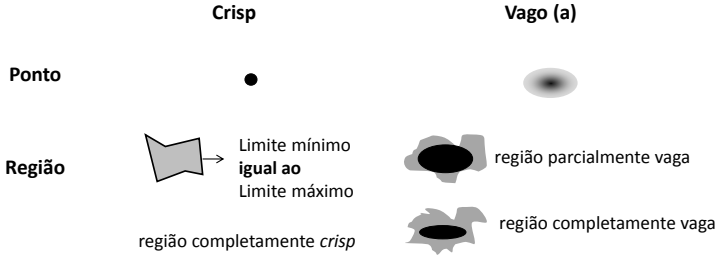


Figura 2.4: Pontos vagos e regiões vagas de acordo com o modelo QMM (Adaptada de Bejaoui (2009)).

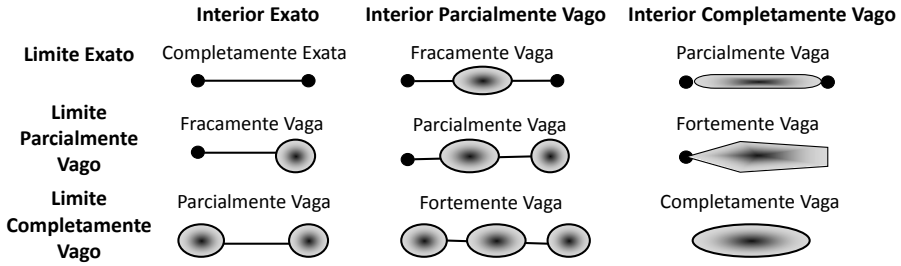


Figura 2.5: Linhas vagas de acordo com o modelo QMM (Adaptada de Bejaoui (2009)).

Também na Figura 2.5 são usados os advérbios *fracamente*, *parcialmente*, *fortemente* e *completamente*, os quais constituem recursos do modelo QMM para prover uma caracterização qualitativa. Utiliza-se *fracamente* para descrever uma característica que ocorre apenas uma vez. Por exemplo, uma linha com borda *crisp* e interior vago é caracterizada por um componente vago, denotando uma linha *fracamente vaga*. De forma análoga, os advérbios *parcialmente*, *fortemente* e *completamente* são empregados para relatar duas, três ou quatro vezes a ocorrência de uma característica, respectivamente.

O modelo exato VASA (*Vague Spatial Algebra*) (PAULY; SCHNEIDER, 2010) propõe uma álgebra que define tipos de dados espaciais vagos genéricos e denominados *vponto2D*, *vlinha2D* e *vpolígono2D*, os quais podem se referir a objetos espaciais simples ou complexos. Além disso, este modelo define operações para manipulação destes objetos. Dessa forma, esta álgebra é mais completa que os outros modelos apresentados.

Um objeto espacial vago é descrito por um par de objetos espaciais complexos *crisp* do mesmo tipo de dado disjuntos ou adjacentes. Um dos objetos espaciais define o *núcleo*, enquanto que o outro define a *conjectura*, ou seja, a parte hipotética. Enquanto o núcleo se refere à porção conhecida e determinada, a conjectura se refere à porção vaga. Ou seja, o núcleo certamente pertence ao objeto, mas a parte hipotética pode ou não pertencer a ele. Além disso, os interiores do núcleo e da parte hipotética são disjuntos. De modo geral, os objetos espaciais podem comprimir-se ou expandir-se de acordo com um limite mínimo e um limite máximo. Por exemplo, um lago cujo nível de água pode variar conforme o nível de evaporação e de precipitação. Mais formalmente, um dado espacial vago do tipo α é construído a partir de um construtor $v(\alpha) = \alpha \times \alpha$, tal que $\alpha \in \{ponto, linha, região\}$. Ou seja, um ponto vago w é definido por dois pontos *crisp*, tal que $w = v(ponto)$. Assim, para o objeto espacial vago $w = (w_n, w_c)$, onde o w_n corresponde ao núcleo do objeto e w_c a conjectura, tem-se que $disjunto(w_n, w_c) \vee\ toca(w_n, w_c)$.

Na Figura 2.6 são exemplificados dados espaciais vagos do modelo exato VASA. Na Fi-

gura 2.6a é exibido um ponto vago simples e na Figura 2.6d é exibido um ponto vago complexo (multiponto) compostos pelo núcleo na cor preta e pela conjectura na cor cinza. Na Figura 2.6b é exibida uma linha vaga simples e na Figura 2.6e é exibida uma linha vaga complexa (multilinha), cujo traçado contínuo pertence ao núcleo e tracejado pertence à conjectura. Por fim, na Figura 2.6c é mostrada uma região (polígono) vaga simples e na Figura 2.6f é mostrada uma região vaga complexa (multipolígono) cujas regiões pretas compõem o núcleo, enquanto que as regiões cinza pertencem à conjectura.

Além disso, o modelo exato VASA define um conjunto de operações para manipular seus tipos de dados espaciais vagos, os quais inclui operadores geométricos de conjuntos vagos (e.g. união, intersecção e diferença), predicados topológicos vagos (e.g. disjunção vaga, sobreposição vaga), operadores numéricos (e.g. distância entre dois pontos vagos) e operações específicas de tipos (e.g. borda de uma região vaga).

Em especial, os predicados topológicos vagos podem retornar três valores lógicos, uma vez que o modelo exato VASA é baseado em três níveis de vagueza (núcleo, conjectura e o exterior deles). Logo, os valores lógicos definidos são *true*, *false* e *maybe*. Sejam A e B dois objetos espaciais vagos, de maneira geral, os seguintes relacionamentos topológicos *crisp*, por meio das matrizes 9-IM ou DE-9IM (Seção 2.3), são realizados: $A_n \times B_n$, $A_n \times (B_n \cup B_c)$, $(A_n \cup A_c) \times B_n$ e $(A_n \cup A_c) \times (B_n \cup B_c)$.

Adicionalmente, os operadores numéricos retornam um par de valores correspondendo a um valor máximo e a um valor mínimo. Por exemplo, para o cálculo da área de uma região vaga, o valor mínimo corresponde a área que com certeza a região vaga tem, enquanto o valor máximo corresponde a uma área máxima de sua extensão, considerando portanto a conjectura em seu cálculo.

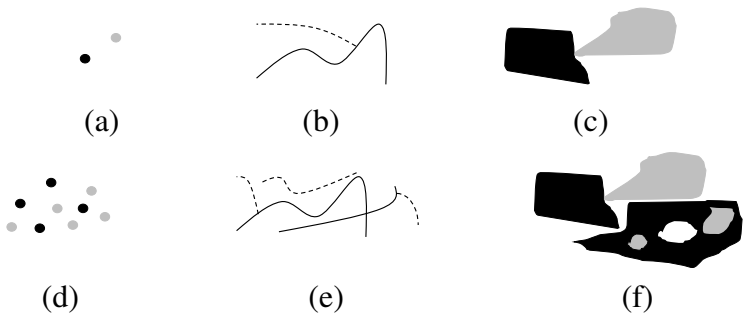


Figura 2.6: Exemplos de dados espaciais vagos simples (a)-(c) e complexos (d)-(f) seguindo o modelo exato VASA.

2.4.2 Modelos Baseados na Teoria de Conjuntos *Fuzzy*

A teoria de conjuntos *fuzzy* (ZADEH, 1965) também é usada para a modelagem de dados espaciais vagos. Os conceitos da teoria de conjuntos *fuzzy* são resumidos na Seção 2.4.2.1. Na Seção 2.4.2.2 são discutidas as representações de dados espaciais vagos utilizando a teoria de conjuntos *fuzzy*.

2.4.2.1 Teoria de Conjuntos *Fuzzy*

A teoria de conjuntos *fuzzy* é uma generalização da teoria de conjuntos clássica booleana. Seja X um conjunto clássico (*crisp*) de objetos, chamado de *universo*. A pertinência em um subconjunto clássico A de X pode então ser descrita pela *função característica* $\chi_A : X \to \{0,1\}$ tal que para todo $x \in X$ garante-se que $\chi_A(x) = 1$ se, e somente se, $x \in A$, e $\chi_A(x) = 0$ caso contrário. Um conjunto *fuzzy* relaxa essa discriminação rígida e mapeia todos os elementos de X para o intervalo real $[0,1]$ indicando o *grau de pertinência* desses elementos em um conjunto em questão. Seja X novamente o universo. A função $\mu_{\tilde{A}} : X \to [0,1]$ é chamada de *função de pertinência* do *conjunto fuzzy* $\tilde{A} = \{(x, \mu_{\tilde{A}}(x)) \,|\, x \in X\}$. Assim, a teoria de conjuntos *fuzzy* permite que um elemento tenha participação parcial em um conjunto *fuzzy* e que tenha diferentes *valores de pertinência* em diferentes conjuntos *fuzzy*.

As operações de conjuntos *fuzzy* são também generalizações das operações de conjuntos clássicos. Sejam $\tilde{A}$ e $\tilde{B}$ conjuntos *fuzzy* em X, as operações de intersecção *fuzzy*, união *fuzzy*, diferença *fuzzy*, diferença limitada *fuzzy* (a qual corresponde a operação de diferença em conjuntos *crisp*), diferença absoluta *fuzzy* (a qual corresponde a operação de diferença simétrica em conjuntos *crisp*) e contenção de conjunto *fuzzy* são definidas como segue, respectivamente (ZADEH, 1965; DUBOIS; PRADE; PRADE, 2000)

- $\tilde{A} \cap \tilde{B} = \{(x, \mu_{\tilde{A}\cap\tilde{B}}(x)) \,|\, x \in X \wedge \mu_{\tilde{A}\cap\tilde{B}}(x) = \min(\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x))\}$
- $\tilde{A} \cup \tilde{B} = \{(x, \mu_{\tilde{A}\cup\tilde{B}}(x)) \,|\, x \in X \wedge \mu_{\tilde{A}\cup\tilde{B}}(x) = \max(\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x))\}$
- $\tilde{A} - \tilde{B} = \{(x, \mu_{\tilde{A}-\tilde{B}}(x)) \,|\, \mu_{\tilde{A}-\tilde{B}}(x) = \min(\mu_{\tilde{A}}(x), 1 - \mu_{\tilde{B}}(x)\}$
- $\tilde{A} \dot{-} \tilde{B} = \{(x, \mu_{\tilde{A}\dot{-}\tilde{B}}(x)) \,|\, \mu_{\tilde{A}\dot{-}\tilde{B}}(x) = \max(0, \mu_{\tilde{A}}(x) - \mu_{\tilde{B}}(x))\}$
- $\tilde{A} \Delta \tilde{B} = \{(x, \mu_{\tilde{A}\Delta\tilde{B}}(x)) \,|\, x \in X \wedge \mu_{\tilde{A}\Delta\tilde{B}}(x) = |\mu_{\tilde{A}}(x) - \mu_{\tilde{B}}(x)|\}$
- $\tilde{A} \subseteq \tilde{B} \Leftrightarrow \forall x \in X : \mu_{\tilde{A}}(x) \leq \mu_{\tilde{B}}(x)$

Um corte alfa (*alpha-cut*, ou ainda, α-*cut*) e um corte alfa rígido (*strict alpha-cut*, ou ainda, *strict* α-*cut*) de um conjunto *fuzzy* $\tilde{A}$ para um valor específico α é um conjunto *crisp* definido como segue, respectivamente

- $\tilde{A}^{\geq\alpha} = \{x \in X \,|\, \mu_{\tilde{A}}(x) \geq \alpha \wedge 0 \leq \alpha \leq 1\}$
- $\tilde{A}^{>\alpha} = \{x \in X \,|\, \mu_{\tilde{A}}(x) > \alpha \wedge 0 \leq \alpha < 1\}$

Quando α é 1 para o α-*cut* de conjunto *fuzzy* $\tilde{A}$, o resultado é chamado de *core* de $\tilde{A}$ (núcleo de $\tilde{A}$), i.e. $core(\tilde{A}) = \{x \in X \,|\, \mu_{\tilde{A}}(x) = 1\}$. Quando α é 0 para o *strict* α-*cut* de um conjunto *fuzzy* $\tilde{A}$, o resultado é chamado de *support* de $\tilde{A}$ (suporte de $\tilde{A}$), i.e. $supp(\tilde{A}) = \{x \in X \,|\, \mu_{\tilde{A}}(x) > 0\}$.

Generalizações das operações de intersecção e união trocam os operadores *min* e *max* por normas triangulares (*t-norm*) e conormas triangulares (*t-conorm*), respectivamente (KLEMENT; MESIAR; PAP, 2000). Uma t-norm T é definida como uma operação binária comutativa, associativa, não-decrescente em $[1,0]$, com a assinatura $T : [0,1]^2 \to [0,1]$ satisfazendo condições de limite $T(1,x) = x$ e $T(0,x) = 0$ para todo $x \in [0,1]$ (KLEMENT; MESIAR; PAP, 2000). Para qualquer t-norm, existe uma *t-conorm* obtida pela lei De Morgan. Assim, uma t-conorm S é definida como uma operação binária comutativa, associativa, não-decrescente em $[1,0]$, com a assinatura $S : [0,1]^2 \to [0,1]$ satisfazendo as condições de limite $S(1,x) = 1$ e $S(0,x) = x$ para todo $x \in [0,1]$ (KLEMENT; MESIAR; PAP, 2000). Sejam $a,b \in [0,1]$, as t-norms consideradas nesse trabalho são $tnorm = \{T_m, T_l, T_p, T_{nm}, T_{Hp}, T_{Ep}, T^*\}$, definidas como

1. $T_m(a,b) = \min(a,b)$ (*standard intersection*)
2. $T_l(a,b) = \max(0, a+b-1)$ (*Lukasiewicz t-norm*)
3. $T_p(a,b) = ab$ (*product t-norm*)
4. $T_{nm}(a,b) = \begin{cases} \min(a,b) & \text{if } a+b > 1 \\ 0 & \text{otherwise} \end{cases}$ (*nilpotent minimum*)
5. $T_{Hp}(a,b) = \begin{cases} 0 & \text{if } a = 0 \vee b = 0 \\ \frac{ab}{a+b-ab} & \text{otherwise} \end{cases}$ (*Hamacher product*)
6. $T_{Ep}(a,b) = \frac{ab}{2-(a+b-ab)}$ (*Einstein product*)
7. $T^*(a,b) = \begin{cases} a & \text{if } b = 1 \\ b & \text{if } a = 1 \\ 0 & \text{otherwise} \end{cases}$ (*drastic intersection*)

As respectivas t-conorms das t-norms anteriores são definidas a seguir. Sejam $a, b \in [0,1]$, as t-conorms consideradas nesse trabalho são $tconorm = \{S_m, S_{bs}, S_p, S_{nm}, S_{Hs}, S_{Es}, S^*\}$, definidas como

1. $S_m(a,b) = \max(a,b)$ (*standard union*)
2. $S_{bs}(a,b) = \min(1, a+b)$ (*bounded sum*)
3. $S_p(a,b) = a+b-ab$ (*probabilistic sum*)
4. $S_{nm}(a,b) = \begin{cases} \max(a,b) & \text{if } a+b<1 \\ 1 & \text{otherwise} \end{cases}$ (*nilpotent maximum*)
5. $S_{Hs}(a,b) = \frac{a+b-2ab}{1-ab}$ (*Hamacher sum*)
6. $S_{Es}(a,b) = \frac{a+b}{1+(ab)}$ (*Einstein sum*)
7. $S^*(a,b) = \begin{cases} a & \text{if } b=0 \\ b & \text{if } a=0 \\ 1 & \text{otherwise} \end{cases}$ (*drastic union*)

A altura (*height*) de um conjunto *fuzzy* $\tilde{A}$ é definido como $h(\tilde{A}) = sup_x[\mu_{\tilde{A}}(x)]$ (JAMSHIDI; VADIEE; ROSS, 1993). Portanto, a altura é o maior (*supremum*) grau de pertinência de uma função de pertinência de $\tilde{A}$. Um conjunto *fuzzy* $\tilde{A}$ é chamado *normal* quando $h(\tilde{A}) = 1$, e *non-normal* quando $h(\tilde{A}) < 1$. A normalização de um conjunto *fuzzy* $\tilde{A}$ é definido como $Norm_{\mu_{\tilde{A}}}(x) = [\mu\tilde{A}(x)/h(\tilde{A})]$ para todo $x \in X$ (JAMSHIDI; VADIEE; ROSS, 1993).

A concentração (C) de um conjunto *fuzzy* $\tilde{A}$ aumenta a vagueza (i.e. diminui o grau de pertinência de todos os elementos), enquanto a dilatação (D) de um conjunto *fuzzy* $\tilde{A}$ diminui a vagueza (i.e. aumenta o grau de pertinência de todos os elementos) (JAMSHIDI; VADIEE; ROSS, 1993). Estas operações são definidas como

- $\mu_{C(\tilde{A})}(x) = [\mu_{\tilde{A}}(x)]^p$ para todo $x \in X$ onde $p > 1$
- $\mu_{D(\tilde{A})}(x) = [\mu_{\tilde{A}}(x)]^r$ para todo $x \in X$ onde $r \in]0,1[$

Por fim, existem algumas notações para representar textualmente um conjunto *fuzzy* (JAMSHIDI; VADIEE; ROSS, 1993). Seja X o universo, as definições a seguir são possíveis notações textuais de um conjunto *fuzzy* $\tilde{A}$:

1. $\tilde{A} = \sum_{x_i \in X} \mu_{\tilde{A}}(x_i)/x_i$ quando X é finito e discreto

2. $\tilde{A} = \int_x \mu_{\tilde{A}}(x)/x$ quando X é contínuo

É importante notar que os sinais de soma e integral simbolizam a união dos graus de pertinência e a barra (/) simboliza um separador e não realiza uma divisão.

2.4.2.2 Dados Espaciais *Fuzzy*

Existem diversas representações para modelagem de dados espaciais vagos que se baseiam na teoria de conjuntos *fuzzy*. Uma delas é o *Fuzzy Minimum Boundary Rectangle* (FMBR), o qual inclui o uso da teoria de conjuntos *fuzzy* para definir os graus de pertinência de acordo com uma função de pertinência para regiões vagas. Um FMBR é composto por dois ou mais MBRs, dos quais ao menos um representa a parte conhecida e o restante representam a parte incerta. Tais MBRs são formados a partir de regiões *crisp* para descrever a região vaga. A primeira região é chamada de núcleo (parte conhecida) e a segunda chamada de fronteira (parte incerta).

Outros diversos trabalhos adotam dados espaciais *fuzzy* em áreas como a geociência. Primeiramente Altman (1994) demonstrou como aplicar a teoria de conjuntos *fuzzy* em objetos espaciais em uma relação binária no domínio de $\mathbb{N}^2$ ($\mathbb{N}$ denota o conjunto dos números naturais). Schneider (1999) propôs tipos de dados espaciais *fuzzy* como *pontos fuzzy*, *linhas fuzzy* e *regiões fuzzy*, bem como operações geométricas de conjunto *fuzzy* como união geométrica *fuzzy*, intersecção geométrica *fuzzy* e diferença geométrica *fuzzy*. Outros trabalhos apresentaram extensões de objetos espaciais *fuzzy* para criar a partição vaga, como em Dilo, By e Stein (2007).

Esta pesquisa de mestrado usa como base os conceitos e definições de dados espaciais *fuzzy* (SCHNEIDER, 1999, 2008, 2001; DILO, 2006a; DILO; BY; STEIN, 2007), uma vez que é possível utilizar estes tipos de dados para representação de fenômenos do mundo real e manipulá-los por meio de diversas operações, tais como as operações geométricas de conjunto. Pontos *fuzzy* (*fpoint*), linhas *fuzzy* (*fline*) e regiões *fuzzy* são os tipos de dados espaciais *fuzzy*. É importante enfatizar que estes tipos de dados espaciais *fuzzy* são complexos. Intuitivamente, um objeto do tipo ponto *fuzzy* complexo $\tilde{P}$ representa um subconjunto de $\mathbb{R}^2$, onde cada ponto de $\tilde{P}$ tem uma certa pertinência espacial nesse conjunto. Formalmente, um *ponto fuzzy simples* $\tilde{p}$ representado por (a,b) em $\mathbb{R}^2$, definido como $\tilde{p}(a,b)$, é um *fuzzy singleton* em $\mathbb{R}^2$ definido pela função de pertinência $\mu_{\tilde{p}(a,b)}(x,y) = m \in]0,1]$ se $(x,y) = (a,b)$, e $\mu_{\tilde{p}(a,b)}(x,y) = 0$ caso contrário. Seja P_f o conjunto de todos os pontos *fuzzy* simples, e $\tilde{p}(a,b)$, $\tilde{q}(c,d) \in P_f$ com $a,b,c,d \in \mathbb{R}$, além da

disjunção de $\tilde{p}(a,b)$ e $\tilde{q}(c,d)$ que ocorre quando $(a,b) \neq (c,d)$, o tipo espacial *fuzzy fpoint* é definido como

$$\mathit{fpoint} = \{Q \subseteq P_f \,|\, \forall\, \tilde{p}(a,b), \tilde{q}(c,d) \in Q : \tilde{p}(a,b) \text{ e } \tilde{q}(c,d) \text{ são } \mathit{disjuntos} \wedge Q \text{ é finito}\}$$

A disjunção entre os pontos *fuzzy* simples de um *fpoint* é requerido uma vez que o grau de pertinência de cada ponto *fuzzy* simples deve ser único. A Figura 2.7a ilustra um exemplo de um ponto *fuzzy* complexo (*fpoint*) composto por cinco pontos *fuzzy* simples.

Intuitivamente, uma linha *fuzzy* tem o mesmo formato geométrico de uma linha *crisp* (Figura 2.1b). Contudo, cada ponto em sua extensão é associado com um grau de pertinência indicando o quanto um ponto pertence a linha. A função de pertinência de uma linha deve ser contínua, a qual garante que os graus de pertinência entre os pontos mudam continuamente em sua extensão. Formalmente, uma *linha fuzzy simples* $\tilde{l}$ é definida por uma função de pertinência $\mu_{\tilde{l}} : f_{\tilde{l}} \to [0,1]$ com $f_{\tilde{l}} : [0,1] \to \mathbb{R}^2$ tal que

(i) $f_{\tilde{l}}$ é contínua, $\mu_{\tilde{l}}$ é continua
(ii) $\forall a,b \in]0,1[: a \neq b \Rightarrow f_{\tilde{l}}(a) \neq f_{\tilde{l}}(b)$
(iii) $\forall a \in \{0,1\}\ \forall b \in]0,1[: f_{\tilde{l}}(a) \neq f_{\tilde{l}}(b)$
(iv) $f_{\tilde{l}}(0) < f_{\tilde{l}}(1) \vee (f_{\tilde{l}}(0) = f_{\tilde{l}}(1) \wedge \forall a \in]0,1[: f_{\tilde{l}}(0) < f_{\tilde{l}}(a))$

A Condição (i) requer que $f_{\tilde{l}}$ e $\mu_{\tilde{l}}$ sejam funções contínuas. A função contínua $f_{\tilde{l}}$ modela uma linha *crisp* simples, enquanto a função contínua $\mu_{\tilde{l}}$ garante a transição suave de graus de pertinência entre pontos próximos ao longo da extensão da linha simples. Os pontos $f_{\tilde{l}}(0)$ e $f_{\tilde{l}}(1)$ representam os *pontos finais* de $f_{\tilde{l}}$. Condição (ii) permite voltas ($f_{\tilde{l}}(0) = f_{\tilde{l}}(1)$) mas proíbe a igualdade de pontos do interior e assim auto-intersecções. Condição (iii) não permite a igualdade de um ponto do interior com um ponto final. Condição (iv) requer que em uma *linha fuzzy simples fechada*, o ponto $f_{\tilde{l}}(0)$ deve ser mais a esquerda, ou seja, o menor ponto com respeito a ordem lexicográfica $<$. A principal razão das Condições (ii) à (iv) é de garantir a representação única de uma linha *fuzzy* simples.

Baseada nessas condições, uma linha *fuzzy* simples $\tilde{l}$ é dada por um conjunto de pontos *fuzzy* $\tilde{l} = \{(p, \mu_{\tilde{l}}(p)) \,|\, p \in f_{\tilde{l}}([0,1])\}$. Seja SL_f o conjunto de todas as linhas *fuzzy*, e $\tilde{l}_1, \tilde{l}_2 \in SL_f$, é possível definir os predicados *c-disjunto* e *c-toca* como

(i) $\tilde{l}_1$ e $\tilde{l}_2$ são *c-disjunto* $:\Leftrightarrow \mathit{supp}(\tilde{l}_1) \cap \mathit{supp}(\tilde{l}_2) = \varnothing$
(ii) $\tilde{l}_1$ e $\tilde{l}_2$ *c-toca* $:\Leftrightarrow f_{\tilde{l}_1}(]0,1[) \cap f_{\tilde{l}_2}(]0,1[) = \varnothing \wedge$
$\{f_{\tilde{l}_1}(0), f_{\tilde{l}_1}(1)\} \cap \{f_{\tilde{l}_2}(0), f_{\tilde{l}_2}(1)\} \neq \varnothing$

Seja $f_{\tilde{l}_1}, \ldots, f_{\tilde{l}_n} \in SL_f$ para algum $n \in \mathbb{N}$, para todo $1 \leq i, j \leq n$ e para todo $a, k \in \{0,1\}$ então é definido $V^a_{\tilde{l}_i} = \{(j,k) \,|\, f_{\tilde{l}_i}(a) = f_{\tilde{l}_j}(k)\}$. Isso define que $V^a_{\tilde{l}_i}$ registra cada linha *fuzzy* simples com seu respectivo ponto final que é incidente a $\tilde{l}_i$. É importante notar que sempre é assegurado que $(i,a) \in V^a_{\tilde{l}_i}$. Com essas definições, o tipo de dado espacial *fuzzy fline* é definido como

$$\begin{aligned} \mathit{fline} = \{\textstyle\bigcup_{i=1}^{n} \tilde{l}_i \,|\, & n \in \mathbb{N} \wedge \forall\, 1 \leq i \leq n : \tilde{l}_i \in SL_f \wedge \\ & \forall\, 1 \leq i < j \leq n : (\tilde{l}_i \text{ e } \tilde{l}_j \text{ são } \textit{c-disjunto} \vee \tilde{l}_i \text{ e } \tilde{l}_j \textit{ c-toca}) \wedge \\ & \forall\, 1 \leq i \leq n \; \forall a \in \{0,1\} : (|V^a_{\tilde{l}_i}| = 1) \vee (|V^a_{\tilde{l}_i}| > 2)\} \end{aligned}$$

A última condição assegura a representação de unicidade. Se $|V^a_{\tilde{l}_i}| = 2$ fosse permitido, as linhas *fuzzy* simples intersectadas nos pontos finais, poderiam ser juntadas em uma linha *fuzzy* simples. A Figura 2.7b ilustra um exemplo de uma *fline* composta por quatro linhas *fuzzy* simples.

Intuitivamente, uma região *fuzzy* tem o mesmo formato geométrico de uma região *crisp* (Figura 2.1c). Porém, com uma borda vaga ou um interior incerto além de uma possível localização não bem definida. Dessa forma, cada ponto de uma região *fuzzy* é associado com um grau de pertinência indicando o quanto um ponto pertence a região, e uma função de pertinência é requirida para modelar uma transição suave dos graus de pertinências. Formalmente, um conjunto de pontos $\tilde{A}$ em um plano tem uma função de pertinência $\mu_{\tilde{A}} : \mathbb{R}^2 \to [0,1]$. Porém, irregularidades devem ser evitadas, como linhas e pontos isolados bem como linhas e pontos faltando na forma de cortes no interior da região (SCHNEIDER, 1999). O operador *cl* remove cortes adicionando pontos apropriados em seu lugar, sendo definido como $\tilde{A} = cl(int(\tilde{A}))$. Assim, o conjunto *fuzzy* $\tilde{A}$ é chamado como *conjunto regular fuzzy fechado*. Adicionalmente, o operador *int* elimina pontos e linhas pendentes uma vez que seu interior são vazios, sendo definido como $\tilde{A} = int(cl(\tilde{A}))$. Assim, o conjunto *fuzzy* é chamado como *conjunto regular fuzzy aberto*. Um conjunto regular aberto talvez consista de vários componentes desconectados onde cada componente tenha buracos. Aplicações mostram que bordas de regiões *fuzzy* podem ser completamente *fuzzy*, completamente *crisp* ou parcialmente *fuzzy* e *crisp*. Por esse propósito, é definida a fronteira (*frontier*) de um conjunto *fuzzy* $\tilde{A}$ como

$$\mathit{frontier}(\tilde{A}) = \{((x,y), \mu_{\tilde{A}}(x,y)) \,|\, (x,y) \in \mathit{supp}(\tilde{A}) - \mathit{supp}(\mathit{int}(\tilde{A}))\}$$

Em outras palavras, a fronteira de uma região *fuzzy* $\tilde{A}$ é composta por todos os pontos *fuzzy* simples de $\tilde{A}$ que não são pontos do interior da região. Com estas definições, é possível definir o tipo de dado espacial *fuzzy fregion* como

(a) (b) (c)

Figura 2.7: Exemplos de *pontos fuzzy* (*fpoint*) (a), *linhas fuzzy* (*fline*) (b), e uma *região fuzzy* (*fregion*) (c), respectivamente.

$$
\begin{aligned}
\mathit{fregion} = \{\tilde{R} \subseteq \mathbb{R}^2 \times\,]0,1] \mid \\
&\text{(i)} \quad \mathit{int}(\mathit{cl}(\tilde{R})) \subseteq \tilde{R} \subseteq \mathit{cl}(\mathit{int}(\tilde{R})) \\
&\text{(ii)} \quad \mathit{frontier}(\tilde{R}) \subseteq \mathit{frontier}(\mathit{cl}(\mathit{int}(\tilde{R}))) \\
&\text{(iii)} \quad \mathit{frontier}(\tilde{R}) \in \mathit{fline} \\
&\text{(iv)} \quad \mu_{\tilde{R}} \text{ é uma função contínua por partes}\}
\end{aligned}
$$

A Condição (i) define o intervalo possível de um conjunto de pontos de $\tilde{R}$ entre seu conjunto regular *fuzzy* aberto e seu conjunto regular *fuzzy* fechado. Em particular, isso assegura que o interior de $\tilde{R}$ esteja livres de anomalias geométricas. A Condição (ii) corresponde ao conceito de "fronteira parcial", onde um extremo é que $\tilde{R}$ não tenha fronteira ($\mathit{frontier}(\tilde{R}) = \varnothing$) e outro extremo é que $\tilde{R}$ tenha uma fronteira completa ($\mathit{frontier}(\tilde{R}) = \mathit{frontier}(\mathit{cl}(\mathit{int}(\tilde{R})))$). No último caso, todos os graus de pertinência são iguais a 1, correspondendo assim a uma linha *crisp* (ou seja, uma fronteira *crisp*). A Condição (iii) assegura que a fronteira de $\tilde{R}$ é uma linha *fuzzy*. Isso permite fronteiras com partes desconectadas e proíbe pontos *fuzzy* simples. Por fim, a Condição (iv) requer uma distribuição suave dos graus de pertinência com possíveis exceções (ou seja, uma função contínua por partes). A Figura 2.7c mostra um exemplo de um objeto *fregion* contendo apenas uma região *fuzzy* simples.

2.5 Data Warehouse Geográfico

Um *data warehouse* (DW) integra dados oriundos de várias fontes de dados visando auxiliar na tomada de decisão estratégica. Assim constitui-se um banco de dados multidimensional, histórico, volumoso, não volátil e orientado a assunto (KIMBALL; ROSS, 2002; CIFERRI, 2013). Em DW implementados em banco de dados relacionais, medidas numéricas e dimensões são representadas usando-se o esquema estrela (KIMBALL; ROSS, 2002; CIFERRI, 2013), o qual é composto por tabelas de fato e tabelas de dimensão. As tabelas de fato armazenam as medidas numéricas, já as tabelas de dimensão contêm os atributos descritivos que contextualizam essas medidas. Na Figura 2.8 é mostrado o esquema estrela de uma aplicação de varejo, com a tabela de fato *LineOrder* e as tabelas de dimensão *Customer*, *Supplier*, *Part* e *Date* (O'NEIL, 2009).

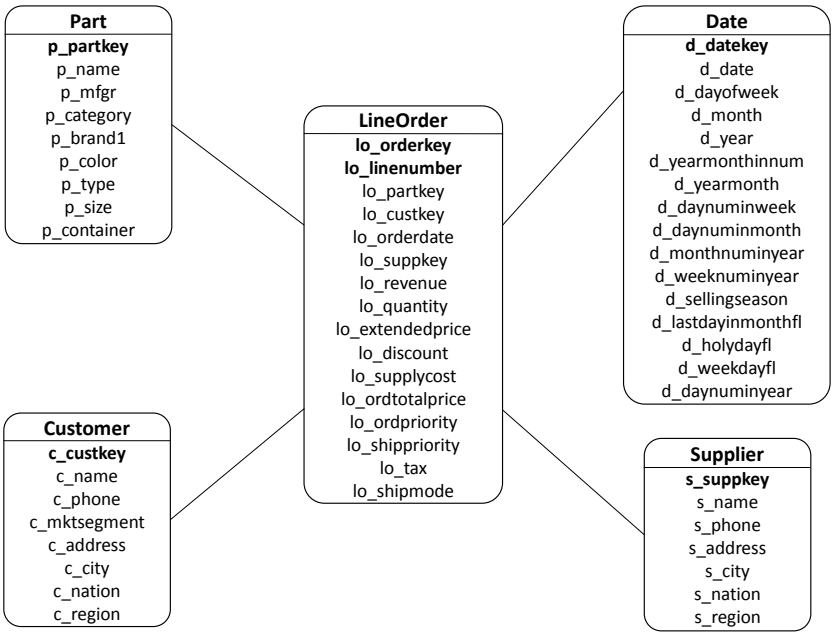


Figura 2.8: Um esquema estrela de uma aplicação de varejo (Adaptada de O'Neil (2009)).

Frequentemente, atributos de uma tabela de dimensão são relacionados entre si por meio de *hierarquias de atributos*, as quais especificam diferentes níveis de agregação e, consequentemente, diferentes granularidades. Por exemplo, na tabela de dimensão *Customer* da Figura 2.8, existe a hierarquia de atributos $(c\_region) \preceq (c\_nation) \preceq (c\_city) \preceq (c\_address)$. As hierarquias de atributos são a base para a organização dos dados do DW em níveis de agregação, permitindo que consultas multidimensionais *On-line Analytical Processing* (OLAP) sejam realizadas. Tais consultas OLAP são processadas por ferramentas OLAP (CARNIEL; SIQUEIRA, 2012) e incluem consultas *drill-down*, *roll-up*, *slice-and-dice*, *pivoting* e *drill-across* (KIMBALL; ROSS, 2002; CIFERRI, 2013). Enquanto consultas *drill-down* analisam os dados em níveis de agregação progressivamente mais detalhados, consultas *roll-up* investigam os dados em níveis de agregação progressivamente menos detalhados. A operação *slice-and-dice* permite que usuários restrinjam os dados sendo analisados a um subconjunto desses dados. Diferentes perspectivas dos mesmos dados podem ser obtidas pela operação *pivoting*, a qual reorienta a visão multidimensional dos dados, ou seja, modifica a ordem de exibição das dimensões. Por fim, consultas *drill-across* são consultas que comparam medidas numéricas de esquemas estrela distintos que são relacionadas entre si por uma ou mais dimensões em comum.

No esquema estrela, os atributos de uma hierarquia são mantidos na mesma tabela de dimensão, gerando redundância de dados. O esquema floco de neve evita essa redundância normalizando a hierarquia de atributos, porém, acrescenta custos adicionais de junções entre tabelas no processamento de consultas OLAP (KIMBALL; ROSS, 2002).

Um *data warehouse geográfico* (DWG) difere de um DW convencional por armazenar adicionalmente dados espaciais como atributos específicos em tabelas de dimensão ou como medidas em tabelas de fato (STEFANOVIC; HAN; KOPERSKI, 2000; MALINOWSKI; ZIMÁNYI, 2008; MATEUS, 2010). Em DWGs, hierarquias de atributos podem ser definidas também sobre atributos espaciais de uma ou mais tabelas de dimensão espacial. Uma hierarquia espacial predefinida é uma associação 1:N ou M:N entre atributos espaciais de mais alta e mais baixa granularidade, a qual é determinada por um relacionamento topológico (por exemplo, pelo relacionamento espacial "está contido") (MALINOWSKI; ZIMÁNYI, 2008).

Na Figura 2.9 é ilustrado um esquema híbrido de DWG (MATEUS, 2010), derivado do *benchmark* Spadawan (SIQUEIRA, 2010). Esse esquema difere de um esquema estrela convencional por incluir tabelas de dimensão espaciais. *Customer*, *Supplier*, *Part* e *Date* são tabelas de dimensão convencionais que armazenam somente dados convencionais (por exemplo, dados descritivos) que são redundantes, enquanto que *C_Address*, *S_Address*, *City*, *Nation* e *Region* são tabelas de dimensão espaciais que são armazenadas separadamente, com base no nível de granularidade, visando evitar redundância dos dados espaciais. Nas tabelas de dimensão espaciais, atributos com o sufixo *_geo*, como *c_address_geo*, são atributos espaciais que armazenam geometrias. Note que os esquemas híbridos e floco de neve são diferentes, uma vez que o primeiro não normaliza as hierarquias de atributos convencionais. O esquema híbrido evita a redundância de dados espaciais em suas tabelas de dimensão, desde que é reconhecido que a redundância de dados espaciais em DWG afeta negativamente o desempenho no processamento de consultas *Spatial* OLAP (SOLAP), e implica em maiores requisitos de espaço de armazenamento (SIQUEIRA, 2008, 2009; MATEUS, 2010). Consultas SOLAP estendem consultas OLAP com predicados espaciais, como *intersects*, *inside* e *contains*.

Adicionalmente, DWGs viabilizam consultas espaciais como a consulta por abrangência (*range query*), a qual recupera todos os objetos que satisfazem certo relacionamento topológico com um retângulo chamado *janela de consulta* (GAEDE; GÜNTHER, 1998), e como a consulta aos k-vizinhos mais próximos, a qual retorna os k objetos mais próximos de um determinado objeto espacial (MOHAN, 2008).

Em adição às características discutidas anteriormente de DWGs, os atributos espaciais armazenados podem ter características de dados espaciais vagos. O modelo *Vague Spatial Cube* define como estes atributos espaciais vagos podem ser modelados conceitualmente em um DWG, constituindo assim um DWG vago (SIQUEIRA, 2014). Em geral, dimensões e fatos podem adicionalmente armazenar dados espaciais vagos, seguindo os modelos exatos (Seção 2.4.1) ou modelos *fuzzy* (Seção 2.4.2). Além disso, as hierarquias espaciais são estendidas para a

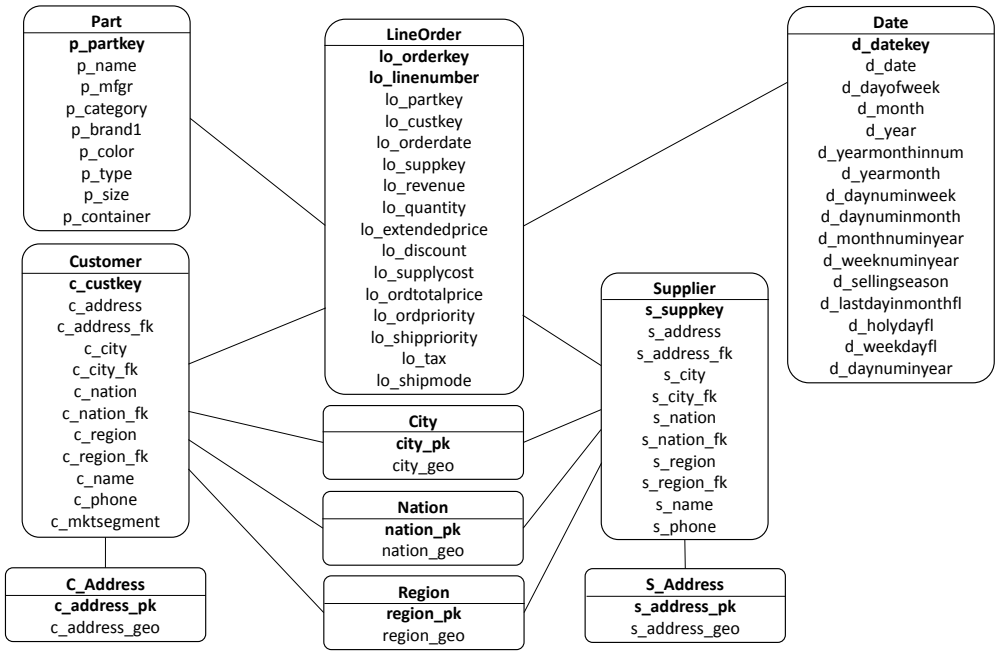


Figura 2.9: Um esquema híbrido de uma aplicação de varejo com atributos espaciais (Adaptada de Mateus (2010)).

manipulação de dados espaciais vagos. Por exemplo, uma hierarquia espacial pode ser *completamente vaga* (formada por atributos espaciais vagos), *híbrida* (formada por atributos convencionais, espaciais *crisp* e espaciais vagos), *completamente espacial* (formada por atributos espaciais vagos ou *crisp*), *parcialmente vaga* (onde ao menos um atributo é espacial vago) e *completamente crisp* (formada apenas por atributos espaciais *crisp*). Por fim, operações para manipular dados espaciais vagos em DWG vagos são também definidos, tais como agregações espaciais vagas e janelas de consultas sobre atributos espaciais vagos.

Apesar de existirem propostas de armazenamento de dados espaciais vagos em DWGs (Seção 3.4 do Capítulo 3), a principal limitação do estado da arte, é a inexistência de um TAD que manipule os atributos espaciais vagos em um DWG vago. A principal vantagem do uso de um TAD é o armazenamento de dados espaciais vagos em estruturas únicas e a sua manipulação por meio de operações de uma forma mais simples, escondendo a complexidade dessas operações para o usuário final. Além disso, tornar as consultas analíticas mais claras e enxutas para o usuário final. Outro ganho potencial é em relação ao desempenho nas operações, devido a utilização de estruturas internas próprias para o seu armazenamento. Esta pesquisa de mestrado enfoca nesta lacuna, ou seja, na definição de um TAD, denominado VagueGeometry, para permitir o armazenamento e uso de dados espaciais vagos em banco de dados espaciais bem como em DWGs.

2.6 Considerações Finais

Neste capítulo foram discutidos vários conceitos relacionados a banco de dados espaciais, modelos de dados espaciais vagos e seu uso em *data warehouses geográficos*. Dentre os modelos de dados espaciais vagos existentes, foram abordados principalmente os modelos exatos e os modelos *fuzzy*. Com isso, foram destacados os desafios para a representação da *vagueza espacial* e a definição de um TAD para o uso em DWGs. Esses desafios foram enfocados durante o desenvolvimento do mestrado.

Em particular, o modelo considerado para a implementação dos dados espaciais vagos considerado no desenvolvimento deste mestrado foi o modelo exato VASA, uma vez que é a mais completa e define de forma mais consistente os dados espaciais vagos (os quais podem ser simples e complexos). Tal modelo é mais completo que os demais por definir diversas operações que são importantes para manipulação de objetos espaciais em geral (GÜTING, 1994), tais como predicados topológicos vagos, operações geométricas de conjunto e operações numéricas. Além disso, como resultado adicional foi considerado o estudo da teoria de conjuntos *fuzzy* para representação da vagueza espacial, por meio dos tipos de dados espaciais *fuzzy*.

No próximo capítulo, Capítulo 3, são resumidos trabalhos correlatos à presente proposta, a qual visa discutir sobre implementações de dados espaciais existentes no SGBD PostgreSQL, TADs, uso de dados espaciais vagos em DWGs e implementações de dados espaciais vagos em banco de dados espaciais. Ao longo do capítulo também são destacadas limitações desses trabalhos e diferenciais da proposta da dissertação de mestrado.

Capítulo 3

TRABALHOS CORRELATOS

Este capítulo resume os trabalhos relacionados à presente dissertação de mestrado. O capítulo também discute as limitações existentes nos trabalhos correlatos e as motivações do desenvolvimento deste trabalho.

3.1 Considerações Iniciais

Trabalhos existentes voltados à incorporação de dados espaciais vagos em DWGs incluem três diferentes perspectivas. A primeira delas, detalhada nas Seções 3.2 e 3.3, refere-se às funcionalidades oferecidas respectivamente pelo SGBD PostgreSQL com a extensão espacial PostGIS (POSTGIS..., 2014) e pelo iBLOB (CHEN, 2010). Em particular, o iBLOB é uma proposta de um *framework* para desenvolvimento de TADs genéricos. A segunda perspectiva diz respeito aos trabalhos que investigam aspectos relacionados ao armazenamento de dados espaciais vagos no modelo lógico relacional de um DWG (SIQUEIRA, 2011, 2012), e são resumidos na Seção 3.4. A última perspectiva refere-se aos poucos trabalhos na literatura que implementam dados espaciais vagos (KRAIPEERAPUN, 2004; DILO, 2006b; ZINN; BOSCH; GERTZ, 2007; PAULY; SCHNEIDER, 2008). Esses trabalhos são resumidos na Seção 3.5. Em cada uma das seções, são destacadas as limitações dos trabalhos correlatos e discutidas as justificativas do desenvolvimento desta pesquisa de mestrado. O capítulo é finalizado na Seção 3.6 com as considerações finais e uma tabela que compara as implementações de dados espaciais vagos existentes com a proposta desenvolvida neste mestrado.

3.2 Funcionalidades Oferecidas por Bancos de Dados Espaciais

Como funcionalidades espaciais baseadas no padrão OGC (OPEN..., 2014), o SGBD PostgreSQL com a extensão espacial PostGIS (POSTGIS..., 2014) provê suporte aos tipos de dados espaciais *crisp*, os quais podem ser simples, tais como ponto, linha e polígono (i.e., região simples), ou complexos, tais como, multiponto, multilinha e multipolígono (i.e., região complexa). Esses tipos de dados espaciais *crisp* podem ser usados na criação de atributos espaciais em tabelas relacionais e podem ser manipulados em consultas, tais como em predicados topológicos em consultas SQL. Apesar de existirem outras extensões espaciais para outros SGBDs, tais como o *Oracle Spatial* para o SGBD Oracle (ORACLE..., 2014), *DB2 Spatial Extender* para o SGBD DB2 (DB2..., 2014) e *Spatial Extensions* para o MySQL (SPATIAL..., 2014), o SGBD PostgreSQL com a extensão espacial PostGIS (PostgreSQL/PostGIS) é considerada neste projeto por ser de código fonte aberto, gratuito e amplamente usado em pesquisas científicas e na indústria. Exemplos de aplicações que utilizam o PostgreSQL/PostGIS podem ser encontradas em (DELIPETREV; JONOSKI; SOLOMATINE, 2014; GKATZOFLIAS; MELLIOS; SAMARAS, 2013; CORTI, 2014).

As operações da extensão espacial PostGIS são compostas por funções de gerenciamento de dados espaciais que podem construir, acessar, editar e executar operações espaciais, tais como predicados topológicos e de operações geométricas de conjuntos. A construção de objetos espaciais pode ser feita por meio de forma textual, binária ou por funções específicas para construir objetos específicos. Para a forma textual de objetos espaciais pode ser utilizada a representação *Well-known Text* (WKT), enquanto para a forma binária de objetos espaciais é utilizada a representação *Well-known Binary* (WKB). Adicionalmente, o PostGIS inclui as representações *Extended*-WKT (EWKT) e *Extended*-WKB (EWKB), os quais incluem em sua representação o *Spatial Reference System Identifier* (SRID) em suas formas textuais e binárias, respectivamente. Um SRID é um valor numérico único que define o sistema de coordenadas de um objeto espacial. Representações textuais baseados em *eXtensible Markup Language* (XML), tais como o *Geography Markup Language* (GML) e *Keyhole Markup Language* (KML), e baseados em *JavaScript Object Notation* (JSON), tal como o *Geographic JSON* (GeoJSON), também podem ser usados na criação de objetos espaciais *crisp* no PostGIS. Todas estas representações são definidas pela OGC (OPEN..., 2014).

Por exemplo, pode-se definir um ponto com coordenadas $(10.3, 50.4)$ por meio da representação textual WKT como POINT(10.3 50.4) ou por meio da função específica *ST_MakePoint(10.3, 50.4)*. O acesso aos objetos espaciais é realizado por funções específicas que

exploram características dos objetos espaciais ou capturam suas informações. Por exemplo, a função *ST_X* retorna a coordenada *x* de um ponto passado por parâmetro. A edição de objetos espaciais, por sua vez, também pode ser feita de forma semelhante aos métodos assessores. Por fim, o PostGIS também define funções de predicados topológicos como, por exemplo, *intersects* (*ST_Intersects*). Adicionalmente, há a função *ST_Relate* a qual extrai a matriz DE-9IM (Seção 2.3 do Capítulo 2) entre dois objetos espaciais.

Apesar de oferecer essas funcionalidades, o SGBD PostgreSQL/PostGIS não incorpora um tipo de dado específico para manipular dados espaciais vagos. Consequentemente, a complexidade da representação de dados espaciais vagos reutilizando os tipos de dados do PostGIS pode ser alta, causando perdas tanto na legibilidade da consulta quanto no desempenho no processamento das operações. Surge, então, a necessidade de se especificar um TAD próprio para armazenar e permitir a consulta aos dados espaciais vagos. Esta pesquisa de mestrado visa preencher essa lacuna, por meio da proposta do TAD VagueGeometry.

3.3 Armazenamento Inteligente de Grandes Objetos Binários: iBLOB

No trabalho de Chen (2010), é proposta a estrutura denominada *Intelligence Binary Large Objects* (iBLOB) para manipular e armazenar objetos complexos. O principal foco do referido trabalho é permitir que se criem estruturas específicas de dados para determinadas aplicações, conforme a necessidade do usuário.

Na Figura 3.1 são mostradas duas abordagens já existentes e discutidas nesse trabalho correlato e a abordagem do iBLOB. Na Figura 3.1a é mostrada a abordagem de arquitetura em camadas, na qual cada TAD é definido como uma camada individual claramente separada da aplicação final e que usa o SGBD para armazenar os dados separados em tabelas relacionadas. O lado negativo dessa abordagem é que o SGBD não compreende o dado complexo que está sendo armazenado, limitando as operações e dependendo das funcionalidades providas pelo *middleware*, causando atrasos no processamento dos objetos complexos (CHEN, 2010).

Já na Figura 3.1b é mostrada uma abordagem amplamente adotada, a exploração da extensibilidade do SGBD, definindo TADs específicos internamente no SGBD, tal como a apresentada na Seção 2.2 do Capítulo 2. Dessa forma, uma arquitetura integrada é formada, e as aplicações acessam diretamente o SGBD estendido por meio dos TADs definidos como atributos em tabelas relacionais. A vantagem introduzida por essa abordagem refere-se ao desempenho provido no acesso de aplicações externas para a utilização dos TADs. Porém, essa abordagem depende

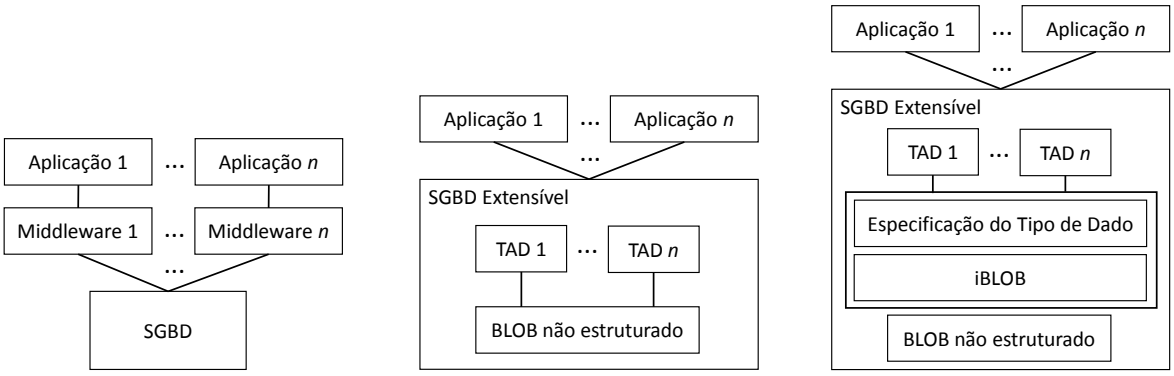


Figura 3.1: (a) Arquitetura baseada em camadas (*middleware*), (b) arquitetura integrada ao SGBD e (c) arquitetura do iBLOB (CHEN, 2010) (Adaptada de Chen (2010)).

da definição da estrutura de armazenamento binária de forma não estruturada (BLOB -*Binary Large Objects*), ou seja, serializada em disco. Essa manipulação de baixo nível pode degradar o desenvolvimento de TADs, pois para um desenvolvedor de um TAD é mais relevante o desenvolvimento de algoritmos eficientes para as suas operações de alto nível do que operações em nível binário e serializado.

Em Chen (2010) é proposta a abordagem ilustrada na Figura 3.1c, incluindo a definição da especificação de um tipo de dado e a forma de seu armazenamento (iBLOB) internamente no SGBD.

Para definir um TAD, por meio do iBLOB, é necessária a definição de sua estrutura hierárquica por meio de uma gramática que define todos os seus elementos que compõe o tipo de dado complexo. Dessa forma, um objeto complexo é criado de acordo com a sua gramática, a qual é armazenada junto a ele. Na Figura 3.2b é mostrado um exemplo de uma gramática para definir uma região (polígono) como TAD, baseando-se em sua estrutura hierárquica ilustrada na Figura 3.2a. O armazenamento é feito por meio de um vetor que mantém de forma sequenciada os subobjetos do tipo de dado. Para acessar e manipular esse vetor, foram definidas as categorias de funcionalidades da seguinte forma: (i) *construção e duplicação*, as quais são operações para construir objetos ou replicá-los; (ii) *referência interna*, a qual é responsável pela recuperação de subobjetos do vetor por meio de índices; (iii) *leitura e escrita*, as quais são utilizadas para inserir, recuperar e excluir subobjetos do vetor usando referências internas; e, (iv) *operações de manutenção e de propriedades do vetor*, as quais podem ser usadas para determinar, por exemplo, o tamanho do vetor em *bytes* e o número de objetos contidos nesse vetor.

Apesar de Chen (2010) não usar o iBLOB para definir objetos espaciais vagos, e sim somente para definir objetos espaciais *crisp*, o iBLOB é um exemplo de extensão de TADs

(a) (b)

Figura 3.2: Exemplo de definição de uma gramática para definir o tipo de dado *região*. (Adaptada de Chen (2010)).

genéricos. Entretanto, por ser genérico, o iBLOB pode ter problemas de eficiência relacionados à manipulação de grandes quantidades de dados, quando comparado com soluções existentes usando TADs específicos em um SGBD (Figura 3.1b). Isso pode ser causado, por exemplo, devido ao carregamento da gramática associada ao objeto quando seu acesso é realizado. Em especial, a questão de desempenho do iBLOB não foi investigada exaustivamente em Chen (2010), embora o aspecto de desempenho seja importante ao processar operações em um DWG e impacte diretamente nas operações para a tomada de decisão. De forma contrária à proposta do iBLOB, esta pesquisa de mestrado define um TAD para representar dados vagos, usando a arquitetura da Figura 3.1b, devido ao seu desempenho. Além disso, também é investigada a eficiência do TAD proposto no processamento de consultas SOLAP com dados espaciais vagos.

Outra diferença relacionada a presente dissertação refere-se ao fato de que, no trabalho do iBLOB, foi definida apenas uma gramática para dados espaciais *crisp* e seus operadores. De forma contrária, esta pesquisa foca na definição de um TAD para dados espaciais vagos bem como suas operações. Por fim, apesar da proposta do iBLOB ser genérica e independente de SGBD, ele somente foi implementado para o SGBD Oracle.

3.4 Aspectos Relacionados ao Armazenamento de Dados Espaciais Vagos

Siqueira (2011, 2012) investigaram o armazenamento de dados espaciais vagos no modelo lógico relacional de um DWG. Em Siqueira (2011), foi demonstrado que o uso dos esquemas tradicionais de representação de dados espaciais, como o esquema híbrido baseado no esquema estrela, pode degenerar o desempenho no processamento de consultas SOLAP sobre dados espaciais vagos. Testes de desempenho foram efetuados para investigar o impacto de manter dados espaciais vagos em uma única dimensão ou de separá-los em outra tabela. Foram investigados os tipos de dados espaciais vagos pontos e regiões. Os pontos vagos foram implementados como multipontos no SGBD PostgreSQL/PostGIS, enquanto as regiões vagas foram implementadas

usando o modelo Egg-Yolk e multipolígonos (Seção 2.4.1 do Capítulo 2).

Ademais, os experimentos usaram janelas de consultas *crisp* e janelas de consulta vagas para processar consultas SOLAP sobre regiões vagas. Com relação às janelas de consultas vagas, elas foram usadas para identificar todos os objetos espaciais vagos com localizações indefinidas que satisfaziam dois, ou mais, relacionamentos topológicos em relação a duas, ou mais, janelas, respectivamente. Enquanto o primeiro relacionamento topológico *inside* foi especificado para o retângulo interno com uma maior seletividade e consequentemente com um menor grau de incerteza nos resultados, o segundo relacionamento de *intersects* foi especificado para retângulos externos, os quais são menos restritivos e, portanto, indicam um maior grau de incerteza.

Já em Siqueira (2012) foram propostos esquemas específicos de DW para permitir o armazenamento de dados espaciais vagos a partir da implementação de dados espaciais *crisp* em SGBDs baseados em modelos relacionais. Os atributos espaciais foram classificados em convencionais, espaciais *crisp* e espaciais vagos. Quanto aos atributos espaciais vagos, eles foram definidos com base nos modelos QMM e VASA (Seção 2.4.1 do Capítulo 2). Além disso, foram definidas hierarquias que envolvem dados espaciais vagos.

Um estudo de caso sobre um controle de pesticidas foi apresentado para exemplificar os conceitos definidos. Na Figura 3.3 é mostrado o DWG vago proposto para controle de pesticidas em plantações. A tabela de fato *LineOrder* mantém chaves estrangeiras para as dimensões, que podem ser espaciais e/ou convencionais. Ademais, existem 6 tabelas de dimensão espacial (*Supplier*, *Plantation*, *AppliedArea*, *IrrigationChannel*, *CropParcel* e *RuralArea*) representadas por atributos espaciais que têm o sufixo *_geo*. Dessas dimensões, 3 tabelas de dimensão armazenam dados espaciais vagos (*Plantation*, *CropParcel* e *AppliedArea*). Juntamente com as tabelas de dimensão, são representados os tipos de dados espaciais dos objetos daquela dimensão, usando a notação gráfica introduzida em Malinowski e Zimányi (2008). Por exemplo, os endereços da tabela de dimensão *Supplier* são representados por pontos, os canais de irrigação da tabela de dimensão *IrrigationChannel* são representados por linhas e as áreas de plantação da tabela *Plantation* são representadas por polígonos. O tipo de dado utilizado para implementar os atributos espaciais vagos foi o multipolígono do SGBD PostgreSQL/PostGIS.

Um outro tipo de armazenamento de dados espaciais vagos também foi proposto em Siqueira (2012). Este armazenamento separa os dados espaciais vagos em duas partes, o *núcleo* e a parte *duvidosa*. Enquanto o núcleo se refere a parte conhecida e bem definida de um objeto espacial vago, a parte duvidosa se refere a parte que apresenta vagueza espacial. O armazenamento separado é, portanto, baseado nos modelos exatos discutidos na Seção 2.4.2 do

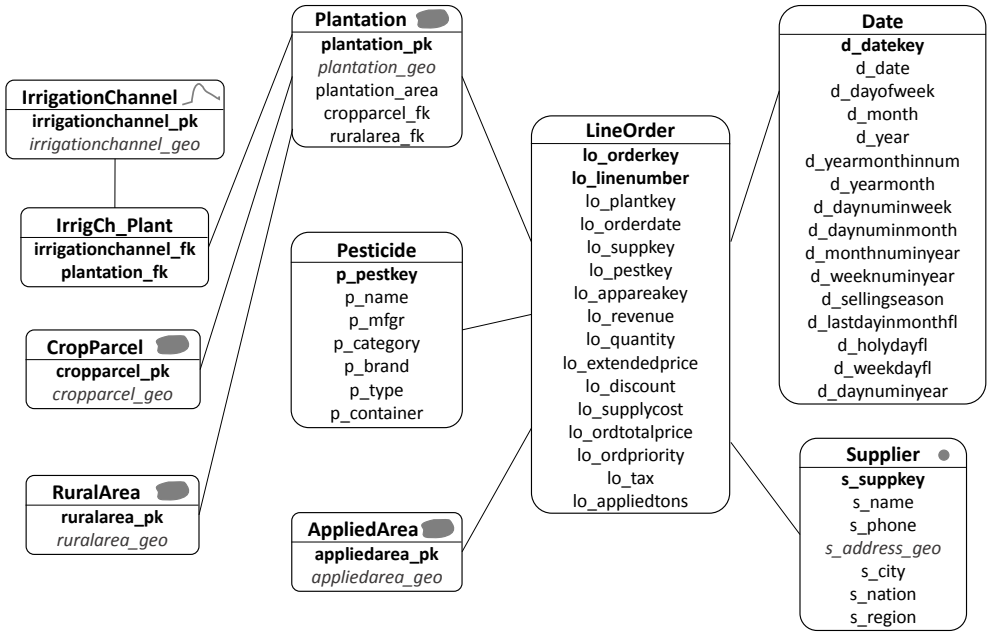


Figura 3.3: Um exemplo de DWG vago para controle de pesticidas sobre plantações (Adaptada de Siqueira (2012)).

Capítulo 2. A separação ocorre nas tabelas de dimensão que contém dados espaciais vagos, assim a parte do núcleo é armazenada em uma tabela enquanto a parte duvidosa em outra. Tais tabelas mantém chaves estrangeiras para a tabela de dimensão. Dessa forma, é possível executar consultas SOLAP considerando apenas a parte duvidosa ou a parte do núcleo. Na Figura 3.4 é mostrado como os dados espaciais vagos da tabela de dimensão *AppliedArea* (Figura 3.3) são separados. Nesta figura, a tabela *AppliedAreaCore* se refere ao núcleo, enquanto a tabela *AppliedAreaDubiety* se refere a parte duvidosa.

Adicionalmente, a parte duvidosa pode manter em uma coluna separada, o grau de pertinência de cada objeto espacial em um intervalo real $]0,1[$. Assim, os dados espaciais vagos são baseados nos modelos *fuzzy* (Seção 2.4.2 do Capítulo 2). Na Figura 3.5 é mostrada a tabela *AppliedAreaDubietyFuzzy*, que contém os graus de pertinência da parte duvidosa dos objetos espaciais vagos. Ainda nesta figura, a parte do núcleo, armazenada na tabela *AppliedAreaCore*, corresponde a parte espacial com grau de pertinência igual a 1. Tais tabelas mantêm chaves estrangeiras para a tabela de dimensão *AppliedArea*. Como resultado, é possível restringir áreas incertas que contêm um grau de pertinência de interesse. Por exemplo, para selecionar todas as áreas nas quais pesticidas foram aplicados com no mínimo um grau de certeza de 50%, define-se uma janela de consulta baseada no relacionamento topológico *intersects* sobre o atributo espacial *dubiety_geo* da tabela *AppliedAreaDubietyFuzzy*, de forma que sejam retornadas somente regiões que tenham pelo menos o grau de pertinência de 0,5 (i.e., $dubiety\_fuzzy >= 0,5$).

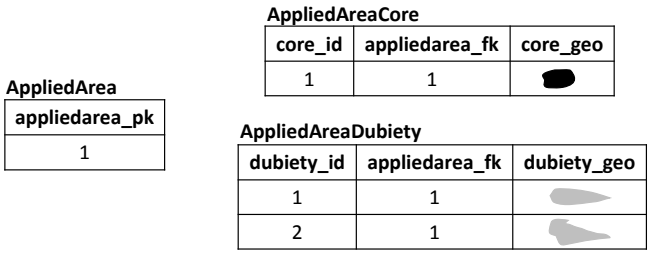


Figura 3.4: Armazenamento separado das partes do núcleo e duvidosa dos dados espaciais vagos contidos em uma tabela de dimensão, baseando-se nos modelos exatos (Adaptada de Siqueira (2012)).

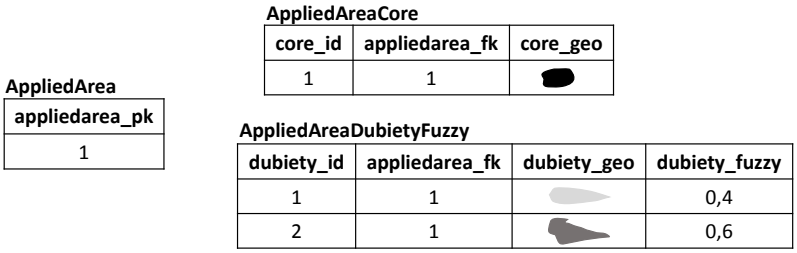


Figura 3.5: Armazenamento separado das partes do núcleo e duvidosa dos dados espaciais vagos contidos em uma tabela de dimensão, baseando-se nos modelos *fuzzy* (Adaptada de Siqueira (2012)).

Dentre as limitações dos trabalhos de Siqueira (2011, 2012), pode-se destacar que eles apenas reutilizam estruturas já existentes no SGBD PostgreSQL/PostGIS na tentativa de incorporar dados espaciais vagos em DWGs e assim processar consultas SOLAP. Além disso, existe a separação de objetos espaciais vagos em tabelas que representam a parte conhecida e a parte duvidosa. Isso pode adicionar mais tempo de processamento por armazenar partes espaciais exatas e incertas em tabelas distintas, uma vez que inclui custos de junções. Ademais, os tipos de dados espaciais vagos e as janelas de consulta vagas não foram implementados diretamente no SGBD como uma extensão espacial. Por fim, os relacionamentos topológicos usados nas consultas foram *crisp* e não vagos, tais como os propostos na VASA (PAULY; SCHNEIDER, 2010). Suprir essas limitações é o objetivo desta pesquisa de mestrado.

3.5 Implementação de Dados Espaciais Vagos

Existem poucos trabalhos na literatura que implementam dados espaciais vagos. Nessa seção são discutidos os trabalhos de Kraipeerapun (2004), Dilo (2006b) (Seção 3.5.2), Pauly e Schneider (2008) (Seção 3.5.1) e Zinn, Bosch e Gertz (2007) (Seção 3.5.3).

3.5.1 Implementação dos Dados Espaciais Vagos da VASA

Pauly e Schneider (2008) apresentam uma implementação de dados espaciais vagos que permite consulta e manipulação de dados espaciais vagos baseados no modelo exato VASA. Esta implementação foi realizada no SGBD Oracle[1]. A definição de um tipo de dado espacial vago é realizada por meio de um par de objetos *crisp* disjuntos ou adjacentes, sendo que esses objetos devem ser do mesmo tipo de dado, de acordo com as especificações da VASA (Seção 2.4.1).

Tais definições foram realizadas no nível mais alto do SGBD, sem modificação de sua estrutura interna, da seguinte forma. Os autores utilizaram a linguagem SQL para definir os dados espaciais vagos, e implementaram os predicados espaciais (por exemplo, *contains*) da seguinte forma. Para cada predicado topológico P, foram implementadas três funções específicas. Sejam A e B, objetos espaciais vagos, as três funções específicas são definidas como: (i) $true\_P(A,B)$, a qual retorna *true* se e somente se o predicado é verdadeiro e *false* caso contrário; (ii) $maybe\_P(A,B)$, a qual retorna *true* se e somente se o predicado talvez aconteça (i.e. retorna *maybe* como resultado) e *false* caso contrário; e (iii) $false\_P(A,B)$, a qual retorna *true* se e somente se o predicado é falso e *false* caso contrário. Essas funções são necessárias uma vez que os predicados espaciais da álgebra VASA podem retornar 3 valores lógicos: *true*, *false* ou *maybe*. A adaptação realizada garantiu o processamento do predicado topológico P sobre dois dados espaciais vagos A e B retornando um dos três valores lógicos da álgebra VASA.

Ainda em Pauly e Schneider (2008) é proposto o operador $\sim$ junto a um relacionamento topológico P, para a manipulação de dados espaciais vagos. Por exemplo, uma operação de *overlap* (i.e. sobreposição) entre dois dados espaciais vagos A e B é representada na forma $\sim overlap(A,B)$. Essa operação retornará *true* se, e somente se, o resultado da sobreposição com certeza ocorrer ou talvez ocorrer (i.e., se o resultado do predicado for igual a *true* ou *maybe*). Caso o predicado *overlap* seja chamado sem o operador $\sim$, ou seja, $overlap(A,B)$, ele somente retornará *true* se a sobreposição com certeza ocorrer (i.e., *true*). Além disso, este mesmo operador pode ser usado para manipular os resultados de operações numéricas, tais como a área de uma região vaga. Dessa forma, $\sim$ é um operador binário, onde o primeiro elemento é o resultado de uma operação numérica envolvendo objetos espaciais vagos (i.e., um valor mínimo e máximo) enquanto o segundo elemento é um valor numérico. Por exemplo, o operador $\sim$ aplicado à operação numérica *area* considerando o valor 50, ou seja, $area(A,B) \sim 50$, retornará *true* se, e somente se, o valor 50 estiver entre o valor mínimo e o valor máximo retornados pela operação numérica *area*. Contudo, o operador $\sim$ não foi implementado devido às necessidades de manipulação dos resultados dos predicados topológicos vagos e operações

[1] http://www.cise.ufl.edu/research/SpaceTimeUncertainty/

numéricas vagas na linguagem de consulta do SGBD e adaptações na linguagem SQL.

Diferentemente desta pesquisa de mestrado, o trabalho de Pauly e Schneider (2008) não proporciona uma forma de representar os dados espaciais vagos internamente no SGBD de forma compacta, apenas oferecendo uma camada que adapta os operadores da álgebra para permitir o tratamento de relacionamentos topológicos com três valores lógicos. Ademais, esse trabalho correlato também não altera o processamento de consultas espaciais vagos no SGBD para garantir o processamento dos predicados topológicos vagos de acordo com os três valores lógicos possíveis de retorno. Outra principal limitação é de que a implementação foi realizada no SGBD Oracle, sendo portanto de uso restrito a aquisição de licenças do SGBD.

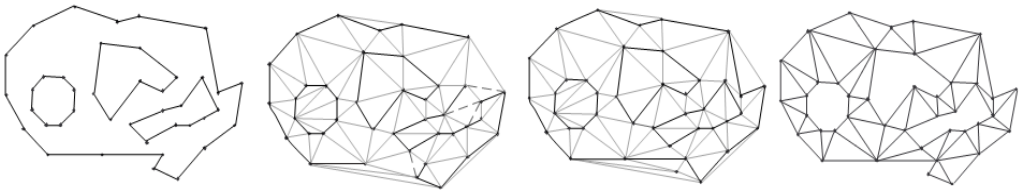
3.5.2 Implementação de Dados Espaciais *Fuzzy*

Kraipeerapun (2004) e Dilo (2006b) implementam estruturas para representar dados espaciais vagos simples para ponto, linha e região baseando-se no modelo *fuzzy*. Um ponto vago (ou *fuzzy*) simples é armazenado como um tripla (x, y, λ), onde $(x, y) \in \mathbb{R}^2$ fornece a localização espacial e $\lambda \in]0, 1]$ o grau de pertinência.

Uma linha vaga (ou *fuzzy*) simples é armazenada como uma sequência finita de triplas $((x_1, y_1, \lambda_1), ..., (x_n, y_n, \lambda_n))$ para algum $n \in \mathbb{N}$, onde cada tripla fornece uma localização (x, y) de um ponto, associado ao grau de pertinência para a linha. Ou seja, uma linha simples vaga é construída por pontos vagos e, usando interpolação linear consecutiva entre dois pontos, é possível calcular o grau de pertinência de um ponto pertencente à linha.

Uma região simples vaga (ou *fuzzy*) consiste em uma fronteira representada por linhas vagas simples e opcionalmente linhas vagas simples para representar buracos. Os pontos da fronteira, representada por uma linha vaga simples, obedecem a uma função de pertinência que decrementa o grau de pertinência de cada ponto da linha. Diversos traçados podem ser definidos para a fronteira vaga, mas ainda tais fronteiras são linhas. Para determinar a informação completa da região vaga simples, ou seja, o grau de pertinência de um determinado ponto da região, foi implementado um método de interpolação baseado em triangulação. O método é realizado em dois passos: construir a triangulação e executar a interpolação dentro de cada triângulo, quando necessário.

Na Figura 3.6 é mostrado o processo de triangulação sobre uma região vaga simples com 3 buracos. A triangulação é composta por 4 fases. Na Figura 3.6a é mostrado o primeiro passo, o qual consiste na definição das linhas vagas simples que compõem os 3 buracos e a fronteira da região vaga simples. Na Figura 3.6b é mostrada a execução da triangulação de Delaunay, usada



(a) (b) (c) (d)

Figura 3.6: Processo para definir uma região simples vaga usando triangulações (Adaptada de Dilo (2006b)).

nesse trabalho. Nessa execução, segmentos que estavam definidos na região podem desaparecer e devem, portanto, ser repostos. Além disso, na reposição dos segmentos podem ocorrer sobreposições em segmentos da triangulação, fazendo com que esses segmentos sobrepostos sejam removidos (linhas pontilhadas na Figura 3.6b). O resultado desse procedimento é mostrado na Figura 3.6c. Por fim, na Figura 3.6d os segmentos da triangulação presentes nos buracos e no exterior da região são removidos. É importante enfatizar que os segmentos da triangulações também são linhas vagas. Pontos vagos podem também ser inseridos no interior da região para melhorar a triangulação e a interpolação, garantindo resultados mais precisos.

A implementação do trabalho de Kraipeerapun (2004) e Dilo (2006b) é realizada utilizando o software GRASS reutilizando estruturas de objetos espaciais existentes. É importante enfatizar que somente os objetos espaciais vagos simples baseados no modelo *fuzzy* e as operações de união, intersecção e diferença (esta operação somente entre regiões e pontos) foram implementados. Assim, os outros operadores de conjuntos espaciais (por exemplo, diferença entre linhas), predicados topológicos (por exemplo, *overlap*) e operadores numéricos (por exemplo, *distância* entre dois objetos espaciais vagos) não foram implementados. Diferentemente, esta pesquisa de mestrado contempla esses aspectos na definição de um TAD para manipular dados espaciais vagos em forma de uma extensão espacial para o SGBD PostgreSQL.

3.5.3 A Extensão para o SGBD PostgreSQL denominada Shapelet

Em Zinn, Bosch e Gertz (2007) é proposta uma extensão do SGBD PostgreSQL para dados espaciais vagos baseado no modelo probabilístico. O novo tipo de dado proposto, chamado de *Shapelet*, é baseado na técnica de composição de imagens desenvolvida na astronomia.

A base do Shapelet é um conjunto de funções que "perturbam" o padrão da função Gaussiana. O primeiro termo da decomposição é a função Gaussiana, e os termos de mais alta ordem envolvem produtos da Gaussiana com um conjunto de polinômios. Por causa do fator

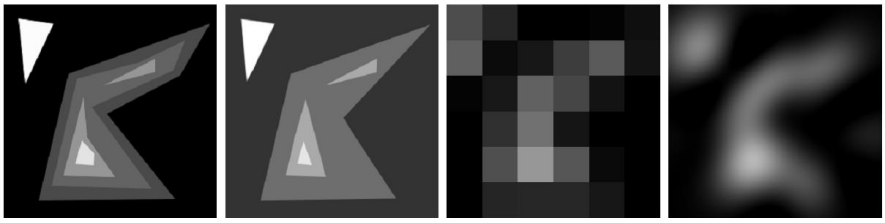


Figura 3.7: Representações de objetos espaciais usando polígonos, *pixels* e *shapelet* (Adaptada de Zinn, Bosch e Gertz (2007)).

de curva das funções Gaussianas, Shapelets podem representar os objetos vagos. Na Figura 3.7 é mostrada uma comparação feita no trabalho com representações usando polígonos, pixels e shapelets de desenhos geométricos.

Na implementação do Shapelet foi definida uma classe principal em C++ que provê as funções para o novo tipo de dado Shapelet ao PostgreSQL. Tais funções foram implementadas usando a linguagem procedural baseada em Python do PostgreSQL. O tipo de dado Shapelet foi implementado por meio de matrizes usando a GNU *Scientific Library* para as operações matriciais. Portanto, o Shapelet foi implementado usando técnicas de *raster*. Além disso, funções para exportação de objetos Shapelet no formato .PNG e sua prototipação foram implementadas em Perl.

As principais rotinas da extensão Shapelet são: (i) *input/output* de conjuntos de caracteres (strings) ASCII e imagens .PNG, respectivamente; (ii) operações aritméticas (adição, subtração, normalização, multiplicação e multiplicação escalar); (iii) integrais e circunvoluções; (iv) mudanças de resolução das imagens; (v) operações de conjuntos (união, intersecção e sobreposição); e (vi) determinação de retângulos envolventes mínimos. A última rotina tem como objetivo a indexação dos Shapelets baseados na estrutura de indexação R-Tree (utilizando o conjunto de índices padrões do PostgreSQL) para diminuir os custos das operações que envolvem o cálculo de integrais, inclusive para a exportação de imagens.

Diferentemente do tipo de dado Shapelet, o qual aborda o uso exclusivo de imagens, esta pesquisa de mestrado propõe o uso de geometrias vetoriais para representar dados espaciais vagos. Além disso, o trabalho correlato descrito nessa seção não investiga ou implementa predicados topológicos vagos. Outra característica refere-se ao fato de que o Shapelet foi desenvolvido para resolver problemas específicos de áreas como a astronomia. De forma contrária, o TAD proposto nesta dissertação de mestrado foi desenvolvida para processar consultas analíticas que utilizarão operadores espaciais e predicados topológicos vagos de forma eficiente.

3.6 Considerações Finais

Neste capítulo foram descritos os principais trabalhos correlatos a presente dissertação de mestrado. Esses trabalhos foram categorizados em: (i) funcionalidades oferecidas pelo SGBD PostgreSQL/PostGIS e TADs genéricos (iBLOB); (ii) trabalhos voltados ao armazenamento de dados espaciais vagos em DWGs; e (iii) trabalhos que implementam dados espaciais vagos. Também foram identificadas as limitações nos trabalhos correlatos e destacados os diferenciais desta dissertação de mestrado.

Resumidamente, esta dissertação de mestrado tem como objetivo e diferencial principais o fato de propor um novo TAD, denominado VagueGeometry o qual é baseado no modelo exato VASA, para representar dados espaciais vagos no SGBD PostgreSQL/PostGIS. A proposta do TAD VagueGeometry engloba uma forma de armazenamento interna para os dados espaciais vagos, os quais são complexos e podem possuir diversas partes disjuntas, além de manipulação de objetos espaciais vagos, tais como relacionamentos topológicos e operações numéricas envolvendo dados espaciais vagos. Operadores para estender a linguagem SQL e assim realizar tratamentos específicos envolvendo tais operações também são empregados.

Na Tabela 3.1 é mostrada uma comparação entre as implementações de dados espaciais vagos discutidos na Seção 3.5 e as funcionalidades oferecidas pelo TAD VagueGeometry destacadas na última coluna em cinza. Foram consideradas as principais características que um sistema de tipos espaciais deve possuir de acordo com Güting (1994). Essas características são a existência de *predicados topológicos* (e.g. *overlap*), *operações geométricas de conjunto* (e.g. união), *operações numéricas* (e.g. área) e *operadores nativos* (e.g. operador $\sim$) envolvendo dados espaciais vagos. Todas as implementações provêm suporte a todas as operações geométricas de conjunto vagos, com exceção da implementação dos tipos de dados espaciais *fuzzy* (KRAIPEERAPUN, 2004; DILO, 2006b) que não implementa a operação de diferença entre linhas vagas. Somente a implementação da VASA provê suporte aos predicados topológicos vagos e operações numéricas vagas, porém com adaptações para lidar com os três valores lógicos e os valores mínimos e máximos que são retornados por estas operações respectivamente (Seção 3.5.1). Nenhum trabalho correlato implementa operadores nativos para manipular as operações envolvendo dados espaciais vagos. O TAD VagueGeometry engloba todas essas operações sem apresentar as desvantagens mencionadas e propõe a implementação de operadores nativos.

Além disso, na Tabela 3.1 também é considerado o suporte de funções de entrada e saída no formato textual e binário, uma vez que tais funções são necessárias para a criação e recuperação de objetos espaciais vagos. É importante notar que nenhum trabalho correlato se preocupa em

Tabela 3.1: Comparação das funcionalidades oferecidas pelos trabalhos correlatos e com o TAD VagueGeometry.

Funcionalidade	Shapelet (ZINN; BOSCH; GERTZ, 2007)	Implementação VASA (PAULY; SCHNEIDER, 2008)	Implementação dos Tipos de Dados Espaciais *Fuzzy* (KRAIPEERAPUN, 2004; DILO, 2006b)	TAD VagueGeometry
Predicados Topológicos	Não	Sim	Não	Sim
Operações Geométricas de Conjunto	Sim	Sim	Sim	Sim
Operações Numéricas	Não	Sim	Não	Sim
Operadores Nativos	Não	Não	Não	Sim
Entrada e Saída no Formato Textual	Não	Não	Sim (formato do GRASS)	Sim
Entrada e Saída no Formato Binário	Sim	Não	Sim (arquivos do GRASS)	Sim
Implementado em um SGBD	Sim	Sim	Não	Sim

como o usuário final pode criar e visualizar textualmente um objeto espacial vago. Apesar da implementação dos tipos de dados *fuzzy* ter uma forma de entrada textual e binária, esta é feita pelo GIS GRASS e não permite o usuário final entender como é representado um dado espacial vago. Além disso, o Shapelet (ZINN; BOSCH; GERTZ, 2007) possuí uma forma de entrada e saída somente por imagens, uma vez que sua implementação é baseada em *raster*. Apesar da implementação da VASA não prover qualquer tipo de representação, um objeto espacial vago é criado fornecendo de forma sequenciada via linha de comando e manualmente todos os pares de coordenadas que formam o núcleo e a conjectura do objeto. Como último parâmetro de comparação, foi considerado se a implementação é concebida em um SGBD, onde aplicações finais podem acessar diretamente os TADs implementados. Somente a implementação dos tipos de dados espaciais *fuzzy* não foi realizada em um SGBD. O TAD VagueGeometry também engloba essas importantes características, ao definir formalmente os formatos textuais e binários dos objetos espaciais vagos. Com isso aplicações finais são capazes de se comunicar, escolher o tipo de representação mais adequado ao seu contexto, e então utilizar os dados espaciais vagos em uma conexão direta com o SGBD.

No próximo capítulo, Capítulo 4, é descrita, detalhada e exemplificada a proposta principal desta dissertação de mestrado. Ainda, neste capítulo, é mostrado como aplicações podem acessar diretamente o SGBD PostgreSQL para seu uso. Posteriormente, no Capítulo 5, uma avaliação experimental é descrita considerando o tempo de execução de consultas espaciais en-

volvendo objetos espaciais vagos, para medir o desempenho no processamento de consultas analíticas com predicados topológicos. Contudo, na comparação é considerado o armazenamento do núcleo e conjectura em colunas distintas e a definição dos predicados topológicos vagos por meio da linguagem PL/pgSQL com as funções fornecidas pelo PostGIS. Ou seja, os predicados foram implementados no nível mais alto do SGBD somente reutilizando as funcionalidade já existentes. Os trabalhos correlatos, que implementam dados espaciais vagos e que foram descritos neste capítulo, não foram considerados nesta avaliação experimental pelos seguintes motivos: o Shapelet é baseado no modelo probabilístico, utiliza somente imagens como entrada e saída além de não prover suporte aos predicados topológicos; a implementação da VASA existente é somente no SGBD Oracle, o qual tem restrições de licença para o seu uso; e por fim, a implementação dos tipos de dados *fuzzy* não foi realizada em um SGBD além de também não fornecer suporte aos predicados topológicos. Outra parte da avaliação experimental é avaliar o desempenho no processamento de consultas SOLAP sobre ambiente de DWG. Nesta avaliação, foi considerado o trabalho de Siqueira (2012) (Seção 3.4), o qual armazena a parte exata e duvidosa de um objeto espacial vago em tabelas distintas. Para processar os predicados topológicos vagos sobre o DWG que segue esta modelagem, foi usado a implementação dos predicados por meio das funções em PL/pgSQL.

Capítulo 4

O TIPO ABSTRATO DE DADOS VAGUEGEOMETRY

Este capítulo detalha a principal proposta desta pesquisa de mestrado, o TAD VagueGeometry, o qual foi implementado como uma extensão do SGBD PostgreSQL. Definições dos tipos de dados do TAD VagueGeometry bem como suas representações e operações são detalhadas e exemplificadas neste capítulo.

4.1 Considerações Iniciais

Neste capítulo é apresentado o principal resultado desta pesquisa de mestrado, o TAD VagueGeometry. O capítulo está organizado da seguinte forma. Na Seção 4.2, a implementação do TAD VagueGeometry é detalhada, a qual especifica seus tipos de dados e as suas estruturas de dados internas. Na Seção 4.3 são descritas as operações do TAD VagueGeometry, inclusive suas representações textuais e binárias que permitem o uso do TAD em aplicações finais. Na Seção 4.4 é proposta melhorias visando um melhor desempenho no processamento de predicados topológicos vagos. Para exemplificar as operações do TAD VagueGeometry, na Seção 4.5 é descrito um exemplo de aplicação que usa o TAD VagueGeometry para manipular dados espaciais vagos. Por fim, na Seção 4.6 as considerações finais sobre o capítulo são feitas.

A documentação completa do TAD VagueGeometry que contém todas as especificações, manual de instalação e detalhamento das operações com exemplos pode ser acessada em `http://gbd.dc.ufscar.br/vaguegeometry/`. Esta documentação além de conter todo o conteúdo descrito neste capítulo, abrange mais exemplos e operações, que por limitação de espaço, não são descritas aqui.

4.2 Especificação dos Tipos de Dados de VagueGeometry

A proposta do TAD VagueGeometry, baseado no modelo exato VASA, foi implementado na linguagem C e engloba os tipos de dados mostrados na Figura 4.1 em fundo branca. O maior nível da hierarquia é o tipo VagueGeometry. Um objeto do tipo VagueGeometry pode assumir diferentes tipos de dados, os quais podem ser instanciados como: *ponto vago* (VAGUEPOINT), *multiponto vago* (VAGUEMULTIPOINT), *linha vaga* (VAGUELINESTRING), *multilinha vaga* (VAGUEMULTILINESTRING), *polígono vago* (VAGUEPOLYGON) ou *multipolígono vago* (VAGUEMULTIPOLYGON). Cada objeto espacial vago é composto por um par de objetos espaciais *crisp* disjuntos ou adjacentes do mesmo tipo de dado, os quais são destacados em cinza na Figura 4.1. Por exemplo, um ponto vago (VAGUEPOINT) é composto por um par de pontos *crisp*, onde um representa o núcleo e outro a conjectura. É importante enfatizar que o par de objetos espaciais *crisp* também têm o mesmo identificador de referência de sistema espacial (SRID).

O modelo exato VASA tem como principal objetivo a reutilização de implementações já existentes de tipos de dados espaciais *crisp* para representação de objetos espaciais vagos. Dessa forma, o TAD VagueGeometry utiliza o módulo GEOS (GEOS..., 2014) para reutilizar algoritmos de manipulação de objetos espaciais *crisp*. O módulo GEOS é uma biblioteca em C/C++ amplamente utilizada por SGBDEs (tal como o PostgreSQL/PostGIS) e SIGs (tal como GRASS), que segue os padrões estabelecidos pela OGC (OPEN..., 2014).

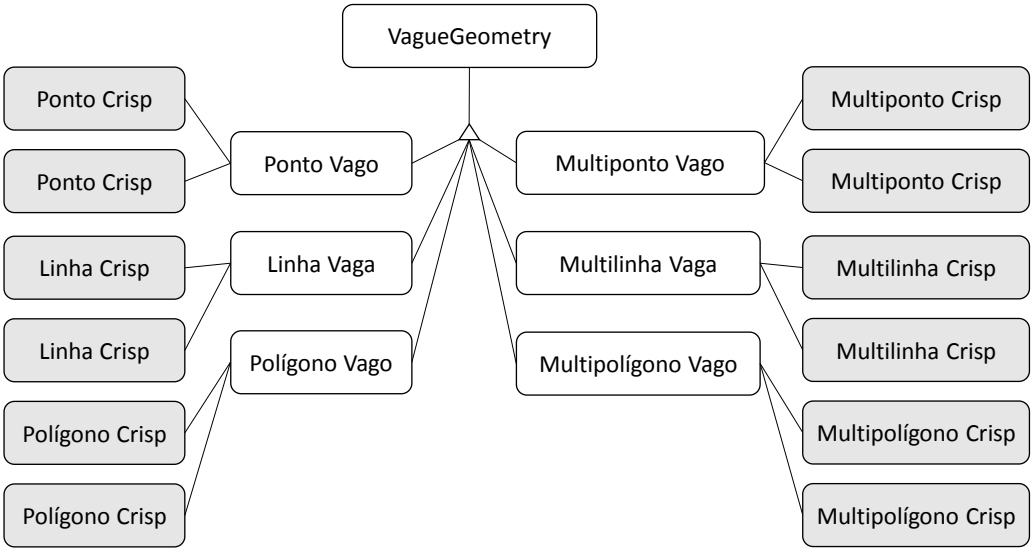


Figura 4.1: Os tipos de dados do TAD VagueGeometry

Nesse sentido, além de utilizar o módulo GEOS, este TAD também reutiliza os objetos espaciais *crisp* do PostGIS. Assim, a estrutura interna do TAD VagueGeometry é um par de objetos do PostGIS de mesmo tipo de dado. O tipo de dado do PostGIS é o GEOMETRY, o qual é representado internamente no código fonte pela estrutura LWGEOM. A estrutura do tipo de dado VagueGeometry, denominada VAGUEGEOM, é apresentada no trecho de código do Algoritmo 1, e detalhada como segue:

- O elemento *type* armazena o tipo do dado vago que VAGUEGEOM representa, o qual pode ser: ponto vago, linha vaga, polígono vago, multiponto vago, multilinha vaga e multipolígono vago (Figura 4.1);
- O elemento *flags* armazena em um *byte*, se o objeto espacial vago tem somente o núcleo, ou somente a conjectura, ou ambos, ou é vazia;
- Os elementos *kernel* e *conjecture* formam o par de objetos espaciais *crisp* do mesmo tipo de dado, por meio do tipo de dado interno usado pelo PostGIS (LWGEOM).

Algoritmo 1 Estrutura de dados para armazenar um objeto VagueGeometry em memória principal

```
1: typedef struct {
2:   uint8_t type;
3:   uint8_t flags;
4:   LWGEOM *kernel;
5:   LWGEOM *conjecture;
6: } VAGUEGEOM;
```

4.2.1 A Estrutura para o Armazenamento no SGBD PostgreSQL

A estrutura VAGUEGEOM (Algoritmo 1) pode possuir tamanho variável, uma vez que a quantidade de pontos contido nas geometrias não é fixo e dependente da aplicação. Com isso, existe a necessidade de um processo de serialização, o qual transforma um objeto espacial vago em um fluxo contínuo de dados (*data stream*) para ser inserido no PostgreSQL. Dessa forma a estrutura VAGUEGEOMSERIALIZED, mostrada no Algoritmo 2, é definida para armazenar de forma serializada os objetos espaciais vagos, além de ser manipulada diretamente pelo núcleo do SGBD PostgreSQL. Para isso, é necessário seguir as especificações do SGBD PostgreSQL descritas na Seção 2.2 do Capítulo 2. A estrutura VAGUEGEOMSERIALIZED é detalhada como segue:

- O elemento *size* armazena em um inteiro de 4 *bytes*, o tamanho do objeto instanciado em *bytes*, o qual é requerido pelo PostgreSQL e manipulado somente por ele;
- O elemento *srid* armazena o SRID dos objetos *crisp* em um vetor de caracteres (i.e., 3 *bytes*) em forma serializada utilizando aritmética de ponteiros;
- O elemento *flags* se refere ao mesmo elemento *flags* da estrutura VAGUEGEOM (Algoritmo 1);
- O elemento *data* é um vetor que armazena de forma serializada os objetos do núcleo (*kernel*) e conjectura (*conjecture*), os quais são objetos do PostGIS;

Algoritmo 2 Estrutura de dados para armazenar um objeto VagueGeometry no SGBD PostgreSQL

```
1: typedef struct {
2:   uint32_t size;
3:   uint8_t srid[3];
4:   uint8_t flags;
5:   uint8_t data[1];
6: } VAGUEGEOMSERIALIZED;
```

A ordem da serialização de um objeto VAGUEGEOMSERIALIZED (Algoritmo 2) é um fator importante, uma vez que após recuperar um objeto armazenado no PostgreSQL, o processo inverso deve ser feito. Ou seja, deve ser possível a transformação do fluxo contínuo de *bytes* em objetos mantidos em memória principal (estrutura VAGUEGEOM) para realizar as operações de alto nível. Outro fator importante é o alinhamento dos dados e se necessário, a inserção de preenchimentos (*padding*) para satisfazer a condição do alinhamento. O alinhamento adotado pelo PostGIS, bem como pelo TAD VagueGeometry, é de 8 *bytes*. Isso significa que o tamanho total do objeto deve ser múltiplo por 8 e que com isso, é possível recuperar, a cada 8 *bytes*, partes de um objeto acessando diretamente sua posição no vetor serializado. Ou seja, as coordenadas *x* e *y* de um ponto podem ser acessadas sem a necessidade de um processo custoso de transformação. Os três primeiros elementos da estrutura VAGUEGEOMSERIALIZED, os quais são o *size*, *srid* e *flags*, tem um tamanho total de 8 *bytes*, assim, satisfazendo o alinhamento imposto. Por outro lado, o alinhamento deve também ser mantido pelo elemento *data*, que pode ter tamanho variável.

Uma vez que um objeto VagueGeometry é composto por dois objetos do PostGIS (Algoritmo 1), a serialização utilizada pelo TAD VagueGeometry reutiliza a serialização de objetos espaciais *crisp* provido pelo PostGIS. Esta reutilização é para serializar o núcleo e a conjectura

Tabela 4.1: Ordem de serialização de um objeto espacial *crisp* utilizada pelo PostGIS.

Tipo de Dado	Ordem de Serialização
Ponto *Crisp*	<tipo de dado POINT> <número de pontos (0 para vazio e 1 caso contrário)> [coordenada x] [coordenada y]
Linha *Crisp*	<tipo de dado LINESTRING> <número de pontos (0 para vazio e 1 caso contrário)> para cada ponto i da linha: [coordenada x_i] [coordenada y_i]
Polígono *Crisp*	<tipo de dado POLYGON> <número de anéis (0 para vazio e 1 caso contrário)> para cada anel j do polígono: <número de pontos do anel j> se o número de anéis é impar: <*padding*> para cada anel j do polígono e cada ponto i de j: [coordenada x_{j_i}] [coordenada y_{j_i}]
Objeto Espacial *Crisp* Complexo	<tipo de dado do objeto espacial complexo> <número de componentes (0 para vazio e 1 caso contrário)> para cada componente i do objeto espacial complexo: [serialização de i]

de um objeto espacial vago. Na Tabela 4.1 é mostrado a ordem de serialização de objetos espaciais *crisp* realizada pelo PostGIS. A notação <> é utilizada para representar que o elemento possuí 4 *bytes*, enquanto a notação [ ] representa que o elemento possuí um múltiplo de 8 *bytes*. Cada tipo de dado tem sua serialização específica o qual pode variar em número de coordenadas. É importante notar que para o tipo polígono é necessário um *padding* de 4 *bytes* uma vez que o número de anéis (i.e. a borda externa somado ao número de buracos) pode quebrar o alinhamento. Além disso, o processo de serialização para um objeto espacial *crisp* complexo é o processo de serialização de cada componente do objeto, uma vez que um objeto espacial *crisp* complexo é composto por vários objetos espaciais *crisp* simples de mesmo tipo. Por exemplo, o processo de serialização de um multipolígono, serializa cada sub polígono, uma vez que um multipolígono é composto por vários polígonos simples.

A ordem de serialização do elemento *data* da estrutura VAGUEGEOMSERIALIZED é detalhada na Tabela 4.2. Existem quatro casos de serialização de acordo com a existência ou não de núcleo e conjectura: (i) quando o objeto espacial vago é vazio (ou seja, não tem núcleo

Tabela 4.2: Ordem de serialização de um objeto espacial vago do TAD VagueGeometry.

Caso	Ordem de Serialização
Objeto espacial vago vazio	<tipo de dado do objeto espacial vago> <*padding*>
Existência somente do núcleo	⟨*flag* do PostGIS referente ao núcleo⟩ ⟨*padding*⟩ ⟨*padding*⟩ ⟨*padding*⟩ <*padding*> [serialização do núcleo]
Existência somente da conjectura	⟨*flag* do PostGIS referente a conjectura⟩ ⟨*padding*⟩ ⟨*padding*⟩ ⟨*padding*⟩ <*padding*> [serialização da conjectura]
Existência do núcleo e da conjectura	<tamanho em bytes do núcleo> <*padding*> ⟨*flag* do PostGIS referente ao núcleo⟩ ⟨*padding*⟩ ⟨*padding*⟩ ⟨*padding*⟩ <*padding*> [serialização do núcleo] ⟨*flag* do PostGIS referente a conjectura⟩ ⟨*padding*⟩ ⟨*padding*⟩ ⟨*padding*⟩ <*padding*> [serialização da conjectura]

e conjectura definidos); (ii) o objeto espacial vago tem somente o núcleo; (iii) o objeto espacial vago tem somente a conjectura; e, (iv) o objeto espacial vago tem um núcleo e uma conjectura. Para o caso (iv) também é armazenado o tamanho em *bytes* do núcleo para que seja possível o acesso da conjectura de forma mais rápida.

Em geral, é reutilizada a ordem de serialização apresentada na Tabela 4.1 para a serialização do núcleo e da conjectura. Dessa forma, a ordem de serialização pode variar de acordo com o tipo de dado do objeto espacial *crisp*. Por exemplo, a serialização de um ponto vago que contém núcleo e conjectura irá reutilizar a serialização do ponto *crisp* da Tabela 4.1. Adicionalmente, a notação ⟨⟩ representa um elemento de 1 *byte*. A *flag* do PostGIS, que é também armazenada junto ao núcleo ou conjectura, representa informações internas do objeto espacial *crisp*, tal como a sua dimensionalidade (bidimensional ou tridimensional) e a presença ou não de MBR.

Por fim, para manter o alinhamento de 8 *bytes*, é necessário adicionar *padding* em todos os casos.

4.2.2 As Estruturas VagueBool e VagueNumeric

No modelo exato VASA, como discutido na Seção 2.4.1 do Capítulo 2, os predicados topológicos vagos (por exemplo, *overlap*) retornam um valor lógico que pode assumir 3 valores: *true*, *false* ou *maybe*. Portanto, visando o retorno dos predicados topológicos vagos foi implementada a estrutura VAGUEBOOL que assume um dos três valores lógicos da álgebra VASA. A estrutura VAGUEBOOL é mostrada no Algoritmo 3, possuindo apenas 1 elemento que pode assumir três valores distintos: 0 para *false*, 1 para *true* e 2 para *maybe*. Por ter tamanho fixo de 1 *byte*, tal estrutura não necessita de um processo de serialização.

Algoritmo 3 Estrutura de dados para armazenar os três valores lógicos da VASA.

```
1: typedef struct {
2:   uint8_t vbool;
3: } VAGUEBOOL;
```

Já os operadores numéricos (por exemplo, a distância entre duas linhas vagas) retornam um valor máximo e um valor mínimo. Devido a isso, uma nova estrutura, denominada VAGUENUMERIC, foi implementada. Esta estrutura, mostrada no Algoritmo 4, possui dois elementos responsáveis pelo armazenamento dos valores numéricos mínimo e máximo retornados pelos operadores numéricos envolvendo objetos espaciais vagos. Assim como a estrutura VAGUEBOOL, a estrutura VAGUENUMERIC não necessita de outra estrutura para serialização, uma vez que é conhecido o tamanho total do objeto. O tamanho total de um objeto VAGUENUMERIC é de 16 *bytes*.

Algoritmo 4 Estrutura de dados para armazenar os valores mínimo e máximo de uma operação numérica da VASA.

```
1: typedef struct {
2:   double min;
3:   double max;
4: } VAGUENUMERIC;
```

4.2.3 A Definição do TAD VagueGeometry no SGBD PostgreSQL

No Algoritmo 5 é descrito o comando SQL CREATE TYPE para criar o tipo de dado VagueGeometry no SGBD PostgreSQL. Este comando estabelece como o novo tipo de dado irá

ser tratado pelo SGBD PostgreSQL. Além disso, seus parâmetros estão estritamente relacionados a implementação na linguagem de baixo nível utilizada. O parâmetro que indica o tamanho interno do dado (*internallength*) é variável. As funções de *input* e *output* são configuradas conforme as especificações do SGBD PostgreSQL para transformar representações textuais em internas e vice versa. Além disso, as funções *send* e *receive* foram definidas para transformar representações binárias em internas e vice-versa. Modificadores de tipos de dados também são definidos pelas funções *typmod_in* e *typmod_out* para tratar as restrições de tipos. O delimitador ':' também foi definido para vetores na linguagem SQL contendo objetos do tipo VagueGeometry. A categoria do tipo VagueGeometry é geométrico, indicado pelo parâmetro *category = 'G'*. O método de armazenamento escolhido foi o *main*, devido à sua vantagem de compressão e tentativa exaustiva de armazenamento na tabela principal. Por fim, o alinhamento dos dados serializados tem como tamanho padrão 8 *bytes* (double) devido às coordenadas usadas em dados espaciais.

Algoritmo 5 A especificação do tipo VagueGeometry no SGBD PostgreSQL

```
1: CREATE TYPE VagueGeometry (
2:   internallength = variable,
3:   input = vg_in,
4:   output = vg_out
5:   send = vg_send,
6:   receive = vg_recv,
7:   typmod_in = vg_typmod_in,
8:   typmod_out = vg_typmod_out,
9:   delimiter = ':',
10:  category = 'G',
11:  alignment = double,
12:  storage = main
13: );
```

Para as estruturas VAGUEBOOL e VAGUENUMERIC também foram criados seus tipos de dados por meio do comando SQL CREATE TYPE. Estes comandos são mostrados nos Algoritmos 6 e 7, respectivamente. Ambas as estruturas têm tamanho fixo, dessa forma o *internallength* de VAGUEBOOL é de 1 *byte* e o *internallength* de VAGUENUMERIC é de 16 *bytes*. Para ambos também foram definidas funções de *input* e *output*, bem como funções de *send* e *receive*. A categoria do tipo VAGUEBOOL é do tipo lógico, representado pela categoria 'B' e do VAGUENUMERIC é do tipo numérico, representado pela categoria 'N'. Além disso, o alinhamento dos dados de VAGUEBOOL é *char* (ou seja, 1 *byte*) e para o VAGUENUMERIC é *double* (ou seja, 8 *bytes*). Por fim, ambos os tipos de armazenamento é *plain*, uma vez que não é necessária a compactação de seus valores por serem pequenos e de tamanhos fixos.

Algoritmo 6 A especificação do tipo VagueBool no SGBD PostgreSQL

```
1: CREATE TYPE VagueBool (
2:   internallength = 1,
3:   input = vb_in,
4:   output = vb_out
5:   send = vb_send,
6:   receive = vb_recv,
7:   delimiter = ',',
8:   category = 'B',
9:   alignment = char,
10:  storage = plain
11: );
```

Algoritmo 7 A especificação do tipo VagueNumeric no SGBD PostgreSQL

```
1: CREATE TYPE VagueNumeric (
2:   internallength = 16,
3:   input = vn_in,
4:   output = vn_out
5:   send = vn_send,
6:   receive = vn_recv,
7:   delimiter = ',',
8:   category = 'N',
9:   alignment = double,
10:  storage = plain
11: );
```

As funções para manipular cada tipo de dado (VagueGeometry, VagueBool e VagueNumeric) bem como suas funções (e.g., *input*, *output*, *send* e *receive*), presentes nos Algoritmos 5, 6 e 7, foram implementadas na linguagem C e assim associadas para a linguagem SQL. Tal associação é criada por meio do comando SQL CREATE FUNCTION, o qual toma como parâmetro a função na linguagem C da extensão, de acordo com a documentação do SGBD PostgreSQL (POSTGRESQL..., 2014). Assim, quando uma função SQL é chamada por uma aplicação, o SGBD PostgreSQL automaticamente chama a função correspondente na linguagem C e realiza o tratamento adequado dos objetos na memória principal e em disco. Esse tratamento é o mesmo feito para os tipos de dados primários (e.g, números inteiros) do SGBD PostgreSQL, dessa forma, o desempenho é o mesmo para os objetos dos tipos VagueGeometry, VagueBool e VagueNumeric.

Um exemplo de criação de uma função usando a linguagem SQL, a qual faz a ligação para uma função na linguagem C, é mostrada no Algoritmo 8. Este algoritmo realiza a ligação da função em C, *VG_in*, para a função *vg_in* na linguagem SQL usada como parâmetro para a função de *input* no Algoritmo 5. A função *vg_in* é definida pelo comando CREATE FUNC-

TION, o qual toma como parâmetro uma *string* e retorna um objeto VagueGeometry, ou seja, realiza a transformação textual para a representação interna do objeto. A linha 3 especifica qual função *vg_in* liga, especificando na linha 4 sua linguagem. Para o restante das funções que manipulam os novos tipos de dados definidos, a mesma estratégia é definida. Nesta dissertação todas as funções não serão mostradas devido a limitação de espaço. Porém, na documentação online (Seção 4.1) é possível o acesso do código destas funções assim como o código fonte do TAD VagueGeometry.

Algoritmo 8 A função *input* do tipo VagueGeometry.

```
1: CREATE OR REPLACE FUNCTION vg_in(cstring)
2:   RETURNS vaguegeometry
3:   AS 'MODULE_PATHNAME', 'VG_in'
4: LANGUAGE 'c' IMMUTABLE STRICT;
```

4.3 Operações do TAD VagueGeometry

Foram implementadas várias funções, para manipular dados espaciais vagos, categorizadas como:

- Funções de entrada e saída: recebe dados espaciais vagos em sua forma textual ou binária e os armazenam internamente, bem como o inverso. Estas funções são detalhadas na Seção 4.3.1;
- Métodos assessores: edita, remove ou acessa partes dos dados espaciais vagos. Estas funções são detalhadas na Seção 4.3.2;
- Operações geométricas de conjuntos: operações de união, intersecção e diferença entre dados espaciais vagos. Estas funções são detalhadas na Seção 4.3.3;
- Operações específicas de tipos: manipula dados espaciais vagos de tipos pré-determinados (por exemplo, a borda de regiões vagas). Estas funções são detalhadas na Seção 4.3.4;
- Predicados topológicos: verifica os relacionamentos topológicos existentes entre objetos espaciais vagos e retorna um objeto do tipo VagueBool, por exemplo, o predicado espacial "inside" pode retornar *maybe*, *true* ou *false*. Estas funções são detalhadas na Seção 4.3.5;
- Operações numéricas: calculam medidas numéricas de objetos espaciais vagos (por exemplo, a área de uma região vaga), bem como entre objetos espaciais vagos (por exemplo, a distância entre duas regiões vagas), e retornam um objeto do tipo VagueNumeric, o

qual contém um valor máximo e um valor mínimo. Estas funções são detalhadas na Seção 4.3.6.

- Operadores: realiza manipulações dos resultados derivados dos predicados topológicos vagos e operações numéricas. Estes operadores são detalhados na Seção 4.3.7.

É importante notar que todas as operações usam o prefixo VG_ em suas assinaturas para denotar que é uma função do TAD VagueGeometry. Ao longo das subseções, exemplos também são citados.

4.3.1 Funções de Entrada e Saída

Apesar de ser possível definir atributos do tipo VagueGeometry, somente com as definições já mostradas até aqui, não é possível inserir ou recuperar objetos espaciais vagos do tipo VagueGeometry. Nesse sentido, para manipular esse tipo de dado é necessário definir funções de entrada e saída. Existem dois tipos de funções de entrada e saída. A primeira usa a representação textual, enquanto a segunda usa a representação binária. Em geral, a função de entrada transforma a representação textual ou binária para a representação interna, e a função de saída realizada a transformação inversa.

Visando auxiliar na definição das representações textuais e binárias, é necessário definir as seguintes funções auxiliares: *name*, *id*, *WKT*, *GML*, *KML*, *GeoJSON*, *WKB* e *binary*. As funções *name* e *id* recebem como parâmetro um objeto do tipo VagueGeometry A. Assim, $name(A)$ extraí a palavra chave `VAGUEPOINT` quando A é um ponto vago, `VAGUELINESTRING` quando A é uma linha vaga, `VAGUEPOLYGON` quando A é uma região vaga, `VAGUEMULTIPOINT` quando A é um multiponto vago, `VAGUEMULTILINESTRING` quando A é uma multilinha vaga e `VAGUEMULTIPOLYGON` quando A é uma região complexa vaga. Já $id(A)$ extraí o identificador numérico 1 quando A é um ponto vago, 2 quando A é uma linha vaga, 3 quando A é uma região vaga, 4 quando A é um multiponto vago, 5 quando A é uma multilinha vaga e 6 quando A é uma região complexa vaga. As funções *WKT*, *GML*, *KML*, *GeoJSON* e *WKB* recebem como parâmetro um objeto espacial *crisp* o e extraí a sua respectiva representação textual. Por exemplo, $WKT(o)$ extraí a representação WKT de o, enquanto $WKB(o)$ extraí a representação WKB de o. Finalmente, *binary* extraí o formato binário a partir de um número inteiro.

Os possíveis formatos textuais de um objeto VagueGeometry são: (i) *Vague Well-Known Text* - VWKT (baseado no formato WKT), (ii) *Vague Geographic Markup Language* - VGML (baseado no formato GML), (iii) *Vague Keyhole Markup Language* - VKML (baseado no formato KML) e (iv) *Vague Geographic JavaScript Object Notation* - vGeoJSON (baseado no

formato GeoJSON). O formato binário de um objeto VagueGeometry é o *Vague Well-Known Binary* (baseado no formato WKB).

Tais formatos são definidos como segue. Seja um objeto espacial vago A, do tipo VagueGeometry, o qual pode assumir seis diferentes tipos de dados (Figura 4.1), formado por um núcleo A_n e conjectura A_c (i.e. objetos espaciais *crisp*), os formatos VWKT, VGML, VKML, vGeoJSON e VWKB são definidos, respectivamente, como

(i) $VWKT(A) = name(A)(WKT(A_n); WKT(A_c))$

(ii) $VGML(A)$ = <vgml:$name(A)$>
<vgml:Kernel>
$GML(A_n)$
</vgml:Kernel>
<vgml:Conjecture>
$GML(A_c)$
</vgml:Conjecture>
</vgml:$name(A)$>

(iii) $VKML(A)$ = <vkml:$name(A)$>
<vkml:Kernel>
$KML(A_n)$
</vkml:Kernel>
<vkml:Conjecture>
$KML(A_c)$
</vkml:Conjecture>
</vkml:$name(A)$>

(iv) $vGeoJSON(A)$ = {"type": "$name(A)$",
"kernel": $GeoJSON(A_n)$,
"conjecture": $GeoJSON(A_c)$
}

(v) $VWKB(A) = endianess + binary(id(A)) + WKB(A_n) + WKB(A_c)$

É importante notar que na representação VWKB, o símbolo de soma é usado para denotar a união entre os dados serializados e não para realizar a soma aritmética. Além disso, *endianess* indica como os *bytes* estão organizados em memória principal. Existem duas formas, as

quais podem ser *big-endian* ou *little-endian*, seguindo as especificações da representação WKB. Ademais, é possível adicionar o SRID na representação VWKB, formando assim o *Extended*-VWKB (EVWKB), de forma semelhante como é feito na representação EWKB do PostGIS.

Com relação a representação VWKT, existem dois tipos de variações. A primeira delas é a possibilidade de adicionar o SRID em sua representação, tal como é feito na representação EWKT do PostGIS. Assim, como resultado, o *Extended*-VWKT (EVWKT) é criado. A outra variação é a possibilidade de uma representação mais reduzida de um objeto espacial vago, removendo os nomes dos tipos de dados *crisp* das representações WKT. Isto é possível uma vez que um objeto espacial vago é construído a partir de dois objetos espaciais *crisp* do mesmo tipo, e por isso os tipos de dados do núcleo e conjectura são previamente conhecidos. Por exemplo, um objeto espacial vago do tipo VAGUEPOINT, é representado de forma completa como VAGUEPOINT(POINT(10 10); POINT(5 5)), já a sua forma simplificada é representada como VAGUEPOINT(10 10; 5 5).

Tais representações foram definidas para que diversas aplicações de diferentes contextos possam utilizar o TAD VagueGeometry. Por exemplo, aplicações baseadas em XML ou serviços *web* que utilizem XML como comunicação, podem utilizar as representações VGML ou VKML. Ainda, aplicações baseadas na *web* que utilizem visualizações de mapas em navegadores, podem utilizar a representação vGeoJSON, a qual facilita o acesso aos dados por meio da linguagem JavaScript. Aplicações que manipulam arquivos binários ou que efetuam trocas de informações em baixo nível, podem fazer uso da representação VWKB, que pode apresentar ganhos mais efetivos do que as representações textuais. Por fim, outras aplicações podem fazer uso do VWKT para visualizar de forma mais compacta objetos espaciais vagos.

Com as representações textuais e binárias definidas, é possível inserir e visualizar dados espaciais vagos por meio de funções específicas. As assinaturas das funções que transformam as representações VWKT, EVWKT, VGML, VKML, vGeoJSON, VWKB e EVWKB para o formato interno são listadas respectivamente a seguir

(i) *VG_VagueGeomFromText*(text *VWKT*, integer *SRID*) → VagueGeometry

(ii) *VG_VagueGeomFromEVWKT*(text *EVWKT*) → VagueGeometry

(iii) *VG_VagueGeomFromVGML*(text *VGML*) → VagueGeometry

(iv) *VG_VagueGeomFromVKML*(text *VKML*) → VagueGeometry

(v) *VG_VagueGeomFromvGeoJSON*(text *vGeoJSON*) → VagueGeometry

(vi) *VG_VagueGeomFromVWKB*(bytea *VWKB*, integer *SRID*) → VagueGeometry

(vii) *VG_VagueGeomFromEVWKB*(bytea *EVWKB*) → VagueGeometry

Além dessas representações, existe a possibilidade da criação de objetos espaciais vagos utilizando objetos do PostGIS. A função *VG_MakeVagueGeom* cria um objeto espacial vago a partir de dois objetos Geometry (i.e. objetos do PostGIS), desde que eles sejam disjuntos ou adjacentes. Caso exista sobreposição nos objetos Geometry, a função *VG_EnforceMakeVague-Geom* pode ser utilizada, a qual realiza a operação de diferença entre a conjectura (segundo parâmetro) e o núcleo (primeiro parâmetro), garantindo assim a consistência da representação do objeto espacial vago. Estas funções são interessantes quando aplicações têm objetos espaciais vagos representados em colunas distintas, ou seja, núcleo e conjectura armazenados separadamente. Tais funções possuem as seguintes assinaturas

(i) *VG_MakeVagueGeom*(Geometry *n*, Geometry *c*) → VagueGeometry

(ii) *VG_EnforceMakeVagueGeom*(Geometry *n*, Geometry *c*) → VagueGeometry

Outra possibilidade de criação de objetos espaciais vagos é a utilização do programa *shp2vaguegeom*, para carregar arquivos *shapefiles*, que contenham representações separadas de núcleo e conjectura. Um arquivo *shapefile* é uma representação binária de objetos espaciais *crisp* para interoperabilidade entre SIGs e SGBDEs (ESR, 1998). O programa *shp2vaguegeom* pode ser incluído ao TAD VagueGeometry no momento de sua instalação e obtido na documentação do TAD VagueGeometry (Seção 4.1).

As funções de saída, retornam as representações definidas nessa seção (VWKT, EVWKT, VGML, VKML, vGeoJSON, VWKB e EVWKB) e suas respectivas assinaturas são

(i) *VG_AsText*(VagueGeometry *vg*) → text

(ii) *VG_AsEVWKT*(VagueGeometry *vg*) → text

(iii) *VG_AsVGML*(VagueGeometry *vg*) → text

(iv) *VG_AsVKML*(VagueGeometry *vg*) → text

(v) *VG_AsVGeoJSON*(VagueGeometry *vg*) → text

(vi) *VG_AsVWKB*(VagueGeometry *vg*) → bytea

(vii) *VG_AsEVWKB*(VagueGeometry *vg*) → bytea

4.3.2 Métodos Assessores

Os métodos assessores têm como objetivo realizar operações de edição, remoção ou captura de partes dos objetos espaciais vagos e de objetos VagueNumeric. Sejam *vg* e *vg*2 objetos do tipo VagueGeometry, *g* um objeto do tipo Geometry (i.e. tipo de dado do PostGIS) e *vn* um objeto VagueNumeric resultante de uma operação numérica, as operações assessoras têm as seguintes assinaturas

(i) *VG_GetSRID*(VagueGeometry *vg*) → integer

(ii) *VG_SetSRID*(VagueGeometry *vg*, integer *new_srid*) → VagueGeometry

(iii) *VG_GetType*(VagueGeometry *vg*) → text

(iv) *VG_Conjecture*(VagueGeometry *vg*) → Geometry

(v) *VG_Kernel*(VagueGeometry *vg*) → Geometry

(vi) *VG_Invert*(VagueGeometry *vg*) → VagueGeometry

(vii) *VG_Same*(VagueGeometry *vg*, VagueGeometry *vg2*) → boolean

(viii) *VG_GetMax*(VagueNumeric *vn*) → double precision

(ix) *VG_GetMin*(VagueNumeric *vn*) → double precision

As funções (i) e (ii), captura e redefine o SRID de um objeto espacial vago, respectivamente. É importante enfatizar que ao redefinir um novo SRID por meio da função (ii), as projeções espaciais não são alteradas, somente o identificador é alterado. A função (iii) retorna o tipo de dado de um objeto espacial vago, o qual pode ser VAGUEPOINT, VAGUELINESTRING, VAGUEPOLYGON, VAGUEMULTIPOINT, VAGUEMULTILINESTRING ou VAGUEMULTIPOLYGON. Já as funções (iv) e (v) capturam o núcleo e a conjectura de um objeto espacial vago, respectivamente. Além dessas operações, também é possível fazer a inversão do núcleo pela conjectura e vice versa (função (vi)), e checar se dois objetos espaciais vagos são exatamente os mesmos (função (vii)). Por fim, as funções (viii) e (ix) capturam o valor mínimo e máximo de um objeto do tipo VagueNumeric, respectivamente.

4.3.3 Operações Geométricas de Conjuntos

As operações geométricas de conjuntos envolvendo os dados espaciais vagos são a união, intersecção e diferença. Sejam *A* e *B* dois objetos espaciais vagos (i.e. VagueGeometry), as

operações geométricas de conjunto são intuitivamente definidas como segue. A união entre *A* e *B* retorna um outro objeto VagueGeometry do mesmo tipo, onde o núcleo do resultado é formado pela união dos núcleos de *A* e *B* e a conjectura do resultado é formada pela diferença entre a união das conjecturas de *A* e *B* com a união dos núcleos de *A* e *B*. A intersecção entre *A* e *B* retorna um outro objeto VagueGeometry do mesmo tipo, onde o núcleo do resultado é formado pela intersecção dos núcleos de *A* e *B* e a conjectura do resultado é formada pela união das seguintes intersecções: conjecturas de *A* e *B*; núcleo de *A* com a conjectura de *B*; e, conjectura de *A* com o núcleo de *B*. A diferença entre *A* e *B* retorna um outro objeto VagueGeometry do mesmo tipo, onde o núcleo do resultado é formado pela diferença entre o núcleo de *A* com a união do núcleo e conjectura de *B* e a conjectura do resultado é formada pela união das seguintes operações: intersecção entre as conjecturas de *A* e *B*; intersecção do núcleo de *A* com a conjectura de *B*; e, a diferença entre a conjectura de *A* com a união do núcleo e conjectura de *B*. A definição formal das operações geométricas de conjunto podem ser encontradas em Pauly e Schneider (2010).

Na Figura 4.2 são mostrados dois objetos VagueGeometry do tipo VAGUEMULTIPOLYGON para exemplificar as operações geométricas de conjuntos. Suas representações textuais VWKT no formato simplificado também são mostrados nesta figura. Na Figura 4.3 são mostrados os resultados das operações de união, intersecção e diferença sobre os objetos VagueGeometry mostrados na Figura 4.2. Ainda nesta figura são mostradas suas respectivas representações textuais VWKT no formato simplificado.

As operações geométricas de conjuntos no TAD VagueGeometry são definidas como segue. Seja *vg* um objeto VagueGeometry de qualquer tipo e *vg*2 um objeto VagueGeometry de dimensão menor que *vg* (e.g, um ponto vago, por ser constituído por pontos *crisp* tem dimensão 0 e é de menor dimensão que as linhas vagas e regiões vagas), as operações geométricas de conjuntos têm as seguintes assinaturas

(i) *VG_Union*(VagueGeometry *vg*, VagueGeometry *vg*) $\rightarrow$ VagueGeometry

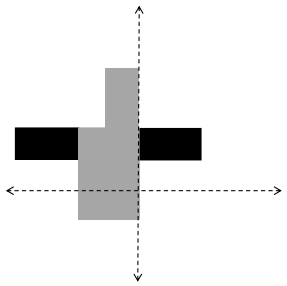
(ii) *VG_Union*([VagueGeometry vg_1, ..., VagueGeometry vg_n]) $\rightarrow$ VagueGeometry

(iii) *VG_Union*(VagueGeometry *vg*) $\rightarrow$ VagueGeometry

(iv) *VG_Intersection*(VagueGeometry *vg*, VagueGeometry *vg*) $\rightarrow$ VagueGeometry

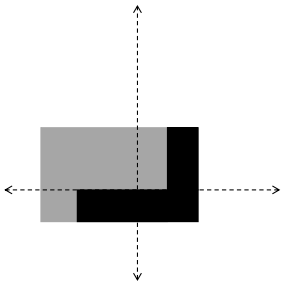
(v) *VG_Intersection*(VagueGeometry *vg*, VagueGeometry *vg2*) $\rightarrow$ VagueGeometry

(vi) *VG_Difference*(VagueGeometry *vg*, VagueGeometry *vg*) $\rightarrow$ VagueGeometry



VAGUEMULTIPOLYGON(((-2 1, -2 2, -4 2, -4 1, -2 1)), ((0 1, 0 2, 2 2, 2 1, 0 1)); ((0 -1, -2 -1, -2 2, -1 2, -1 4, 0 4, 0 -1)))

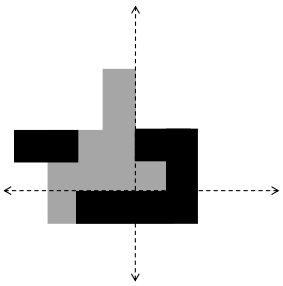
(a)



VAGUEMULTIPOLYGON(((-2 -1, -2 0, 1 0, 1 2, 2 2, 2 -1, -2 -1)); ((-3 -1, -3 2, 1 2, 1 0, -2 0, -2 -1, -3 -1)))

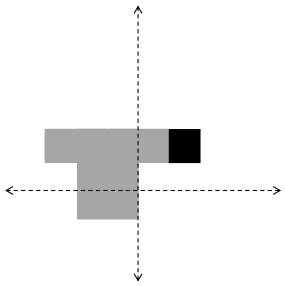
(b)

Figura 4.2: Dois objetos VagueGeometry (a) e (b) do tipo VAGUEMULTIPOLYGON com suas respectivas representações textuais VWKT no formato simplificado.



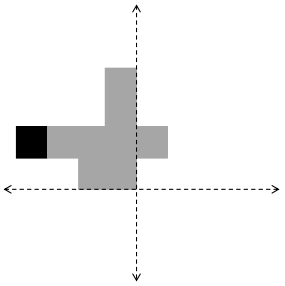
VAGUEMULTIPOLYGON(((-2 1,-4 1,-4 2,-2 2,-2 1)),((0 1,0 2,1 2,2 2,2 1,2 -1,-2 -1,-2 0,1 0,1 1,0 1)); ((-2 -1,-3 -1,-3 1,-2 1,-2 2,-1 2,-1 4,0 4,0 2,0 1,1 1,1 0,0 0,-2 0,-2 -1)))

União



VAGUEPOLYGON((1 2,2 2,2 1,1 1,1 2)); ((-2 -1,-2 0,-2 1,-3 1,-3 2,-2 2,-1 2,0 2,1 2,1 1,0 1,0 0,0 -1,-2 -1))

Intersecção



VAGUEMULTIPOLYGON(((-3 1,-4 1,-4 2,-3 2,-3 1)); ((-2 0,-2 1,-3 1,-3 2,-2 2,-1 2,-1 4,0 4,0 2,1 2,1 1,0 1,0 0,-2 0)))

Diferença

Figura 4.3: Resultado das operações geométricas de conjunto sobre os objetos VagueGeometry (a) e (b) da Figura 4.2 com suas respectivas representações textuais VWKT no formato simplificado.

Em geral, ao menos uma assinatura para cada operação geométrica de conjunto recebe como parâmetros dois objetos VagueGeometry do mesmo tipo (assinaturas (i), (iv) e (vi)). Como resultado, um objeto do mesmo tipo também é retornado, assim como definido no modelo exato VASA. É importante enfatizar que a intersecção de duas linhas vagas, irá retornar uma outra linha vaga com as mesmas propriedades. Para capturar os pontos em comuns de duas linhas vagas, é criado outra função específica chamada *CommonPoints* (Seção 4.3.4).

A operação de união tem outras duas variações, sendo que em (ii) recebe-se um vetor de *n* elementos do mesmo tipo e em (iii) é uma função de agregação que pode receber elementos do mesmo tipo a serem agrupados pela união. Em ambas as funções, a função (i) é chamada

recursivamente, assim, quando elementos de tipos distintos são detectados, a extensão retornará uma mensagem de erro.

De acordo com o modelo exato VASA, a operação de intersecção pode também ter outra variação, onde objetos de tipos diferentes podem ser passados como parâmetro e o resultado será um objeto VagueGeometry de menor dimensão (função (v)). Por exemplo, a intersecção entre pontos vagos e linhas vagas resultará em pontos vagos, uma vez que pontos tem menor dimensão que linhas. Porém, um erro na execução desta operação foi encontrado no processo de validação do TAD VagueGeometry, o qual é detalhado como segue. Sejam A e B dois objetos espaciais vagos, a intersecção é definida formalmente pelo modelo exato VASA (PAULY; SCHNEIDER, 2010) como

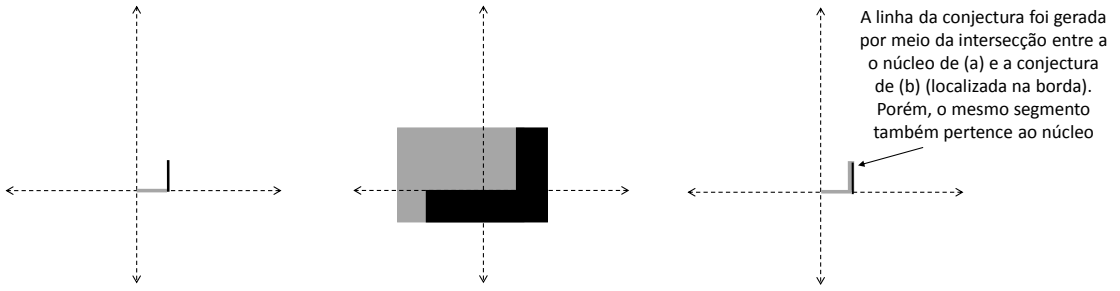
$$intersection(A,B) = (A_n \otimes B_n, (A_c \otimes B_c) \oplus (A_n \otimes B_c) \oplus (A_c \otimes B_n))$$

Onde $\otimes$ e $\oplus$ denotam a intersecção geométrica e união geométrica, respectivamente. Quando A e B são do mesmo tipo, a definição da intersecção gera o resultado esperado (como mostrado na Figura 4.3). Por outro lado, o modelo exato VASA afirma que com esta mesma definição é também possível processar a intersecção considerando tipos distintos. Porém, esta definição pode gerar resultados inconsistentes. Isto ocorre quando existem pontos no núcleo do primeiro objeto que são os mesmos pontos da conjectura e do núcleo do segundo objeto (i.e. o núcleo e a conjectura do segundo objeto são adjacentes). Nesta situação, o resultado tem pontos que pertencem ao núcleo e conjectura ao mesmo tempo. Todavia, isso não é permitido na álgebra VASA. A Figura 4.4 ilustra um exemplo deste caso, para a intersecção entre uma linha vaga e uma região vaga.

Visando corrigir essa inconsistência, a operação de intersecção foi redefinida como segue. A operação de diferença é realizada para evitar os mesmos pontos interiores no núcleo e na conjectura do resultado. Assim, nesse caso, esses pontos irão somente pertencer a parte do núcleo, uma vez que o núcleo possui uma maior certeza que a conjectura e dessa forma, deve ser mantido. Com isso, a operação de intersecção é formalmente redefinida como

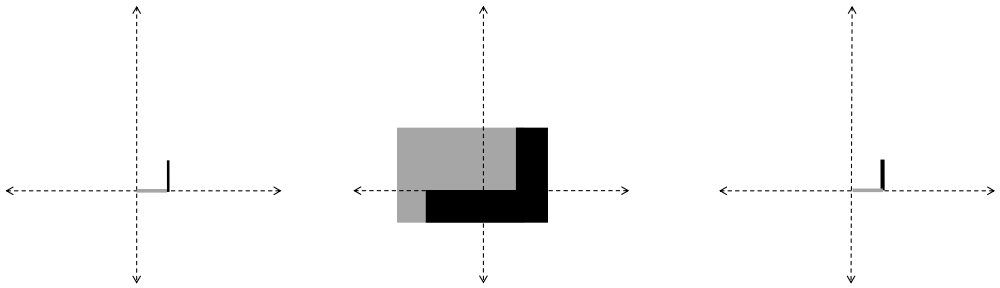
$$intersection(A,B) = (A_n \otimes B_n, ((A_c \otimes B_c) \oplus (A_n \otimes B_c) \oplus (A_c \otimes B_n)) \ominus (A_n \otimes B_n))$$

Onde $\ominus$ denota a operação de diferença geométrica. A Figura 4.5 ilustra o resultado correto produzido pela nova fórmula de intersecção entre os objetos espaciais vagos (a) e (b) da Figura 4.4.



(a) (b) Resultado Inconsistente

Figura 4.4: Intersecção entre uma linha vaga (a) e uma região vaga (b), gerando um resultado inconsistente.



(a) (b) Resultado Esperado

Figura 4.5: O resultado esperado para a intersecção entre uma linha vaga (a) e uma região vaga (b) utilizando a nova definição.

4.3.4 Operações Específicas de Tipos

As operações específicas de tipos têm como objetivo fazer operações que consideram tipos específicos de objetos espaciais vagos. Por exemplo, operações que só são realizadas em polígonos vagos e não em linhas vagas. Seja p um objeto VagueGeometry do tipo VAGUEPOINT ou VAGUEMULTIPOINT, l um objeto VagueGeometry do tipo VAGUELINESTRING ou VAGUEMULTILINESTRING, r um objeto VagueGeometry do tipo VAGUEPOLYGON ou VAGUEMULTIPOLYGON, lr um objeto VagueGeometry do tipo VAGUELINESTRING, VAGUEMULTILINESTRING, VAGUEPOLYGON ou VAGUEMULTIPOLYGON e lp um objeto VagueGeometry do tipo VAGUEPOINT, VAGUEMULTIPOINT, VAGUELINESTRING ou VAGUEMULTILINESTRING, as operações específicas de tipos têm as seguintes assinaturas

(i) *VG_CommonBorder*(VagueGeometry lr, VagueGeometry r) $\rightarrow$ VagueGeometry

(ii) *VG_CommonPoints*(VagueGeometry l, VagueGeometry l) $\rightarrow$ VagueGeometry

(iii) *VG_ConjectureBoundary*(VagueGeometry r, VagueGeometry r) $\rightarrow$ VagueGeometry

(iv) *VG_KernelBoundary*(VagueGeometry r, VagueGeometry r) $\rightarrow$ VagueGeometry

(v) *VG_ConjectureConvexHull*(VagueGeometry p, VagueGeometry p) $\rightarrow$ VagueGeometry

(vi) *VG_KernelConvexHull*(VagueGeometry p, VagueGeometry p) $\rightarrow$ VagueGeometry

(vii) *VG_ConjectureInterior*(VagueGeometry l, VagueGeometry l) $\rightarrow$ VagueGeometry

(viii) *VG_KernelInterior*(VagueGeometry l, VagueGeometry l) $\rightarrow$ VagueGeometry

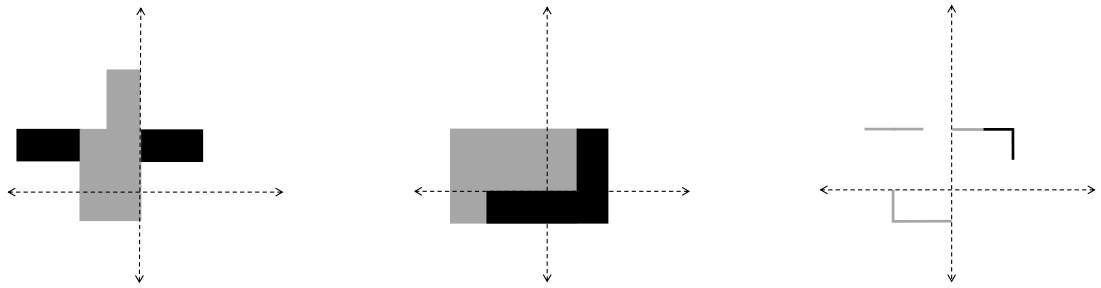
(ix) *VG_ConjectureVertices*(VagueGeometry lr, VagueGeometry lr) $\rightarrow$ VagueGeometry

(x) *VG_KernelVertices*(VagueGeometry lr, VagueGeometry lr) $\rightarrow$ VagueGeometry

A operação de captura da borda em comum (i) retorna a borda em comum entre duas regiões vagas, ou a intersecção entre uma linha vaga e a borda de uma região vaga. Ou seja, a borda em comum serão as linhas da intersecção considerando apenas a linha da borda das regiões vagas. Na Figura 4.6 é mostrado um exemplo desta operação sobre dois objetos VagueGeometry do tipo VAGUEMULTIPOLYGON (mesmas regiões vagas da Figura 4.2) que também mostra suas representações textuais VWKT no formato simplificado.

A operação de captura de pontos em comum (ii) retorna os pontos intersectados entre duas linhas vagas. A mesma definição da intersecção é utilizada, porém retornando pontos ao invés de linhas. Porém, por usar a mesma definição, o mesmo problema da operação da intersecção discutido na Seção 4.3.3 ocorre. A resolução para este caso, é o mesmo apresentado na Seção 4.3.3. Na Figura 4.7 é mostrado um exemplo desta operação sobre dois objetos VagueGeometry do tipo VAGUEMULTILINESTRING que também mostra suas representações textuais VWKT no formato simplificado.

O restante das operações tem variações da parte do núcleo (pelo prefixo *VG_Kernel*) e da parte da conjectura (pelo prefixo *VG_Conjecture*). A diferença entre eles é que na resposta final da operação, as operações com prefixo *VG_Kernel* irá fazer a diferença entre a conjectura e núcleo, enquanto as operações com prefixo *VG_Conjecture* irá fazer a diferença entre o núcleo e a conjectura. Apesar dessas operações retornarem valores muito idênticos, podem existir variações de elementos entre o núcleo e conjectura.



VAGUEMULTIPOLYGON(((-2 1, -2 2, -4 2, -4 1, -2 1)), ((0 1, 0 2, 2 2, 2 1, 0 1)); ((0 -1, -2 -1, -2 2, -1 2, -1 4, 0 4, 0 -1)))

(a)

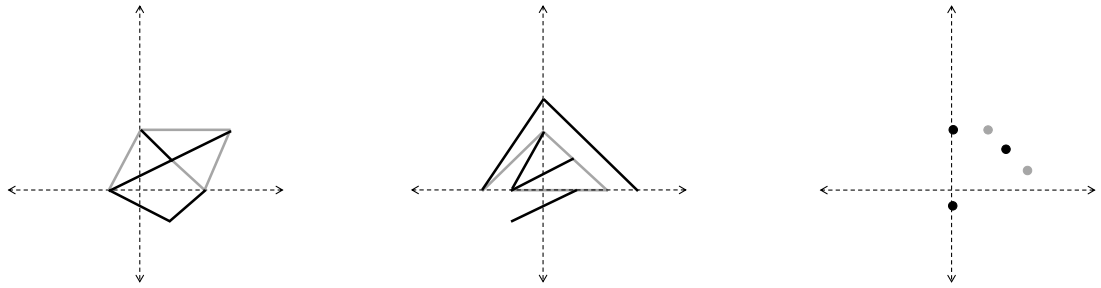
VAGUEMULTIPOLYGON(((-2 -1, -2 0, 1 0, 1 2, 2 2, 2 -1, -2 -1)); ((-3 -1, -3 2, 1 2, 1 0, -2 0, -2 -1, -3 -1)))

(b)

VAGUEMULTILINESTRING((1 2, 2 2),(2 2, 2 1); (0 -1, -2 -1),(-2 -1, -2 0),(-2 2, -1 2),(-2 2, -3 2),(0 2, 1 2))

Resultado *CommonBorder*

Figura 4.6: O resultado da operação *VG_CommonBorder* entre (a) e (b), com suas respectivas representações textuais VWKT no formato simplificado.



VAGUEMULTILINESTRING((2 0, 1 -1, -1 0, 3 2), (1 1, 0 2); (-1 0, 0 2, 3 2, 2 0, 1 1))

(a)

VAGUEMULTILINESTRING((-2 0, 0 3, 3 0), (-1 0, 0 2), (-1 0, 1 1), (-1 -1, 1 0); (-2 0, 0 2, 2 0, -1 0))

(b)

VAGUEMULTIPOINT(0 -0.5, 0 2, 1.6667 1.3333; 1 2, 2.3333 0.66667)

Resultado *CommonPoints*

Figura 4.7: O resultado da operação *VG_CommonPoints* entre (a) e (b), com suas respectivas representações textuais VWKT no formato simplificado.

As operações (iii) e (iv) capturam a borda de regiões vagas. As operações (v) e (vi) transformam pontos vagos em regiões vagas por meio da função *crisp* bem definida *convexhull*, a qual transforma as geometrias no menor polígono que as envolvem. As operações (vii) e (viii) transformam as linhas fechadas vagas (linhas vagas que tem um mesmo ponto inicial e final em pelo menos um de suas partes) em regiões vagas. Finalmente, as operações (ix) e (x) capturam os pontos finais de linhas vagas.

4.3.5 Predicados Topológicos Vagos

Devido à existência de um número muito alto de predicados topológicos envolvendo todas as combinações do núcleo e conjectura de dois objetos espaciais vagos, a álgebra VASA e o

TAD VagueGeometry definem e implementam 10 predicados topológicos que podem ser identificados mais facilmente semanticamente, além de ter nomes representativos. Tais nomes já são bem conhecidos devido a sua relação e o seu uso sobre os dados espaciais *crisp* no modelo 9-IM e DE-9IM (Seção 2.3 do Capítulo 2). Cada predicado topológico retorna um valor lógico que pode assumir um dos três valores: *true*, *maybe* e *false*. Por exemplo, o predicado *overlap* entre um objeto espacial vago *A* e um objeto espacial vago *B* pode retornar 3 valores lógicos distintos com diferentes significados. Caso o predicado retorne *true*, significa que a sobreposição com certeza ocorre. Ao retornar *maybe*, significa que a sobreposição talvez ocorra. Já com o retorno *false*, significa que a sobreposição com certeza não ocorre. O retorno, portanto, é um objeto do tipo VagueBool, implementado exatamente para este propósito.

Os predicados topológicos implementados são: *contains* (*contém*), *coveredBy* (*coberto por*), *covers* (*cobre*), *crosses* (*cruza*), *disjoint* (*disjounto*), *equals* (*igual*), *inside* (*dentro*), *intersects* (*intersecta*), *meets* (*toca*) e *overlap* (*sobreposição*). Sejam $vg1$ e $vg2$ objetos VagueGeometry de qualquer tipo, os predicados topológicos têm as seguintes assinaturas

(i) *VG_Contains*(VagueGeometry *vg1*, VagueGeometry *vg2*) → VagueBool

(ii) *VG_CoveredBy*(VagueGeometry *vg1*, VagueGeometry *vg2*) → VagueBool

(iii) *VG_Covers*(VagueGeometry *vg1*, VagueGeometry *vg2*) → VagueBool

(iv) *VG_Crosses*(VagueGeometry *vg1*, VagueGeometry *vg2*) → VagueBool

(v) *VG_Disjoint*(VagueGeometry *vg1*, VagueGeometry *vg2*) → VagueBool

(vi) *VG_Equals*(VagueGeometry *vg1*, VagueGeometry *vg2*) → VagueBool

(vii) *VG_Inside*(VagueGeometry *vg1*, VagueGeometry *vg2*) → VagueBool

(viii) *VG_Intersects*(VagueGeometry *vg1*, VagueGeometry *vg2*) → VagueBool

(ix) *VG_Meets*(VagueGeometry *vg1*, VagueGeometry *vg2*) → VagueBool

(x) *VG_Overlap*(VagueGeometry *vg1*, VagueGeometry *vg2*) → VagueBool

Sejam *A* e *B* dois objetos VagueGeometry, os predicados topológicos vagos são definidos intuitivamente como segue. O predicado *Contains* (i) é o inverso do predicado *Inside* (vii), ou seja, *VG_Inside*(*A*, *B*) é igual à *VG_Contains*(*B*, *A*). Assim, o predicado *Inside* (vii) retorna *true* se o interior da união do núcleo e conjectura de *A* está contido no interior do núcleo de *B* e não existe intersecção nas bordas. O predicado *Inside* retorna *false* se não há intersecção entre os

interiores de *A* e *B*, ou existem intersecção de bordas ou o núcleo de *A* intersecta o exterior da união do núcleo e conjectura de *B*. Caso contrário, o predicado *Inside* retorna *maybe*.

O predicado *Covers* (iii) é o inverso do predicado *CoveredBy* (vii), ou seja, *VG_CoveredBy*(*A*, *B*) é igual à *VG_Covers*(*B*, *A*). Assim, o predicado *CoveredBy* (ii) retorna *true* se o interior da união do núcleo e conjectura de *A* está contido no interior do núcleo de *B* e existe intersecções de borda entre os núcleos. O predicado *CoveredBy* retorna *false* se não há intersecção entre *A* e *B*, ou *A* está dentro de *B* (i.e. *Inside* é *true*) ou o núcleo de *A* intersecta o exterior da união do núcleo e conjectura de *B*. O predicado *CoveredBy* retorna *maybe* para todos os outros casos.

O predicado *Crosses* (iv) pode ser somente entre linhas vagas ou entre uma linha vaga e uma região vaga, retornando *false* caso contrário. O predicado *Crosses* retorna *true* se o núcleo de *A* cruza o núcleo de *B*. O predicado *Crosses* retorna *maybe* se a união do núcleo e conjectura de *A* cruza com a união do núcleo e conjectura de *B*. Caso contrário, o predicado retorna *false*.

O predicado *Disjoint* (v) retorna *true* se os núcleos e conjecturas de *A* e *B* são disjuntos. O predicado *Disjoint* retorna false se existe intersecção entre os núcleos de *A* e *B*. Caso contrário, o predicado retorna *maybe*. O predicado *Intersects* (viii) é a negação ou complemento do predicado *Disjoint*, ou seja, *VG_Disjoint*(*A*, *B*) é igual à !*VG_Intersects*(*A*, *B*).

O predicado *Equals* (vi) retorna *true* se os objetos *A* e *B* não tiverem conjecturas e os núcleos de *A* e *B* são iguais. O valor *false* é retornado quando o interior do núcleo de um objeto intersecta o exterior da união entre o núcleo e a conjectura do outro objeto. Para todos os outro casos, o predicado retorna *maybe*.

O predicado *Meets* (ix) retorna *true* se a borda do núcleo de *A* toca a borda do núcleo de *B* e o interior da união do núcleo e conjectura de *A* é disjunta do interior da união do núcleo e conjectura de *B*. O predicado retorna *false* se existe intersecção entre os interiores dos núcleos ou *A* é disjunto de *B* (i.e. *VG_Disjoint* é *true*). Caso contrário, o predicado retorna *maybe*.

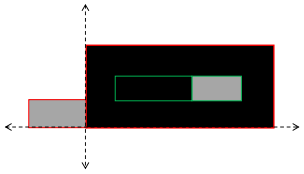
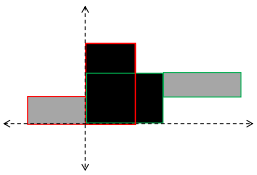
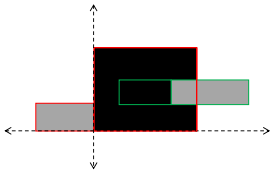
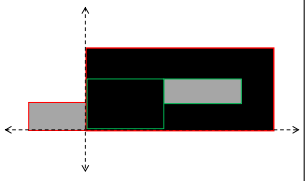
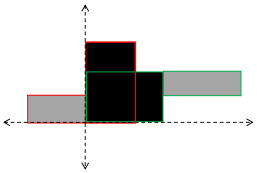
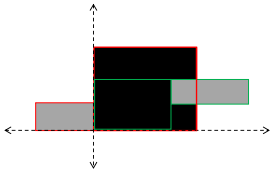
Por fim, o predicado *Overlap* (x) retorna *true* se existe intersecção entre os interiores dos núcleos e intersecção dos interiores dos núcleos com o exterior dos objetos. O predicado retorna *false* se não existe intersecção entre os interiores ou se os interiores e exteriores dos objetos não intersectam. Caso contrário, o predicado retorna *maybe*.

Na Tabela 4.3 são mostrados exemplos de quando cada predicado retorna os valores *true*, *false* e *maybe*. Em cada exemplo, dois objetos espaciais vagos são considerados, um com borda em vermelho e outro com borda em verde. Além disso, as situações de quando o predicado topológico vago retorna *true* e *false* são definidas utilizando a notação textual do modelo 9-IM

(Seção 2.3 do Capítulo 2), de acordo com as definições formais do modelo exato VASA (PAULY; SCHNEIDER, 2010). É importante enfatizar que o predicado *Crosses* foi proposto como adicional nesta dissertação e não está presente no modelo exato VASA. O valor *maybe* é retornado quando o predicado não retorna nem *true* e nem *false*, com exceção do predicado *Crosses*. Na notação textual do modelo 9-IM, o valor * simboliza que na posição correspondente o valor é desconsiderado. Os predicados *Contains*, *Covers* e *Intersects* não foram considerados na Tabela 4.3 por serem obtidos a partir de outros predicados, como detalhado anteriormente.

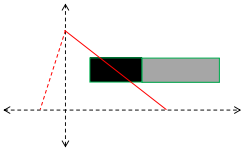
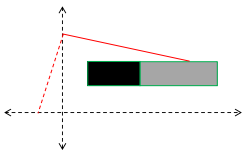
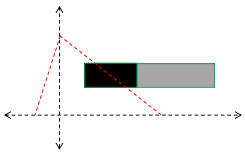
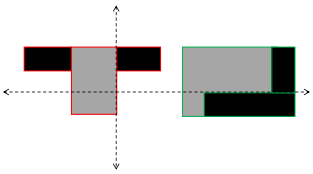
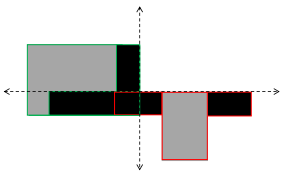
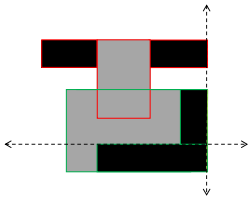
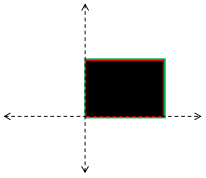
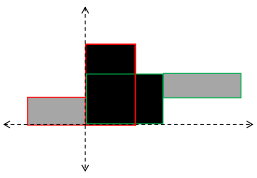
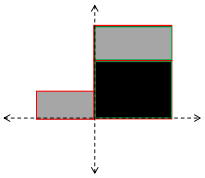
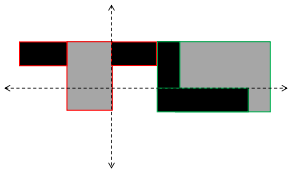
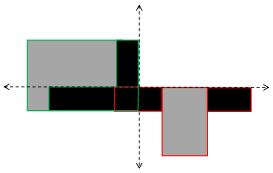
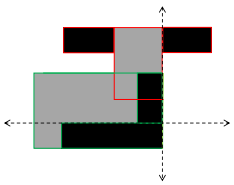
Os relacionamentos do modelo 9-IM entre dois objetos espaciais vagos A e B considerados são: p que simboliza o relacionamento entre os núcleos; q que simboliza o relacionamento entre a união do núcleo e conjectura de A com o núcleo de B; r que simboliza o relacionamento entre o núcleo de A com a união do núcleo e conjectura de B; e, s que simboliza o relacionamento entre a união do núcleo e conjectura de A com a união do núcleo e conjectura de B. Portanto, para realizar os relacionamentos q, r e s, é necessário também a operação de união entre o núcleo e conjectura do respectivo objeto espacial vago.

Tabela 4.3: Definições e exemplos dos predicados topológicos vagos implementados pelo TAD VagueGeometry.

Predicado Topológico Vago	**Retorno *true***	**Retorno *false***	**Retorno *maybe***
Inside	 se q = "T*F*FF***"	 se não é *true* e (r = "**T******" ou r = "****T****")	 se não é *true* ou *false*
Coveredby	 se (q = "T*F**F***" ou q = "*TF**F***" ou q = "**FT*F***" ou q = "**F*TF***") e p = "****T****"	 se não é *true* e (r = "**T******" ou r = "T*F*FF***")	 se não é *true* ou *false*

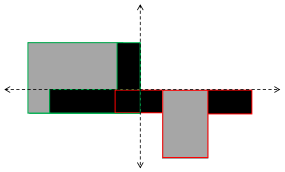
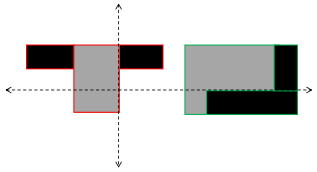
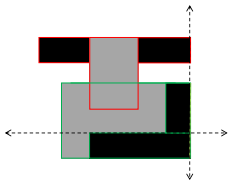
Continua na próxima página

Tabela 4.3 – *Continuado a partir da página anterior*

Predicado Topológico Vago	**Retorno *true***	**Retorno *false***	**Retorno *maybe***
Crosses	 se p = “T*T******” ou p = “T*****T**”	 se não é *true* ou *maybe*	 se não é *true* e (s = “T*T******” ou s = “T*****T**”)
Disjoint	 se não é *false* e s = “FF*FF****”	 se p != “FF*FF****”	 se não é *true* ou *false*
Equals	 se p = “T*F**FFF*” e sem conjecturas em ambos os objetos	 se não é *true* e (r = “**T******” ou q = “******T**”)	 se não é *true* ou *false*
Meets	 se (p = “FT*******” ou p =“F**T*****” ou p =“F***T****”) e s = “F********”	 se não é *true* e (p = “T********” ou s = “FF*FF****”)	 se não é *true* ou *false*

Continua na próxima página

Tabela 4.3 – *Continuado a partir da página anterior*

Predicado Topológico Vago	**Retorno *true***	**Retorno *false***	**Retorno *maybe***
Overlap	 se p = "T*T***T**" e r = "**T******" e q = "******T**"	 se não é *true* e (s = "F********" ou q = "**F******" ou r = "******F**")	 se não é *true* ou *false*

4.3.6 Operações Numéricas

A última categoria de operações que a extensão provê suporte são os operadores numéricos. Em geral, esses operadores retornam um objeto VagueNumeric. Seja r um objeto VagueGeometry do tipo VAGUEPOLYGON ou VAGUEMULTIPOLYGON, l um objeto VagueGeometry do tipo VAGUELINESTRING ou VAGUEMULTILINESTRING, $vg1$ e $vg2$ objetos VagueGeometry de qualquer tipo, os operadores numéricos têm as seguintes assinaturas

(i) *VG_Area*(VagueGeometry r) $\rightarrow$ VagueNumeric

(ii) *VG_Length*(VagueGeometry l) $\rightarrow$ VagueNumeric

(iii) *VG_NearestDistance*(VagueGeometry *vg1*, VagueGeometry *vg2*) $\rightarrow$ VagueNumeric

(iv) *VG_FarthestDistance*(VagueGeometry *vg1*, VagueGeometry *vg2*) $\rightarrow$ VagueNumeric

Para as operações numéricas que envolvem somente um objeto espacial vago (i) e (ii), o valor mínimo do objeto VagueNumeric corresponde ao valor da operação numérica referente ao núcleo, enquanto o valor máximo corresponde a operação numérica aplicada sobre a união do núcleo com a conjectura.

Já para a operação de distância mínima (iii) entre dois objetos espaciais vagos, o valor mínimo corresponde a distância mínima considerando a união do núcleo e conjectura, enquanto que para o valor máximo somente o núcleo. Esta diferença ocorre, pois a distância pode ser

reduzida ao se considerar a conjectura. Para a distância máxima (iv) entre dois objetos espaciais vagos, o valor máximo corresponde a distância máxima considerando a união do núcleo e conjectura, enquanto que para o valor mínimo somente o núcleo. Esta diferença ocorre, pois a distância máxima pode ser ainda maior ao se considerar a união do núcleo e conjectura.

4.3.7 Operadores

Assim como proposto pela álgebra VASA (Seção 3.5.1 do Capítulo 3), quando um predicado topológico é chamado por meio da linguagem SQL sem uso de qualquer operador junto ao predicado, retornará *true* se o relacionamento topológico realmente existir. Ou seja, em uma consulta onde na clausula WHERE tiver a seguinte condição VG_Intersects(*col1*, *col2*), só irão retornar as tuplas onde objetos espaciais vagos de *col1* com certeza intersectar os objetos espaciais vagos de *col2*. Porém, se a consulta necessitar retornar também tuplas das quais talvez ocorra a intersecção, é preciso utilizar um operador específico. Nesse sentido, foram definidos operadores específicos para manipular valores retornados pelos predicados topológicos vagos, mais especificamente os objetos VagueBool. Ou seja, dessa forma, os operadores retornam *true* ou *false* dependendo do contexto com o objetivo de tornar possível a manipulação de três valores lógicos em condições que retornam somente *true* ou *false*. Tais operadores são listados a seguir

- O operador *maybe* $\sim$: retornará verdadeiro se o predicado talvez ocorra ou com certeza ocorra. Caso contrário, ele retornará falso. Um exemplo de seu uso é $\sim$*VG_Intersects*(*col1*, *col2*).
- O operador *really maybe* $\sim\sim$: retornará verdadeiro se o predicado talvez ocorra. Caso contrário, ele retornará falso. Um exemplo de seu uso é $\sim\sim$*VG_Intersects*(*col1*, *col2*).
- O operador *not* !: retornará verdadeiro se o predicado com certeza não ocorrer. Caso contrário, ele retornará falso. Um exemplo de seu uso é !*VG_Intersects*(*col1*, *col2*).

É importante salientar que existem situações para as quais não sejam interessante o uso dos operadores conforme a existência do núcleo e conjectura de dois objetos espaciais vagos A e B. Tais situações são: (i) quando A e B têm somente conjecturas, portanto, um predicado nunca retornará *true* e então não é interessante utilizar um predicado sem o uso de um operador; e, (ii) quando A e B têm somente núcleos, um predicado nunca retornará *maybe* e com isso não é interessante utilizar o operador *really maybe*.

Além desses operadores, também existem operadores para manipular os três valores lógicos respeitando a tabela verdade (definida em (PAULY; SCHNEIDER, 2010)) e são listados a seguir

- O operador &&: realiza a operação lógica AND entre dois objetos do tipo VagueBool e pode ser usado combinando predicados topológicos. Por exemplo, para restringir que somente deve-se retornar tuplas as quais os objetos espaciais vagos de *col1* contém os objetos de *col2* e que os objetos de *col2* também intersectam os objetos de *col3*: *VG_Contains*(*col1*, *col2*) && *VG_Intersects*(*col2*, *col3*). Adicionalmente, é possível combiná-lo com os operadores anteriormente apresentados.

- O operador ||: realiza a operação lógica OR entre dois objetos do tipo VagueBool e pode ser usado combinando predicados topológicos. Por exemplo, para restringir que somente deve-se retornar tuplas as quais os objetos espaciais vagos de *col1* contém os objetos de *col2* ou que os objetos de *col2* intersectam os objetos de *col3*: *VG_Contains*(*col1*, *col2*) || *VG_Intersects*(*col2*, *col3*). Adicionalmente, é possível combiná-lo com os operadores anteriormente apresentados.

Por fim, assim como foram definidos operadores para manipular objetos VagueBool, também foram definidos operadores para manipular objetos do tipo VagueNumeric retornados pelas operações numéricas envolvendo objetos espaciais vagos. Assim comparações podem ser feitas para restringir resultados pelo retorno positivo ou negativo do operador. Estes operadores são listados a seguir

- O operador *maybe* $\sim$: é usado entre um valor numérico e uma operação numérica envolvendo objetos espaciais vagos e retornará verdadeiro se o valor numérico estiver entre o valor mínimo e máximo do retorno da função numérica (ou seja, o VagueNumeric de retorno). Um exemplo é usá-lo para restringir tuplas das quais os objetos de *col1* tenha uma área com cerca de 800: *VG_Area*(*col1*) $\sim$ 800.

- O operador *equals* =: é usado entre um valor numérico e uma função numérica envolvendo objetos espaciais vagos e retornará verdadeiro se o valor for igual ao valor mínimo da função numérica (ou seja, o VagueNumeric de retorno). Um exemplo é usá-lo para restringir tuplas das quais os objetos de *col1* com certeza tenha uma área de no mínimo 800: *VG_Area*(*col1*) = 800.

4.4 Melhoramentos na Implementação do TAD VagueGeometry

Nesta seção são apresentadas melhorias para o processamento de predicados topológicos vagos, uma vez que este processamento pode ser custoso em consultas espaciais. Na Seção 4.4.1 é detalhada a primeira melhoria realizada no TAD VagueGeometry, a qual armazena a união entre o núcleo e a conjectura de um objeto espacial vago visando reduzir o tempo de processamento dos relacionamentos topológicos. Na Seção 4.4.2 é detalhada a segunda melhoria, a qual utiliza MBRs para evitar o processamento dos predicados topológicos vagos em determinadas situações. É importante enfatizar que uma avaliação experimental para medir o desempenho do TAD VagueGeometry considerando estas melhorias propostas foi conduzida e seus resultados são discutidos no Capítulo 5.

4.4.1 Armazenamento da União Entre o Núcleo e a Conjectura

Esta melhoria visa reduzir o número de operações de união realizadas no processamento de um predicado topológico vago do modelo exato VASA. Como discutido na Seção 4.3.5, as operações de união entre o núcleo e a conjectura são realizadas para extrair os relacionamentos topológicos q, r e s por meio da matriz 9-IM. Ou seja, sejam A e B dois objetos espaciais vagos, e *9IM* a função que extraí o relacionamento topológico de dois objetos espaciais *crisp*, o relacionamento q é igual a $9IM(A_n \oplus A_c, B_n)$, o relacionamento r é igual a $9IM(A_n, B_n \oplus B_c)$ e o relacionamento s é igual a $9IM(A_n \oplus A_c, B_n \oplus B_c)$.

Dessa forma, esta melhoria pré-armazena a união entre o núcleo e a conjectura de um objeto espacial vago para auxiliar na extração dos relacionamentos topológicos q, r e s. Intuitivamente, a união entre o núcleo e a conjectura de um objeto espacial vago corresponde a sua total extensão possível. Com isso, um objeto espacial vago é definido como dois objetos espaciais *crisp* de tipos iguais, disjuntos ou adjacentes para representar o núcleo e a conjectura e outro objeto espacial *crisp* que representa a união entre o núcleo e a conjectura. É importante notar que o objeto espacial que representa a união, pode não ser do mesmo tipo de dado do núcleo e a conjectura, porém de mesma dimensionalidade. Por exemplo, para uma região simples vaga que contém um núcleo disjunto da conjectura, a sua união resultará em uma região complexa *crisp*. Seja A um objeto espacial vago que pré-armazena a sua união, ele é formalmente definido como $A = (\langle A_n, A_c \rangle, A_e)$, onde A_n corresponde ao núcleo, A_c corresponde a conjectura e A_e corresponde a sua total extensão possível (i.e. $A_e = A_n \oplus A_c$).

Com esta abordagem, é possível realizar os relacionamentos topológicos q, r e s da seguinte

forma. Sejam A e B dois objetos espaciais vagos que pré-armazenam a união entre o núcleo e a conjectura. O relacionamento q é igual a $9IM(A_e, B_n)$, o relacionamento r é igual a $9IM(A_n, B_e)$ e o relacionamento s é igual a $9IM(A_e, B_e)$.

Para armazenar a união entre o núcleo e a conjectura de um objeto espacial vago no TAD VagueGeometry, as suas estruturas de armazenamento em memória (a estrutura VAGUEGEOM) e a serialização em disco foram alteradas como segue. No Algoritmo 9, o elemento *all_extension* é adicionado a estrutura VAGUEGEOM do Algoritmo 1 para armazenar a união entre o núcleo (elemento *kernel*) e a conjectura (elemento *conjecture*) de um objeto espacial vago. O elemento *all_extension* é diferente de vazio somente quando um objeto espacial vago contém um núcleo e uma conjectura. Isso significa que a união entre o núcleo e a conjectura é somente processada quando estes objetos existem. Além disso, o elemento *flags* armazena também se um objeto contém a sua união pré-armazenada. Com isso, o armazenamento da união do núcleo e sua conjectura pode ser opcional, de acordo com um parâmetro passado pelo usuário no momento da criação de um objeto VagueGeometry.

Algoritmo 9 Estrutura de dados para armazenar um objeto VagueGeometry em memória principal que pré-armazena a união entre o núcleo e conjectura

```
1: typedef struct {
2:   uint8_t type;
3:   uint8_t flags;
4:   LWGEOM *kernel;
5:   LWGEOM *conjecture;
6:   LWGEOM *all_extension;
7: } VAGUEGEOM;
```

Dessa forma, o elemento *all_extension* é somente serializado e armazenado na estrutura VGSERIALIZED (Algoritmo 2), quando o elemento *flags* de um objeto VAGUEGEOM indicar que o elemento *all_extension* é diferente de vazio. Portanto, o elemento *all_extension* é somente serializado quando o usuário desejar e quando o núcleo e a conjectura de um objeto VagueGeometry forem diferentes de vazio. Este elemento é serializado após a serialização dos objetos do núcleo e a conjectura, e também reutiliza a serialização de objetos espaciais *crisp* do PostGIS. Como resultado, um objeto VagueGeometry, que pré-armazena a união entre o núcleo e a conjectura, requer mais espaço de armazenamento. Além disso, o tamanho em *bytes* da conjectura também é armazenado, sendo possível o acesso direto ao elemento *all_extension* quando necessário. Na Figura 4.8 é mostrado a ordem de serialização de objetos VagueGeometry que pré-armazenam a união entre o núcleo e a conjectura. É importante notar que o alinhamento de 8 *bytes* ainda é mantido por meio da inclusão de *paddings*, conforme discutido na Seção 4.2.1. Cada partição destacada na Figura 4.8 representa um alinhamento de 8 *bytes*.

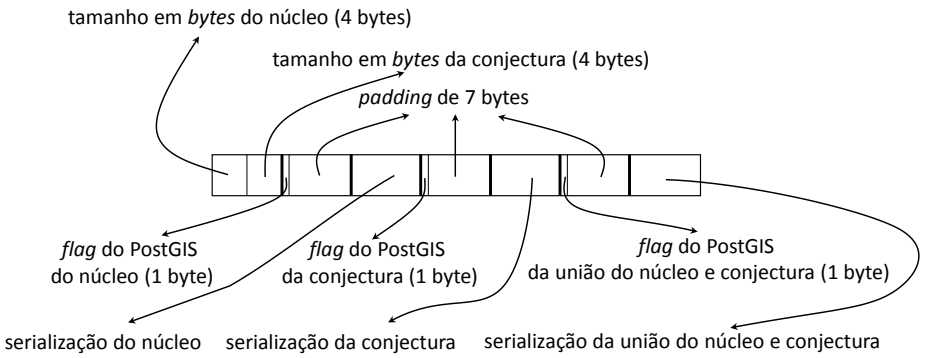


Figura 4.8: Ordem de serialização de um objeto VagueGeometry que pré-armazena a união entre o núcleo e conjectura.

Visando a possibilidade do usuário determinar se deseja armazenar a união entre o núcleo e a conjectura de um objeto VagueGeometry, as funções de entradas foram adaptadas. Um parâmetro booleano é adicionado a estas funções, onde é indicado se o objeto espacial vago irá armazenar ou não a sua união, ou seja, a sua total extensão possível. Por exemplo, as funções *VG_VagueGeomFromText* e *VG_MakeVagueGeom* possuem as seguintes assinaturas

(i) *VG_VagueGeomFromText*(text *VWKT*, integer *SRID*, boolean *punion*) → VagueGeometry

(ii) *VG_MakeVagueGeom*(Geometry *n*, Geometry *c*, boolean *punion*) → VagueGeometry

A função *VG_isPreComputingUnion* também foi implementada para saber se um objeto VagueGeometry está pre-armazenando a sua união. Além disso, a função *VG_PreComputeUnion* também foi definida para armazenar (segundo parâmetro como *true*) ou excluir (segundo parâmetro como *false*) a união entre o núcleo e a conjectura de um objeto VagueGeometry. Seja *vg* um objeto VagueGeometry, tais funções possuem as seguintes assinaturas

(i) *VG_isPreComputingUnion*(vg *VagueGeometry*) → boolean

(ii) *VG_PreComputeUnion*(vg *VagueGeometry*, boolean *punion*) → VagueGeometry

4.4.2 Uso de MBRs nos Predicados Topológicos Vagos

Esta melhoria visa utilizar MBRs para processar um predicado topológico vago somente quando necessário. Esta abordagem utiliza os mesmos fundamentos das bases de dados espaciais *crisp*, que usam aproximações para determinar quando um predicado deve ser processado. A aproximação mais utilizada é o MBR, devido a sua complexidade simples. O PostGIS provê

suporte e faz uso de MBRs para o processamento de relacionamentos topológicos. Dessa forma, o TAD VagueGeometry também propõe o uso de MBRs para objetos espaciais vagos.

Os MBRs do núcleo e da conjectura de um objeto espacial vago são usados para determinar quando um predicado topológico deve ser realmente processado. Dessa forma, com o uso de MBRs é possível determinar o retorno prematuro de predicados topológicos em determinadas situações. São abordadas duas formas de uso dos MBRs em dados espaciais vagos. Sejam A e B dois objetos espaciais vagos, MBR_{Ae} e MBR_{Be} MBRs que correspondem a união entre o núcleo e a conjectura de A e a união entre o núcleo e a conjectura de B respectivamente, MBR_{An} e MBR_{Bn} MBRs do núcleo de A e do núcleo de B, e, MBR_{Ac} e MBR_{Bc} MBRs da conjectura de A e da conjectura de B. A primeira forma de uso é em relação a disjunção. Neste caso, são testadas as seguintes situações

(i) MBR_{Ae} e MBR_{Be} são disjuntos, i.e. $MBR_{Ae} \cap MBR_{Be} = \varnothing$; ou,

(ii) MBR_{An} e MBR_{Ac} são disjuntos de MBR_{Bn} e MBR_{Bc}, i.e. $MBR_{An} \cap MBR_{Bn} = \varnothing \wedge MBR_{An} \cap MBR_{Bc} = \varnothing \wedge MBR_{Ac} \cap MBR_{Bn} = \varnothing \wedge MBR_{Ac} \cap MBR_{Bc} = \varnothing$.

Na Figura 4.9a é mostrada a situação (i), a qual é a primeira testada uma vez que testa a disjunção somente entre dois MBRs. Se tais MBRs são disjuntos, é possível retornar *true* para o predicado *disjoint*. Caso contrário, a situação (ii) é testada. Esta situação é mostrada na Figura 4.9b, onde os MBRs que correspondem pela união dos núcleos e conjecturas de cada objeto intersectam (representados pelos retângulos pontilhados), porém todos os MBRs dos núcleos e conjecturas de cada objeto são disjuntos. Com isso, é possível retornar *true* para o predicado *disjoint* e *false* para os outros predicados. Se as duas situações testadas não ocorrerem, o predicado em questão deve ser processado. A disjunção dos MBRs foi implementada para os seguintes predicados: *disjoint*, *meets*, *crosses*, *intersects*, *equals* e *overlap*.

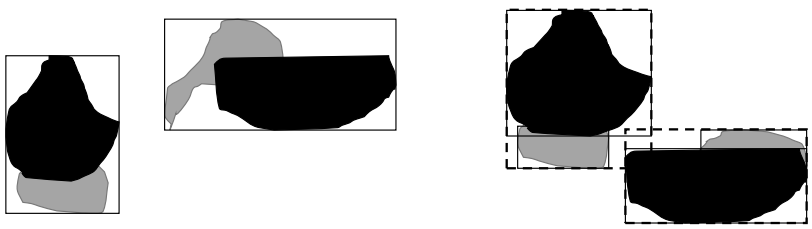


Figura 4.9: Teste de disjunção entre os MBRs. A situação (a) mostra que os MBRs das uniões do núcleo e conjectura dos objetos são disjuntos, enquanto a situação (b) mostra que os MBRs do núcleo e conjectura de um objeto é disjunto dos MBRs do núcleo e conjectura do outro objeto.

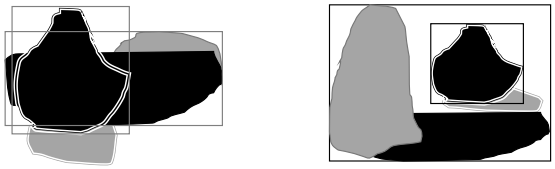
A segunda abordagem de uso dos MBRs é utilizado pelos predicados *coveredBy* e *inside*. Uma vez que os predicados *covers* e *contains* são o inverso desses predicados, eles também usam esta mesma forma de uso dos MBRs. Assim, são testadas as seguintes situações

(i) MBR_{An} não está contido em MBR_{Be}, i.e. $MBR_{An} \nsubseteq MBR_{Be}$; ou,

(ii) MBR_{An} é disjunto de MBR_{Bn} e MBR_{Bc}, i.e. $MBR_{An} \cap MBR_{Bn} = \varnothing \wedge MBR_{An} \cap MBR_{Bc} = \varnothing$.

Na Figura 4.10a é mostrado a situação (i), a qual testa se o MBR do núcleo de A não está contido no MBR da união do núcleo com a conjectura de B. Se esta situação ocorre, é possível retornar *false* para os predicados *inside* e *coveredBy*, uma vez que o interior do núcleo de A intersecta o exterior da união entre o núcleo e a conjectura de B. Caso contrário, a situação (ii) é testada. Esta situação é mostrada na Figura 4.10b, onde o MBR do núcleo de A é disjunto dos MBRs do núcleo e da conjectura de B. É importante notar que esta situação pode ocorrer mesmo que a situação (i) não ocorra. Caso a situação (ii) ocorra, é possível retornar *false* para os predicados *inside* e *coveredBy*. Se as duas situações testadas não ocorrerem, o predicado em questão deve ser processado.

Para ser possível o uso dos MBRs no processamento dos predicados topológicos vagos, foi necessário realizar adaptações na etapa de serialização de um objeto VagueGeometry. A estrutura em memória VAGUEGEOM não foi alterada, uma vez que os objetos do PostGIS, os quais são utilizados por esta estrutura, mantêm também seus respectivos MBRs. Assim, a serialização de um objeto VagueGeometry também armazenou os MBRs do núcleo e da conjectura e o MBR da total extensão possível, quando o objeto VagueGeometry armazena a união entre o núcleo e conjectura (Seção 4.4.1). Para computar o MBR correspondente a união entre o núcleo e a conjectura de um objeto VagueGeometry que não armazena esta informação, a união entre os MBRs do núcleo com a conjectura é executada. Na Figura 4.11a é mostrada a ordem de serialização de objetos VagueGeometry que não pré-armazenam a união entre o



(a) (b)

Figura 4.10: Teste entre os MBRs para os predicados *inside* e *coveredBy*. A situação (a) mostra que o MBR do núcleo do objeto com borda dupla não está contido no MBR correspondente a união do núcleo e conjectura do segundo objeto, enquanto a situação (b) mostra que o MBR do núcleo do objeto com borda dupla é disjunto dos MBRs do núcleo e conjectura do outro objeto.

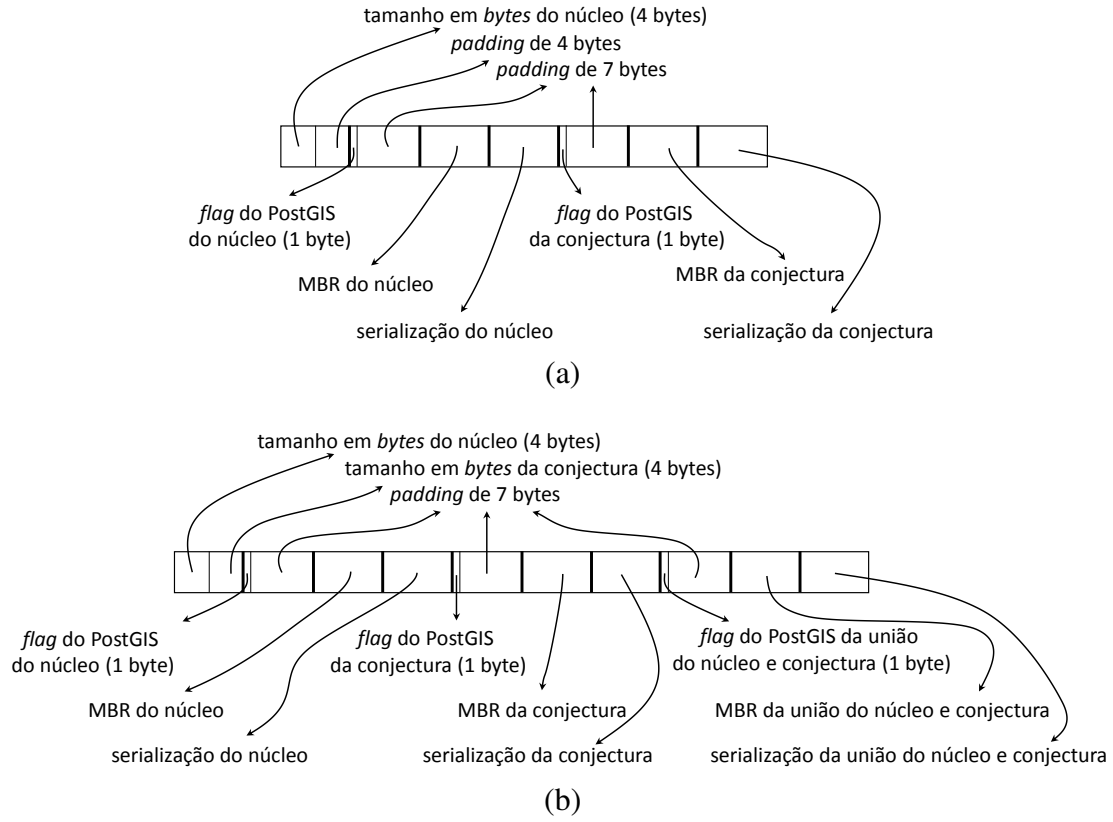


Figura 4.11: Ordem de serialização de um objeto VagueGeometry para armazenar MBRs. A ordem (a) é usada para objetos VagueGeometry que não pré-armazenam a união entre o núcleo e conjectura, enquanto (b) considera este pré-armazenamento na serialização.

núcleo e a conjectura, enquanto na Figura 4.11b é mostrada a ordem de serialização de objetos VagueGeometry que pré-armazenam esta informação. É importante notar que o alinhamento de 8 *bytes* ainda é mantido, uma vez que um MBR é composto por 4 coordenadas. Cada partição destacada na Figura 4.11 representa um alinhamento de 8 *bytes*.

4.5 Exemplo de Aplicação

O objetivo dessa seção é demonstrar com exemplos, o uso das operações do TAD VagueGeometry. O contexto do exemplo de aplicação é gerenciar o plantio de culturas considerando características do solo em um ambiente agrícola.

Dessa forma, o usuário mantém um controle sobre as plantações cultivadas considerando aspectos como a textura do solo, cultura cultivada, locais de pragas e rotas de animais. Além disso, lagos também são armazenados. Nesse sentido, as seguintes relações são consideradas

textura(*id*:integer, *textura*:vagueregion, *descricao*:text)

animal(*id*:integer, *rota*:vagueline, *descricao*:text, *animal*:text)

praga(*id*:integer, *praga*:vaguepoint, *descricao*:text, *tipo*:text)

plantacao(*id*:integer, *plantacao*:vagueregion, *descricao*:text)

lago(*id*:integer, *lago*:vagueregion, *descricao*:text)

Em geral, o atributo *id* é a chave primária de cada relação e o atributo *descricao* é a descrição espacial do fenômeno. A *textura*, *plantacao* e *lago* são representados por regiões vagas (i.e. polígonos vagos), uma vez que eles não tem bordas bem definidas. O atributo *rota* na relação *animal* é representada por linhas vagas uma vez que todas as rotas de um animal não são bem conhecidas. O atributo *praga* da relação *praga* é representado por pontos vagos indicando a possibilidade de um ponto ser uma praga em potencial. Por fim, os atributos *animal* na relação *animal* e *tipo* na relação *praga* são somente para informação adicional. Na Figura 4.12 é ilustrado o exemplo de aplicação considerado. Nesta figura, as partes de cores mais claras representam a conjectura, enquanto as cores mais escuras o núcleo. Já para as rotas de animais, as linhas contínuas representam o núcleo e as linhas tracejadas representam a conjectura.

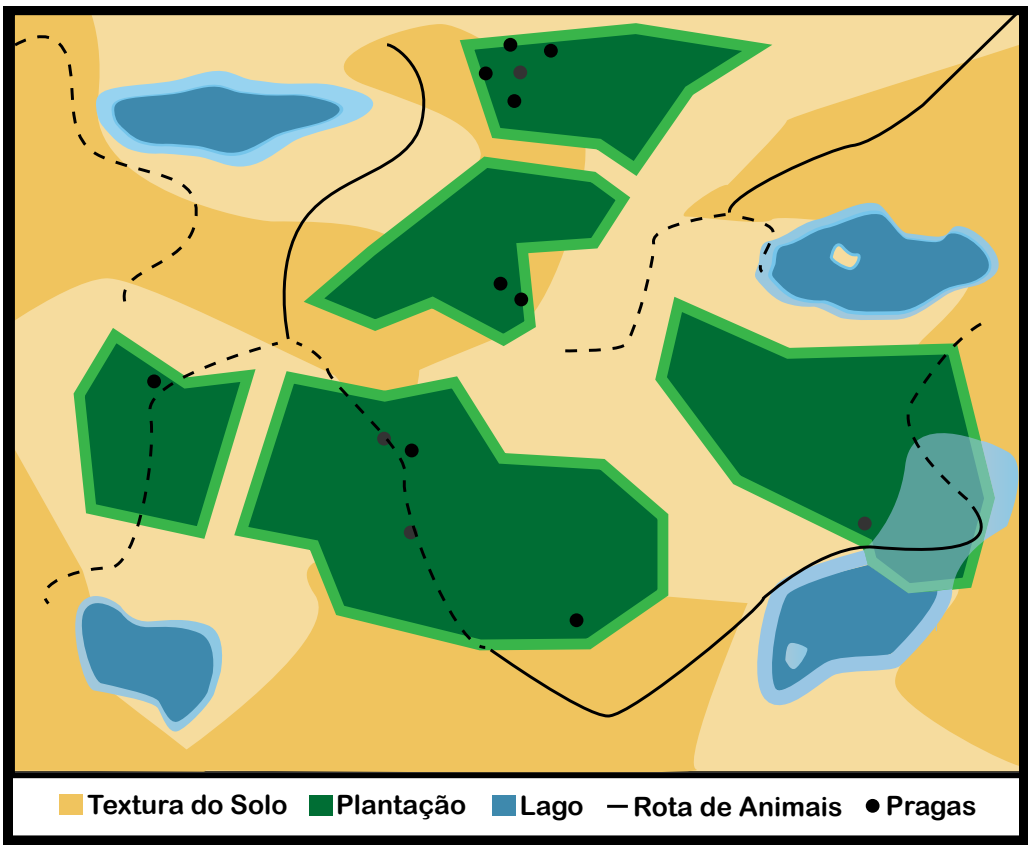


Figura 4.12: Exemplo de aplicação de um ambiente agrícola contendo objetos espaciais vagos.

Com as relações previamente definidas e com o TAD VagueGeometry, é possível criar as tabelas com os atributos espaciais vagos usando os tipos de dados do TAD VagueGeometry. O código SQL para criar a tabela relacional da relação *textura* é

```
CREATE TABLE textura(
id INTEGER PRIMARY KEY,
textura VAGUEGEOMETRY(VAGUEPOLYGON),
descricao TEXT);
```

É importante notar a restrição em parênteses no atributo *textura*, o qual aceitará somente polígonos vagos como entrada, uma vez que essa é uma restrição definida na especificação do exemplo de aplicação. A criação das outras relações são similares. Porém, mudando a restrição para seu respectivo tipo de dado. Por exemplo, para a relação *praga*, o atributo *praga* é criado como `VAGUEGEOMETRY(VAGUEMULTIPOINT)`, uma vez que uma praga pode ser representada por vários pontos vagos.

Após a criação das tabelas, é possível inserir objetos VagueGeometry. Existem duas diferentes formas para inserir um objeto VagueGeometry em uma tabela. A primeira delas é usar a forma canônica provida pelo SGBD PostgreSQL, enquanto que a segunda forma é por meio do uso de funções específicas do TAD VagueGeometry (Seção 4.3.1). Considerando a tabela *praga*, um multiponto vago representando a praga do tipo X é inserido, por meio da forma canônica e formato textual VWKT em sua forma simplificada, usando o código SQL a seguir

```
INSERT INTO praga
VALUES (1,
'VAGUEMULTIPOINT((1 1, 3 3); (2 2, 5 5))'::VAGUEGEOMETRY,
'severa', 'tipo X')
```

É possível usar funções de conversões textuais e binárias para o formato interno para operação de inserção. Por exemplo, um outro multiponto vago pode ser inserido na tabela praga utilizando o formato VWKT em sua forma simplificada e a função *VG_VagueGeomFromText* conforme o comando SQL a seguir

```
INSERT INTO praga
VALUES (1,
VG_VagueGeomFromText('VAGUEMULTIPOINT((1 5, 2 3); (0 2, 2 5))', 4326),
'severa', 'tipo X')
```

Considerando que as tabelas têm todos os objetos inseridos, é possível executar consultas SQL sobre os objetos VagueGeometry. Uma possível consulta consiste em retornar todas as partes em comum entre as rotas de animais e as áreas de uma plantação. Essa consulta usa a operação de intersecção das tabelas *animal* e *plantacao* para saber onde os animais podem ir em cada área plantada. Essa consulta pode ser escrita como

```
SELECT VG_AsText(VG_Intersection(rota, plantacao)), animal
FROM animal, plantacao
```

Além disso, é possível retornar todos os pontos em uma plantação afetadas por pragas. Essa consulta usa a agregação de união para capturar todas as pragas e executa a intersecção sobre cada plantação, uma vez que uma praga pode estar localizada fora de uma área de plantio, e pode ser escrita como

```
SELECT VG_AsText(VG_Intersection(p, plantacao)), descricao
FROM (SELECT VG_Union(praga) as p FROM praga), plantacao
```

Outras consultas podem utilizar predicados topológicos vagos e janelas de consultas. Tais consultas são conhecidas como *range queries*. Neste caso, uma janela de consulta é composta por um ou vários objetos no núcleo e um ou vários objetos na conjectura. Estes objetos comumente tem o formato de retângulos. Um exemplo é mostrado a seguir, que retorna todas as plantações que com certeza intersectam uma janela de consulta *QW*, com a consulta SQL

```
SELECT VG_AsText(plantacao)
FROM plantacao
WHERE VG_Intersects(plantacao, QW);
```

Além disso, operadores podem ser utilizados para restringir um tipo de retorno do relacionamento topológico. Por exemplo, retornar todas as pragas que talvez ou com certeza estejam dentro de uma janela de consulta *QW2*. Esta consulta pode ser escrita como

```
SELECT VG_AsText(plantacao)
FROM plantacao
WHERE ~VG_Intersects(plantacao, QW2);
```

A seguinte consulta SQL faz uso da junção espacial envolvendo dados espaciais vagos para retornar todos os solos com a textura média que não tem qualquer plantação, retornando áreas

em potenciais para o plantio de novas culturas. A junção espacial é uma operação frequente em banco de dados espaciais que visa combinar duas relações de acordo com algum predicado espacial. Neste caso, a junção espacial é realizada utilizando o predicado *inside* com o operador *not* na seguinte consulta SQL

```
SELECT VG_AsText(t.textura), t.descricao
FROM plantacao, textura as t
WHERE !VG_Inside(plantacao, textura) AND t.descricao = 'média'
```

Uma questão importante pode ser retornar as plantações que invadam áreas alagadas de um lago, ou seja, plantações que talvez sejam vizinhas de algum lago, por meio da seguinte consulta SQL que também faz uso da junção espacial

```
SELECT p.descricao
FROM plantacao p, lago l
WHERE ~~VG_Meets(p.plantacao, l.lago)
```

Outro exemplo de consulta com junção espacial é verificar se existe alguma rota de animal que talvez sobreponha alguma plantação, retornando os nomes dos animais e as plantações que podem estar danificadas. Esta consulta pode ser escrita como

```
SELECT animal, VG_AsText(plantacao)
FROM plantacao, animal
WHERE ~VG_Overlap(rota, plantacao)
```

A seguinte consulta retorna plantações que estejam relativamente longe de lagos e assim podem sofrer algum problema com irrigações. Esta consulta pode ser escrita como

```
SELECT p.descricao
FROM plantacao as p, lago as l
WHERE 1000 ~ VG_NearestDistance(p.plantacao, l.lago)
```

Outra possível consulta é saber os pontos em comuns onde animais possivelmente se cruzam, por meio da consulta SQL

```
SELECT VG_AsText(VG_CommonPoints(a.rota, b.rota)), a.animal, b.animal
FROM animal a, animal b
WHERE a.id <> b.id
```

4.6 Considerações Finais

Neste capítulo, o TAD VagueGeometry foi detalhado desde sua especificação interna no SGBD PostgreSQL até suas operações. Além disso, foram apresentados representações textuais e binárias para objetos espaciais vagos baseados na álgebra VASA, as quais podem ser usadas dependendo do contexto da aplicação.

Todas as operações definidas pela álgebra VASA foram implementadas pelo TAD VagueGeometry, além de operações adicionais para manipulação dos objetos espaciais vagos. Dentre essas operações adicionais estão operações para edição, criação e captura de partes de objetos espaciais vagos, como a criação de objetos espaciais vagos a partir de objetos do PostGIS. Adicionalmente, foram implementados novos operadores para manipular resultados dos predicados topológicos vagos (i.e. VagueBool) e operações numéricas (i.e. VagueNumeric). Mais especificamente, para visando um melhor desempenho no processamento dos predicados topológicos vagos foram propostas dois tipos de melhorias, que podem inclusive serem combinadas.

Como principais vantagens, o TAD VagueGeometry é de código fonte aberto e implementado sobre outros projetos de código fonte aberto, tais como o SGBD PostgreSQL e o PostGIS. Além disso, o TAD VagueGeometry proporciona a possibilidade de uso de objetos espaciais vagos para representação de fenômenos do mundo real por diferentes tipos de aplicações. Um exemplo de aplicação foi detalhado para demonstrar o uso das operações definidas por meio de consultas na linguagem SQL. Com este exemplo, foi demonstrado que o TAD VagueGeometry esconde do usuário final, a complexidade envolvida no armazenamento de objetos espaciais vagos e em suas operações. Como resultado, o usuário final pode realizar consultas espaciais envolvendo objetos espaciais vagos por meio de comandos enxutos e simples.

No próximo capítulo, Capítulo 5, o desempenho do TAD VagueGeometry é investigado por meio de uma avaliação experimental. Este teste de desempenho engloba o uso dos predicados topológicos vagos em consultas espaciais, tanto em um ambiente de banco de dados espaciais quanto em um ambiente de *data warehousing*. Além disso, as propostas de melhorias que foram descritas neste capítulo, são avaliadas experimentalmente no Capítulo 5.

Capítulo 5

AVALIAÇÃO EXPERIMENTAL DO TAD VAGUEGEOMETRY

Este capítulo investiga, por meio de uma avaliação experimental, o desempenho do TAD VagueGeometry no processamento de predicados topológicos vagos em banco de dados espaciais bem como em um ambiente de DWG.

5.1 Considerações Iniciais

Neste capítulo, o TAD VagueGeometry é validado e uma avaliação experimental é conduzida para medir o seu desempenho no processamento de consultas com predicados topológicos envolvendo dados espaciais vagos. Os testes de desempenho descritos neste capítulo, comparam o TAD VagueGeometry com a implementação dos predicados topológicos vagos no nível mais alto do SGBD PostgreSQL/PostGIS, ou seja, reutilizando as funcionalidades já existentes pela extensão PostGIS. Dessa forma, os predicados foram implementados por meio da linguagem PL/pgSQL. Para cada predicado topológico vago P, foi implementada uma função na linguagem PL/pgSQL que recebe os núcleos e conjecturas de dois objetos espaciais vagos A e B, ou seja, $P(A_n, A_c, B_n, B_c)$, onde n simboliza a parte do núcleo e c a parte da conjectura. Esta função retorna um texto que indica quando o retorno é *true*, *false* ou *maybe*, uma vez que existe a problemática dos três valores lógicos. Esta configuração é referida ao longo do capítulo como *Baseline*. O TAD VagueGeometry é comparado em relação ao *Baseline* em dois diferentes ambientes, o primeiro ambiente considera banco de dados espaciais enquanto o segundo ambiente considera *data warehouses* geográficos (DWGs).

O capítulo está organizado da seguinte forma. O primeiro ambiente de testes é descrito na Seção 5.2. Na Seção 5.3 são apresentados os resultados da primeira avaliação experimental do TAD VagueGeometry. Posteriormente, na Seção 5.4, as melhorias propostas (Seção 4.4

do Capítulo 4) para o TAD VagueGeometry são avaliadas experimentalmente. Na Seção 5.5 é descrito o segundo ambiente de testes, o qual considera DWGs e consultas SOLAP, e os resultados dos testes de desempenho realizados. Nesta avaliação, o trabalho de Siqueira (2012) também é considerado. Por fim, na Seção 5.6, as considerações finais sobre o capítulo são feitas.

5.2 Ambiente de Testes

Este primeiro ambiente de testes contempla banco de dados espaciais que armazenam objetos espaciais vagos baseados no modelo exato VASA. Os objetos espaciais vagos armazenados nestes bancos de dados foram construídos utilizando o gerador de dados proposto em Proença (2013). Proença (2013) propõe uma ferramenta que gera objetos espaciais vagos sintéticos, onde é possível determinar o volume dos dados, o formato dos objetos (i.e. retângulos ou elipses), quantidade de núcleos e conjecturas de cada objeto espacial vago e o espaço em que o núcleo e conjectura de um objeto ocupará em relação a um objeto espacial base (i.e., o núcleo e conjectura corresponde a 2% de uma região base).

Dois bancos de dados que contêm os mesmos objetos espaciais vagos foram utilizadas. Uma base de dados armazena o núcleo e conjectura dos objetos em colunas distintas, enquanto a segunda base de dados armazena os objetos espaciais vagos utilizando o TAD VagueGeometry. Foram geradas 100 mil regiões vagas distintas, onde cada região vaga é constituída por um objeto no núcleo e um objeto na conjectura nos formatos de elipses. Tais objetos foram gerados sobre a região sudeste do Brasil e correspondem a 1% de sua área. Portanto, existem diversos objetos sobrepostos.

Sobre a primeira base de dados foram processadas os predicados topológicos vagos utilizando somente recursos existentes no SGBD PostgreSQL/PostGIS, ou seja, considerando a configuração *Baseline*. Sobre a segunda base de dados foram processadas os predicados topológicos vagos utilizando o TAD VagueGeometry.

Na Tabela 5.1 são mostrados os modelos das consultas processadas sobre as bases de dados contendo objetos espaciais vagos. O tipo de consulta considerado foi a *range query*, onde o predicado topológico vago *P* é testado entre uma janela de consulta *QW* e um objeto espacial vago. A janela de consulta *QW* contém um núcleo e conjectura no formato retangular. Já o objeto espacial vago testado pelo predicado *P* é armazenado em uma coluna (TAD VagueGeometry) ou em colunas distintas (*Baseline*). Além disso, o tipo de retorno também é variável, o qual é representado pelo *RET* na Tabela 5.1. Isso significa que foram executadas consultas variando o retorno para *true*, *false* e *maybe*. No *Baseline* este tipo de retorno é filtrado por meio de uma

Tabela 5.1: Modelo de consultas executadas sobre o *Baseline* e o TAD VagueGeometry.

Configuração	Modelo de Consulta
Baseline	SELECT id FROM baseline WHERE *RET* = *P*(kernel_geo, conjecture_geo, QW_n, QW_c);
TAD VagueGeometry	SELECT id FROM vaguegeom WHERE *RET* *P*(vg, *QW*);

comparação textual, enquanto no TAD VagueGeometry é utilizado o operador correspondente (Seção 4.3.6 do Capítulo 4).

A plataforma de *software* e *hardware* usada é detalhada como segue. Foi utilizado um computador com processador Intel® Core™ i7-4770 CPU com frequência de 3.40 GHz, disco rígido SATA de 2 TB com 7.200 RPM, e 32 GB de memória principal. O sistema operacional foi o CentOS 6.5 com a versão do *kernel* 2.6.32-431.el6.x86_64, e foi instalado os seguintes programas: PostgreSQL 9.3.3, PostGIS 2.2.0 e GEOS 3.4.2.

5.3 Avaliando o TAD VagueGeometry

Esse experimento foi conduzido pelas seguintes configurações

(i) *Baseline* usou funções implementadas em PL/pgSQL no nível mais alto do SGBD PostgreSQL para processar os predicados topológicos vagos; e,

(ii) *VagueGeometry* usou o TAD VagueGeometry para processar os predicados topológicos vagos.

Cada configuração executou 100 consultas *range query* que variaram o tipo de retorno (*true*, *false* e *maybe*) para cada predicado topológico vago, de acordo com o modelo de consulta da Tabela 5.1. As 100 janelas de consulta usadas foram geradas pela ferramenta proposta em Proença (2013), no formato de retângulos composta por um núcleo e conjectura, onde cada janela correspondeu a 10% da área da região Sudeste do Brasil. Cada janela de consulta foi executada 10 vezes, e posteriormente, a média da soma da execução das 100 janelas de consulta foi calculada para cada predicado e tipo de retorno. Ou seja, para cada predicado topológico vago e tipo de retorno, foi calculado o tempo médio de execução de 100 janelas de consulta. Os predicados considerados foram *disjoint*, *overlap*, *inside*, *intersects*, *coveredBy*, *meets* e *equals*. Os testes

foram realizados localmente para inibir a latência da rede. O cache do sistema operacional e do SGBD foi limpo depois da execução de cada janela de consulta. Nas Figuras 5.1 a 5.3 são mostrados os resultados obtidos.

Claramente, o desempenho do TAD VagueGeometry superou o *Baseline*, o que indica que as estruturas internas e a manipulação dos objetos espaciais vagos em baixo nível é mais eficiente que a definição dos predicados topológicos vagos em uma linguagem de alto nível. O tempo de execução de cada predicado topológico vago não teve variação significativa para cada tipo de retorno. Ou seja, o tempo de execução médio das janelas de consulta não apresentaram mudança substancial entre os retornos *true*, *false* e *maybe*. Além disso, constatou-se que o predicado que exigiu mais tempo de processamento foi o *overlap*, sendo o mais custoso em ambas as configurações. A configuração *Baseline* apresentou tempos de execuções proibitivos.

O TAD VagueGeometry foi mais eficiente que o *Baseline*, apresentando reduções de 42,36% (predicado *meets* com retorno *false*) a 65,94% (predicado *equals* com retorno *maybe*). A redução de tempo é a porcentagem que determina o quanto mais eficiente uma configuração foi sobre uma outra configuração. Em cada figura (5.1 a 5.3), a redução mínima e a redução máxima sobre o *Baseline* são destacadas. Apesar das expressivas reduções apresentadas pelo TAD VagueGeometry, foram investigadas possíveis melhorias. Na próxima seção, a Seção 5.4, são analisados o tempo de processamento dos predicados topológicos vagos utilizando as melhorias propostas na Seção 4.4 do Capítulo 4, as quais visam reduzir ainda mais o tempo de resposta no processamento desses predicados.

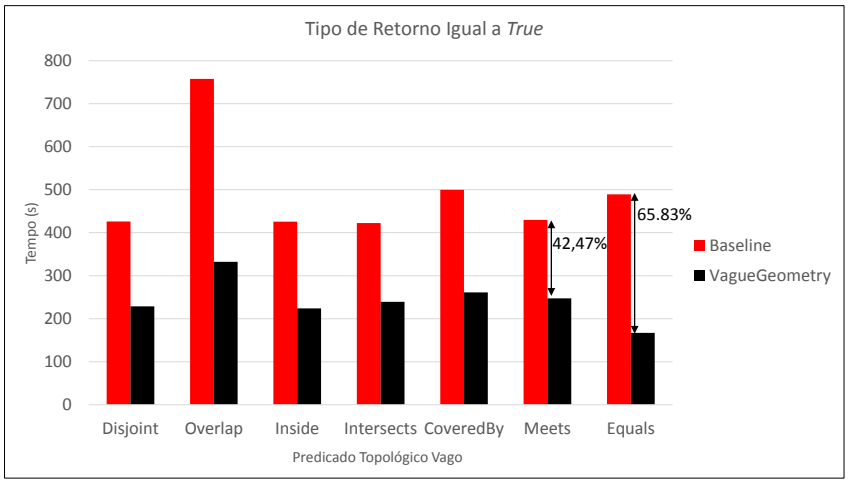


Figura 5.1: Tempo de execução médio para cada predicado topológico vago considerando como tipo de retorno o valor lógico *true*.

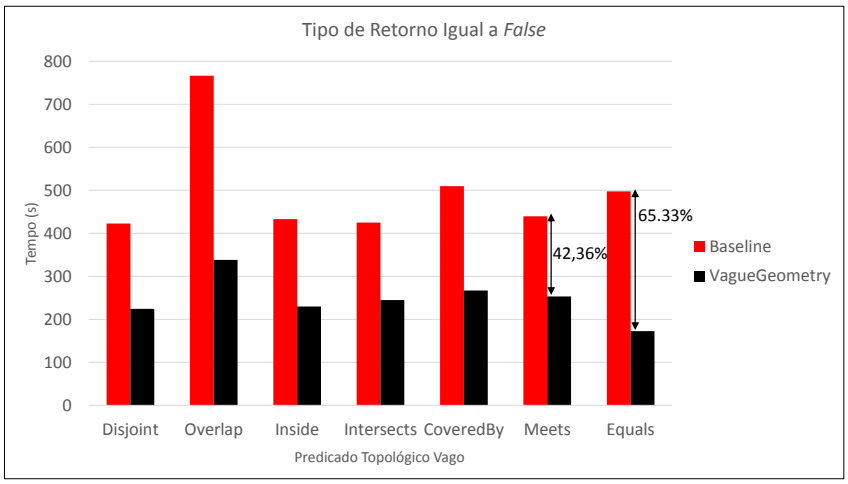


Figura 5.2: Tempo de execução médio para cada predicado topológico vago considerando como tipo de retorno o valor lógico *false*.

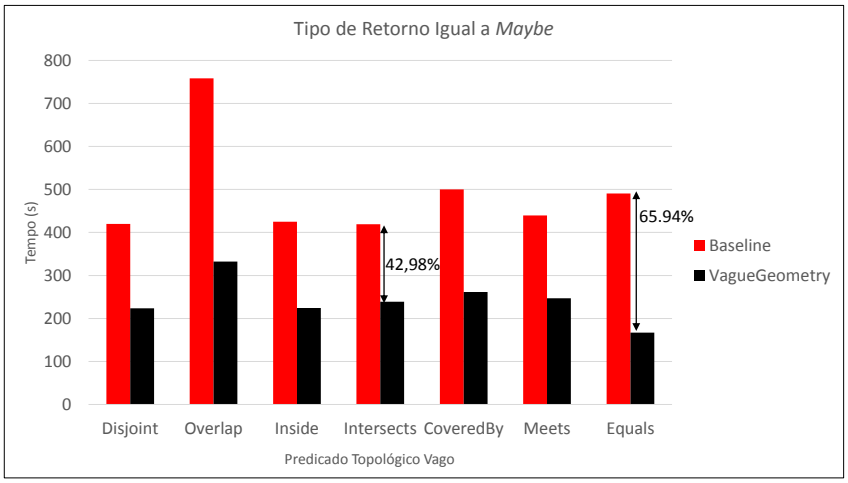


Figura 5.3: Tempo de execução médio para cada predicado topológico vago considerando como tipo de retorno o valor lógico *maybe*.

A construção dos objetos espaciais vagos por meio do TAD VagueGeometry demorou 2,42 segundos. Esta construção utilizou a função *VG_MakeVagueGeom*, para unir e armazenar em uma única estrutura, o núcleo e a conjectura que antes eram armazenados em colunas distintas pela configuração *Baseline*. O requisito de espaço de armazenamento dos objetos espaciais vagos é um fator que também foi avaliado. Quanto maior a complexidade de uma geometria, mais espaço de armazenamento é necessário. O formato geométrico do núcleo e conjectura dos objetos espaciais vagos é a elipse, que em média possuía 33 pontos. Dessa forma, o espaço de armazenamento requerido pelo *Baseline* foi de 108,33 MB, enquanto que para o TAD VagueGeometry foi de 106,43 MB. O espaço de armazenamento requerido por ambas as configurações foi relativamente igual, uma vez que o TAD VagueGeometry reduziu somente 1,76% (i.e., 1,9 MB) em relação ao *Baseline*.

5.4 Comparação das Melhorias do TAD VagueGeometry

Esse experimento utilizou a plataforma de *software* e *hardware* apresentadas na Seção 5.2 e teve como objetivo avaliar as melhorias do TAD VagueGeometry, apresentadas na Seção 4.4 do Capítulo 4, e foi conduzido pelas seguintes configurações

(i) *VagueGeometry* usou o TAD VagueGeometry para processar os predicados topológicos vagos e apresenta os mesmos resultados da Seção 5.3;

(ii) *VagueGeometry + União* usou o TAD VagueGeometry pré-armazenando a união entre o núcleo e a conjectura de cada objeto espacial vago para processar os predicados topológicos vagos;

(iii) *VagueGeometry + MBRs* usou o TAD VagueGeometry e MBRs para processar os predicados topológicos vagos; e,

(iv) *VagueGeometry + União e MBRs* usou o TAD VagueGeometry pré-armazenando a união entre o núcleo e a conjectura de cada objeto espacial vago, além de MBRs, para processar os predicados topológicos vagos.

Cada configuração executou 100 consultas *range query* que variaram o tipo de retorno (*true*, *false* e *maybe*) para cada predicado topológico vago, de acordo com o modelo de consulta da Tabela 5.1. As 100 janelas de consulta usadas foram as mesmas dos testes da Seção 5.3. Cada janela de consulta foi executada 10 vezes, e posteriormente, para cada predicado topológico vago e tipo de retorno, foi calculado o tempo médio de execução das 100 janelas de consulta. Os predicados considerados foram *disjoint*, *overlap*, *inside*, *intersects*, *coveredBy*, *meets* e *equals*. Os testes foram realizados localmente para inibir a latência da rede. O cache do sistema operacional e do SGBD foi limpo depois da execução de cada janela de consulta. Nas Figuras 5.4 a 5.6 são mostrados os resultados obtidos.

O simples pré-armazenamento da união entre o núcleo e a conjectura de cada objeto espacial vago, não configurou a melhora no desempenho do processamento dos predicados topológicos vagos. Isso ocorreu devido ao tipo de consulta *range query*, uma vez que um objeto é fixo (i.e. a janela de consulta) e o outro objeto é variável (i.e. o objeto VagueGeometry presente na tabela). Com isso, é necessário a recuperação constante dos objetos VagueGeometry da tabela relacional. Por estes objetos pré-armazenarem a união, é necessário um maior tempo de recuperação. Além disso, por estes objetos serem maiores, a política interna do PostgreSQL efetuou compressões nestes objetos. Com esta compressão, há a necessidade do PostgreSQL

efetuar a descompressão no ato de sua recuperação. Já o uso de MBRs (configuração *VagueGeometry + MBRs*) proporcionou reduções de 36,70% (predicado *equals* com retorno *maybe*) a 68,53% (predicado *overlap* com retorno *false*) em relação a proposta inicial do TAD VagueGeometry.

Por outro lado, a configuração que apresentou melhor desempenho foi a que combina as duas melhorias, a configuração *VagueGeometry + União e MBRs*, apresentando reduções de 57,24% (predicado *equals* com retorno *maybe*) a 77,95% (predicado *overlap* com retorno *maybe*) em relação a proposta inicial do TAD VagueGeometry, ou seja, a configuração *VagueGeometry*. Isso se deve ao fato de que o retorno prévio de resultados de um predicado topológico vago por meio do uso dos MBRs, fez com que menos objetos VagueGeometry fossem recuperados e assim descomprimidos. Em cada figura (5.4 a 5.6), a redução mínima e a redução máxima sobre o *VagueGeometry*, para cada tipo de retorno, são destacados. Como resultado, nota-se que as melhorias propostas foram essenciais para melhorar ainda mais o desempenho do TAD VagueGeometry.

A construção dos objetos espaciais vagos na configuração *VagueGeometry + União* foi de 4,29 segundos, já para a configuração *VagueGeometry + MBRs* foi de 2,43 segundos e para a configuração *VagueGeometry + União e MBRs* foi de 4,38 segundos. As configurações *VagueGeometry + União* e *VagueGeometry + União e MBRs* utilizaram a função *VG_MakeVagueGeom* com o parâmetro para realizar a união entre o núcleo e a conjectura para a construção dos objetos espaciais vagos. A construção dos objetos da configuração *VagueGeometry + União e MBRs* foi 80,87% mais lenta que a construção dos objetos espaciais vagos da configuração *VagueGeometry*. Porém, o tempo adicional requerido na construção é compensado pelo processamento eficiente de consultas que envolvem predicados topológicos vagos.

O espaço de armazenamento requerido pela configuração *VagueGeometry + União* foi de 71,79 MB, para a configuração *VagueGeometry + MBRs* foi de 109,48 MB e para a configuração *VagueGeometry + União e MBRs* foi de 76,37 MB. É importante enfatizar a compressão automática realizada pelo SGBD PostgreSQL nas configurações *VagueGeometry + União* e *VagueGeometry + União e MBRs*. Dessa forma, além do ganho razoável de desempenho no processamento das consultas, a configuração *VagueGeometry + União e MBRs* reduziu o espaço de armazenamento no mínimo 28,25% (em relação a configuração *VagueGeometry*) e no máximo 30,25% (em relação a configuração *VagueGeometry + MBRs*), devido a essa compressão.

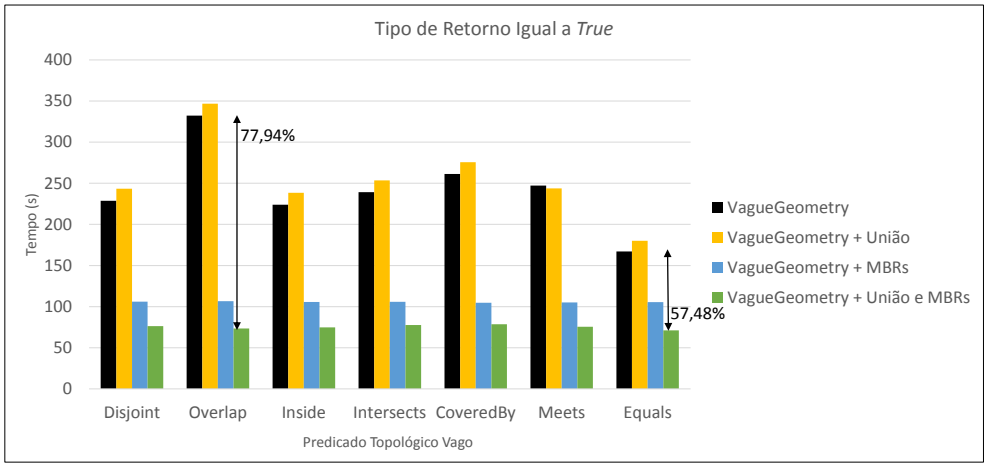


Figura 5.4: Comparação das melhorias propostas para o TAD VagueGeometry, apresentando o tempo de execução médio para cada predicado topológico vago considerando como tipo de retorno o valor lógico *true*.

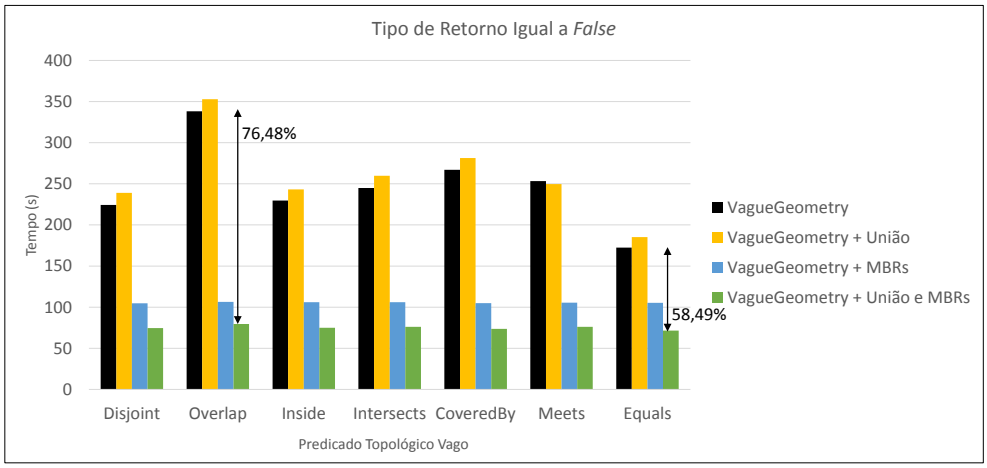


Figura 5.5: Comparação das melhorias propostas para o TAD VagueGeometry, apresentando o tempo de execução médio para cada predicado topológico vago considerando como tipo de retorno o valor lógico *false*.

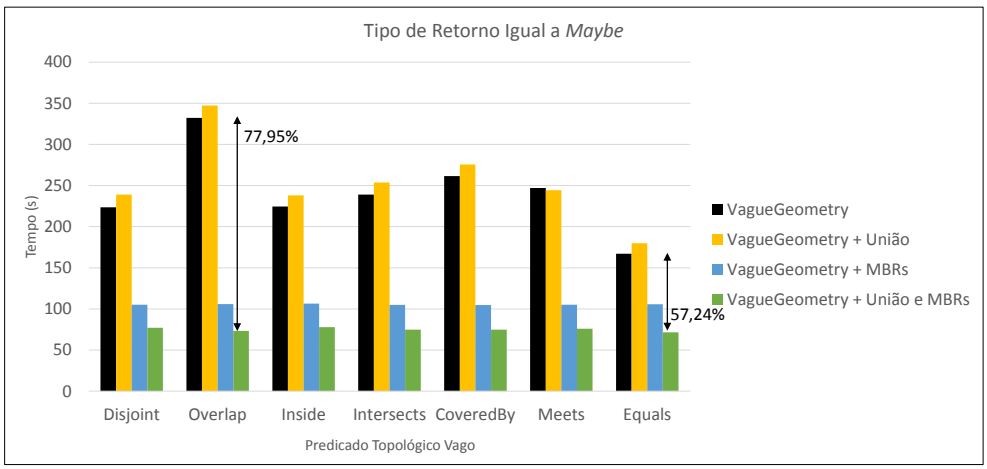


Figura 5.6: Comparação das melhorias propostas para o TAD VagueGeometry, apresentando o tempo de execução médio para cada predicado topológico vago considerando como tipo de retorno o valor lógico *maybe*.

5.5 Análise Experimental em Ambientes de Data Warehousing

A segunda avaliação experimental do TAD VagueGeometry foi conduzida em um ambiente de *data warehousing* que armazena objetos espaciais vagos. Nas seguintes subseções é realizada uma investigação de como incorporar de forma eficiente dados espaciais vagos em data warehouses geográficos. Na Seção 5.5.1 é detalhado o ambiente onde os testes foram realizados, enquanto na Seção 5.5.2 são discutidos os resultados obtidos.

5.5.1 Ambiente de Testes

Este segundo ambiente de testes contempla *data warehouses* geográficos (DWGs) que armazenam objetos espaciais vagos baseados no modelo exato VASA. O ambiente de DWG considerado é mostrado na Figura 5.7, o qual é um esquema adaptado da Figura 3.3 do Capítulo 3 (SIQUEIRA, 2012). O *Star Schema Benchmark* (SSB) (O'NEIL, 2009) com fator de escala 1, foi utilizado para carregar as tabelas de dimensão *Pesticide* (que corresponde a tabela *Part* do SSB), *Date* e *Supplier*, além de produzir 6 milhões de tuplas na tabela de fatos *LineOrder*. Os objetos espaciais vagos foram armazenados na dimensão *AppliedArea*. Nesta dimensão foram armazenadas 10 mil regiões vagas, as quais correspondem a 10% do banco de dados do primeiro ambiente de testes (Seção 5.2). Dessa forma, cada região vaga é constituída por um objeto no núcleo e um objeto na conjectura nos formatos de elipses, correspondendo a 1% da área da região sudeste do Brasil.

Para cada configuração de testes foi criado um DWG específico, que varia a forma com que os objetos espaciais vagos são armazenados na dimensão *AppliedArea*. Na Figura 5.7 é mostrada a forma de armazenamento utilizada pelo *Baseline* do primeiro ambiente de testes, que consiste em armazenar o núcleo e a conjectura dos objetos espaciais vagos em colunas distintas. Outra forma de armazenamento considerado foi o proposto em Siqueira (2012), o qual armazena o núcleo e a conjectura em tabelas distintas (*AppliedAreaCore* para o núcleo e *AppliedAreaDubiety* para a conjectura). Na Figura 5.8 é mostrado este esquema, onde as funções utilizadas pelo *Baseline* são executadas, uma vez que o núcleo e a conjectura são armazenados separadamente. Por fim, na Figura 5.9 é mostrado a forma de armazenamento utilizado pelo TAD VagueGeometry, onde o atributo *appliedarea_geo* é do tipo VAGUEPOLYGON. Pode-se notar que o TAD VagueGeometry oferece um esquema mais enxuto e claro de armazenamento de dados espaciais vagos em dimensões de um DWG, quando comparado aos esquemas utilizados pelo *Baseline* e o proposto em Siqueira (2012).

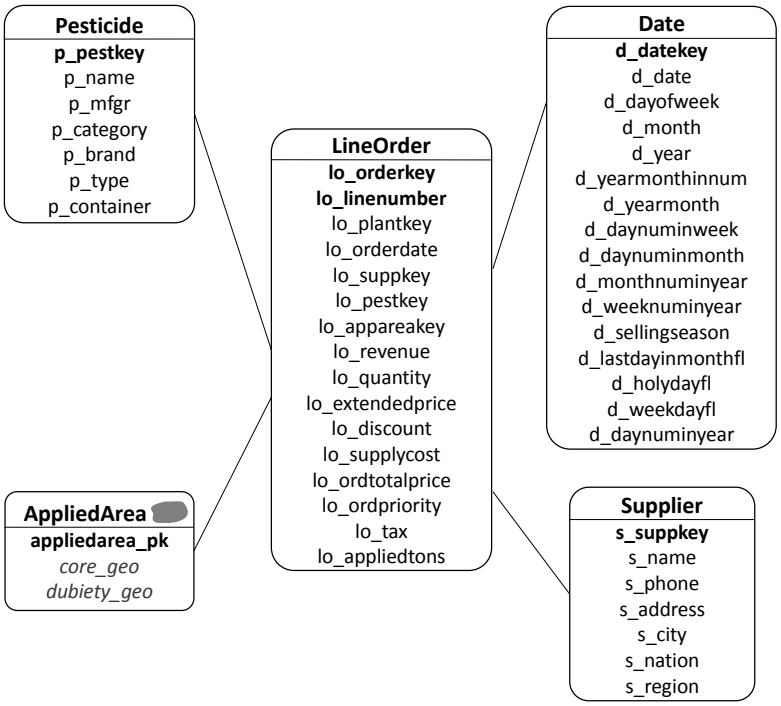


Figura 5.7: Esquema de DWG que mantém dados espaciais vagos na dimensão *AppliedArea* armazenando o núcleo e conjectura em colunas distintas, tal como usado pela configuração *Baseline*.

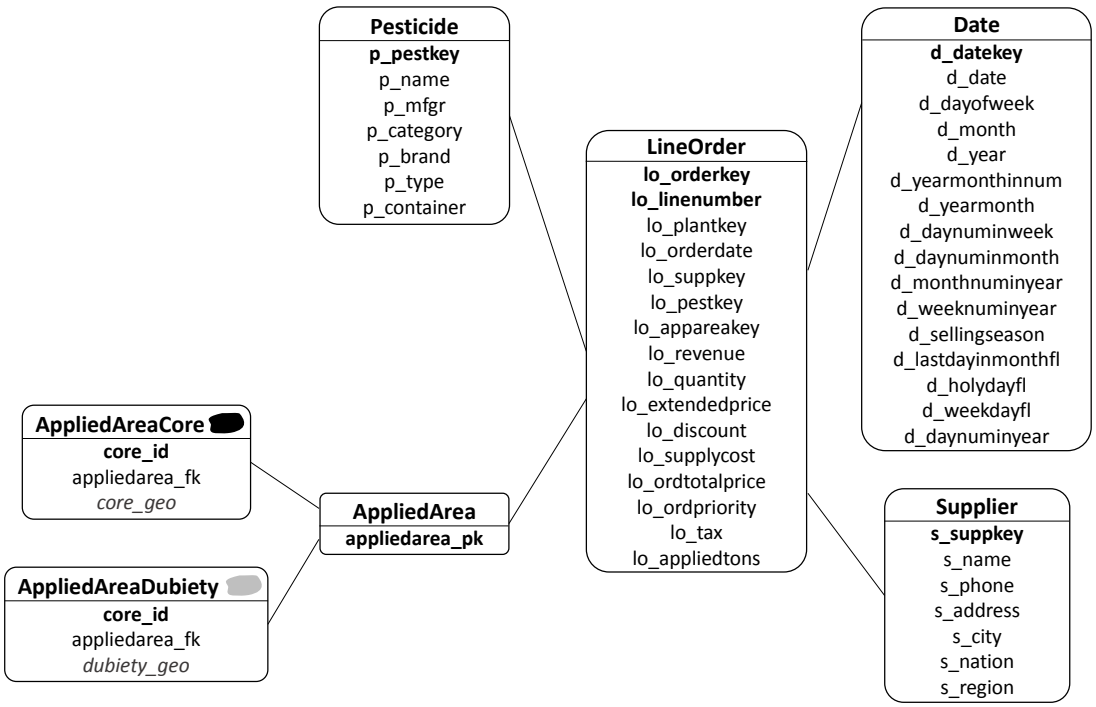


Figura 5.8: Esquema de DWG que mantém dados espaciais vagos na dimensão *AppliedArea* armazenando o núcleo e conjectura em tabelas distintas, conforme proposto em Siqueira (2012).

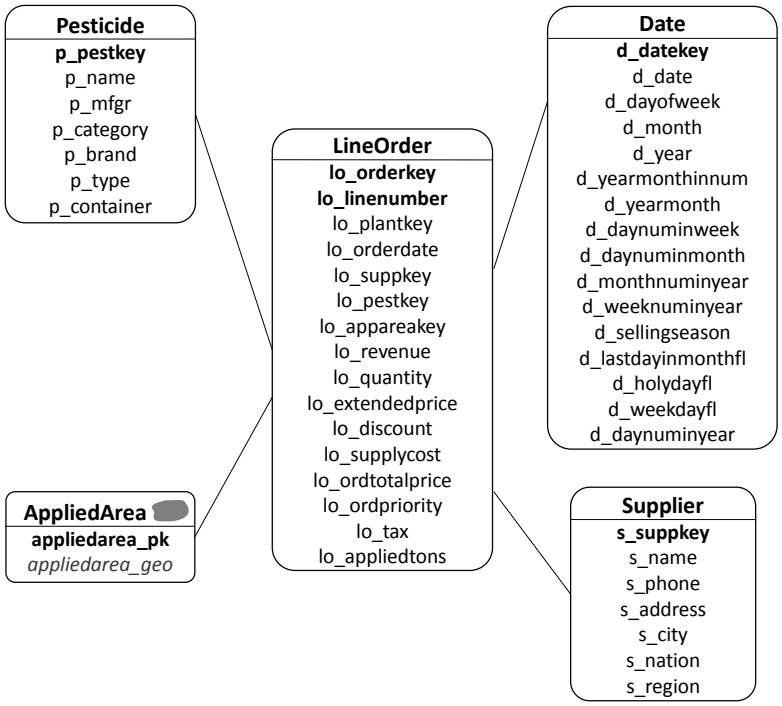


Figura 5.9: Esquema de DWG que mantém dados espaciais vagos na dimensão *AppliedArea* por meio do TAD VagueGeometry.

A seguir é mostrado o modelo de consulta SOLAP processada sobre os DWGs contendo objetos espaciais vagos. Esta consulta é a mesma definida em Siqueira (2012), e realiza diversas junções, ordenações, agregações e operações de *slice-and-dice*. Janelas de consultas *QW*, as quais são compostas por um núcleo e uma conjectura nos formatos retangulares, são processadas sobre os objetos espaciais vagos armazenados na dimensão *AppliedArea* considerando um predicado espacial vago *P*. O tipo de retorno do predicado, representado pelo *RET*, também é considerado. Isso significa que foram executadas consultas variando o retorno para *true*, *false* e *maybe*. No DWG baseado no *Baseline* (Figura 5.7) e no DWG que utiliza a modelagem proposta em Siqueira (2012) (Figura 5.8) este tipo de retorno é filtrado por meio de uma comparação textual, enquanto no DWG que usa TAD VagueGeometry (Figura 5.9) é utilizado o operador correspondente (Seção 4.3.6 do Capítulo 4).

Na Tabela 5.2 é mostrado, para cada esquema de DWG, o *GAP 1* e *GAP 2* presente no modelo de consulta. Estes *GAPs* podem conter novas junções e variar o tipo de chamada do predicado topológico vago. É importante notar que o TAD VagueGeometry simplifica a consulta SOLAP, tornando-a mais clara e enxuta, uma vez que não introduz junções e nem um número alto de argumentos na chamada do predicado topológico vago.

O modelo de consulta SOLAP usada é escrita como

```
SELECT d_year, s_nation, SUM(lo_revenue - lo_supplycost) AS profit,
  SUM(lo_appliedtons) AS pesticide_tons
FROM Date, Supplier, Pesticide, AppliedArea, LineOrder
  GAP 1
WHERE lo_orderdate = d_datekey AND lo_suppkey = s_suppkey
  AND lo_pestkey = p_pestkey AND lo_appareakey = appliedarea_pk
  GAP 2
  AND s_region = 'AMERICA' AND (p_mfgr = 'MFGR#1' OR p_mfgr = 'MFGR#2')
GROUP BY d_year, s_nation
ORDER BY d_year, s_nation;
```

Tabela 5.2: Preenchendo os *GAPs* do modelo de consulta SOLAP, conforme cada configuração de DWG.

Esquema de DWG	*GAP 1*	*GAP 2*
Armazenamento do núcleo e conjectura em colunas distintas (Figura 5.7)		*RET* = *P*(core_geo, dubiety_geo, QW_n, QW_c)
Armazenamento do núcleo e conjectura em tabelas distintas (Figura 5.8)	AppliedAreaCore as c, AppliedAreaDubiety as d	AND appliedarea_pk = c.appliedarea_fk AND appliedarea_pk = d.appliedarea_fk AND *RET* = *P*(core_geo, dubiety_geo, QW_n, QW_c)
TAD VagueGeometry (Figura 5.9)		*RET* *P*(appliedarea_geo, QW)

5.5.2 Avaliação do TAD VagueGeometry em Ambiente de Data Warehouse Geográfico

Esse experimento utilizou a plataforma de *software* e *hardware* descrita na Seção 5.2 e teve como objetivo avaliar o desempenho do TAD VagueGeometry no processamento de consultas SOLAP e foi conduzido pelas seguintes configurações

(i) *Baseline* usou funções implementadas em PL/pgSQL no nível mais alto do SGBD PostgreSQL para processar os predicados topológicos vagos da consulta SOLAP sobre o esquema de DWG da Figura 5.7 que armazenava os objetos espaciais vagos em colunas distintas;

(ii) *Tabelas Separadas* usou funções implementadas em PL/pgSQL no nível mais alto do SGBD PostgreSQL para processar os predicados topológicos vagos da consulta SOLAP sobre o esquema de DWG da Figura 5.8 que armazenava os objetos espaciais vagos em tabelas distintas;

(iii) *VagueGeometry + MBRs* usou o TAD VagueGeometry e MBRs para processar os predicados topológicos vagos da consulta SOLAP sobre o esquema de DWG da Figura 5.9; e,

(iv) *VagueGeometry + União e MBRs* usou o TAD VagueGeometry pré-armazenando a união entre o núcleo e conjectura de cada objeto espacial vago, além de MBRs, para processar os predicados topológicos vagos da consulta SOLAP sobre o esquema de DWG da Figura 5.9.

Cada configuração executou 100 consultas SOLAP que variaram o tipo de retorno (*true*, *false* e *maybe*) de cada predicado topológico vago, de acordo com o modelo de consulta da Tabela 5.2. As 100 janelas de consulta usadas foram as mesmas utilizadas nas Seções 5.3 e 5.4. Cada janela de consulta foi executada 10 vezes, e posteriormente, para cada predicado topológico vago e tipo de retorno, foi calculado o tempo médio de execução das 100 janelas de consulta. Além disso, foi capturado o tempo médio de processamento somente da parte espacial, ou seja, do predicado topológico da consulta SOLAP. Os predicados considerados foram *disjoint*, *overlap* e *inside*. Os testes foram realizados localmente para inibir a latência da rede. O cache do sistema operacional e do SGBD foi limpo depois da execução de cada janela de consulta. Nas Figuras 5.10 a 5.12 são mostrados os resultados obtidos. Nessas figuras, a redução mínima e a redução máxima sobre o *Baseline*, para cada tipo de retorno, são também destacados.

O processamento do predicado topológico vago correspondeu de 0,12% (configuração *VagueGeometry + MBRs*) a 3,86% (configuração *Baseline*) da consulta SOLAP. Isso indica que os fatores que mais influenciaram no processamento foram os filtros, junções, agregações e ordenações, as quais são comumente realizadas em consultas SOLAP. Mesmo com esta correspondência, a configuração *VagueGeometry + MBRs* se sobressaiu em relação as outras configurações. De fato, as configurações *Baseline* e *Tabelas Separadas* apresentaram os piores

desempenhos para processar a parte espacial da consulta SOLAP. Em especial, a configuração *Tabelas Separadas* obteve um pior desempenho no processamento da consulta SOLAP completa, por realizar junções adicionais. Ademais, a configuração *VagueGeometry + União e MBRs* demandou mais tempo que a configuração *VagueGeometry + MBRs* para processar a consulta SOLAP em algumas situações, devido ao volume de dados dos objetos espaciais vagos armazenados na dimensão *AppliedArea*. Dessa forma, o armazenamento da união influenciou para a perda de eficiência no processamento dos predicados, quando um volume menor de objetos espaciais vagos foi considerado. Apesar disso, esta configuração ainda se sobressaiu em relação as configurações *Baseline* e *Tabelas Separadas*. Os melhores ganhos foram da configuração *VagueGeometry + MBRs* com reduções de 0,27% a 4,98% no processamento completo da consulta SOLAP, quando comparado ao *Baseline*. É importante destacar que os resultados foram diferentes dos obtidos para bancos de dados espaciais na Seção 5.3 e que por isso as características intrínsecas de um ambiente de DWG devem ser investigados para analisar o desempenho do processamento de predicados topológicos com dados espaciais vagos.

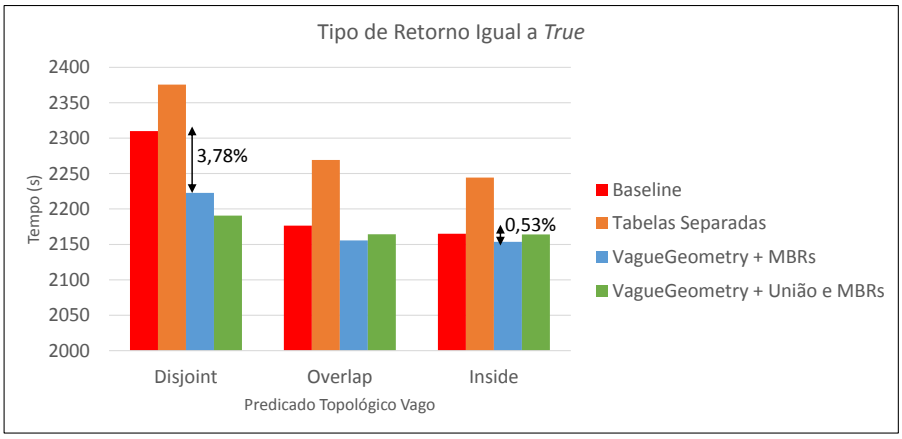


Figura 5.10: Tempo de execução médio, para cada predicado topológico vago, de consultas SOLAP considerando como tipo de retorno o valor lógico *true*.

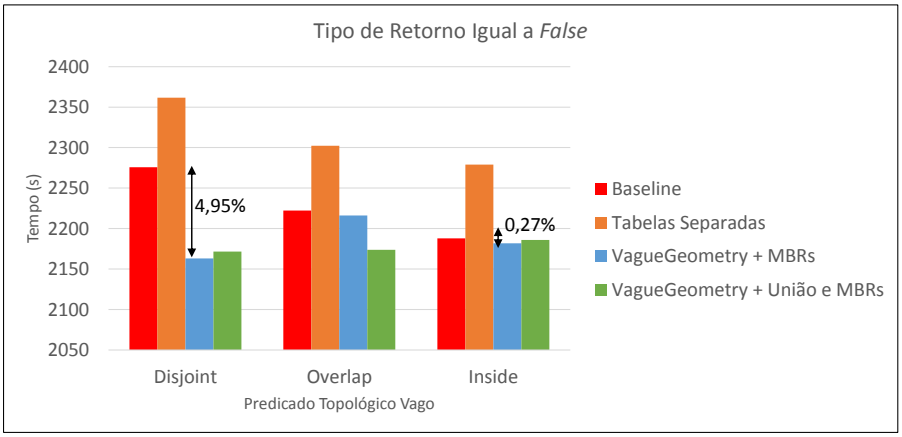


Figura 5.11: Tempo de execução médio, para cada predicado topológico vago, de consultas SOLAP considerando como tipo de retorno o valor lógico *false*.

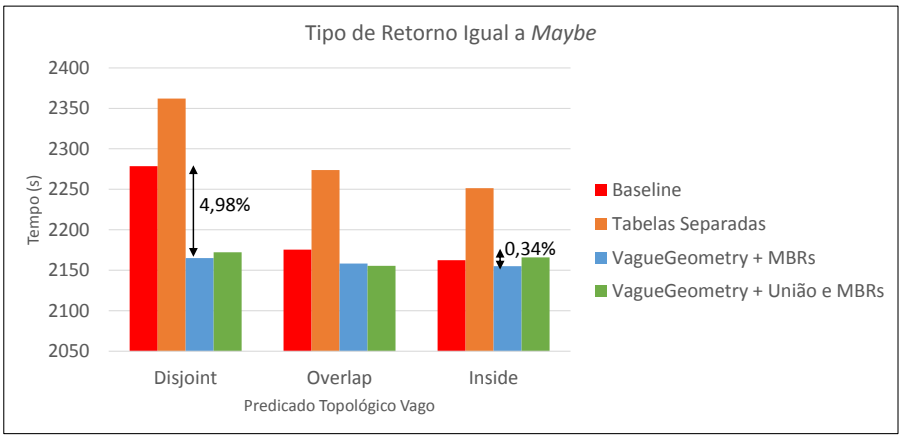


Figura 5.12: Tempo de execução médio, para cada predicado topológico vago, de consultas SOLAP considerando como tipo de retorno o valor lógico *maybe*.

Outro fator importante que colaborou na diferença dos resultados obtidos para banco de dados espaciais na Seção 5.3 foi o otimizador de consultas do SGBD PostgreSQL, o qual determinou em que momento o predicado topológico vago seria executado. Para as configurações *Baseline* e *Tabelas Separadas*, o SGBD PostgreSQL determinou o mesmo plano de consulta. Este plano de consulta executava os predicados topológicos vagos no mesmo nível de complexidade que os filtros, para então realizar as outras operações, tais como junções, agregações e ordenações. Porém, para as configurações do *VagueGeometry + MBRs* e *VagueGeometry + União e MBRs*, os planos de consultas foram diferentes, uma vez que os tamanhos dos objetos são diferentes. Assim, o otimizador de consultas, foi outro fator que fez com que a configuração *VagueGeometry + União e MBRs* demandasse mais tempo de processamento que a configuração *VagueGeometry + MBRs*, uma vez que contribuiu no número de objetos a serem processados pelo predicados topológico vago. Além do tamanho dos objetos, outro fator considerado pelo otimizador de consultas são as estatísticas coletadas referente aos objetos armazenados. Enquanto o PostGIS fornece estatísticas sobre os seus objetos visando sempre melhorar os planos de consulta, o mesmo não é feito para o TAD VagueGeometry. Devido a isso, o SGBD PostgreSQL formulou diferentes planos de consultas, afetando diretamente no desempenho no processamento dos predicados topológicos vagos. Considerando somente este processamento, a configuração *VagueGeometry + MBRs* apresentou reduções de 92,46% a 95,20% em relação ao *Baseline*. Nas Figuras 5.13 a 5.15 são mostrados os resultados obtidos no processamento somente do predicado topológico vago nas consultas SOLAP. Ainda nestas figuras, a redução mínima e a redução máxima sobre o *Baseline*, para cada tipo de retorno, são destacados.

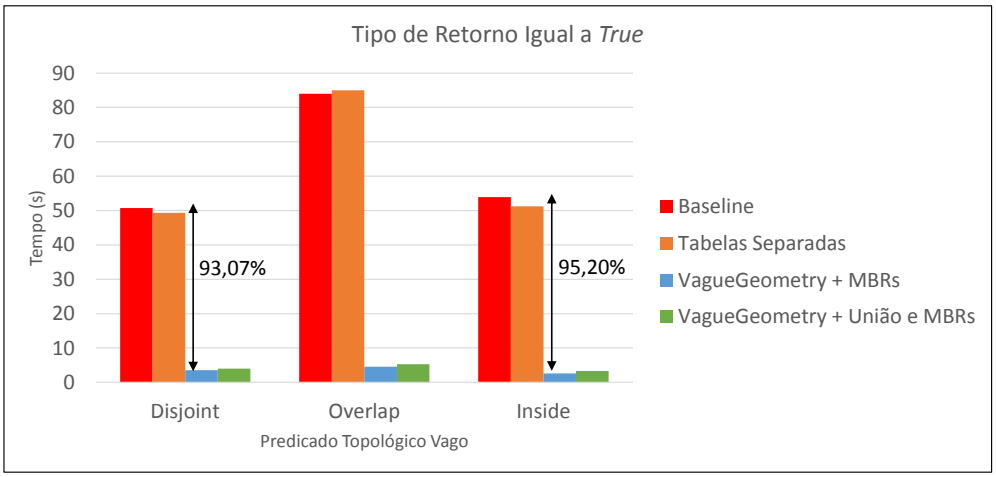


Figura 5.13: Tempo de execução médio somente considerando o processamento de cada predicado topológico vago em consultas SOLAP com o retorno lógico igual a *true*.

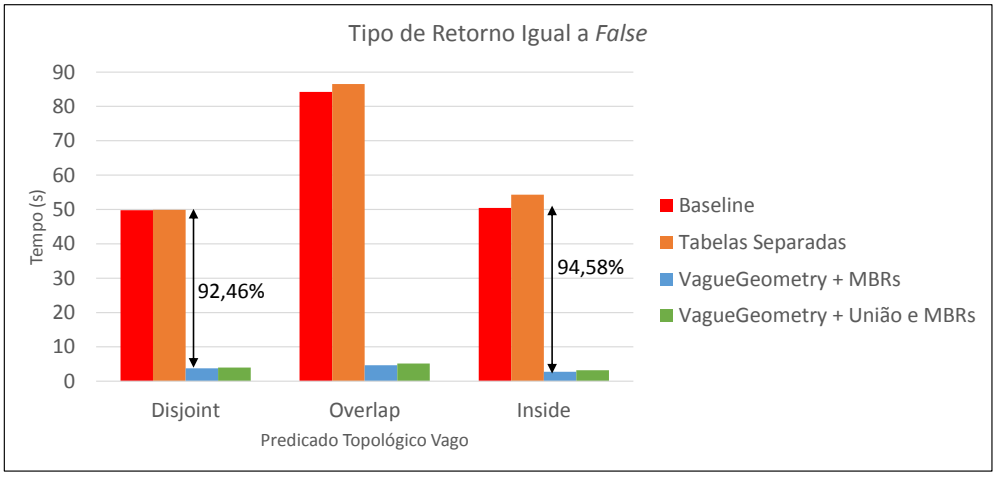


Figura 5.14: Tempo de execução médio somente considerando o processamento de cada predicado topológico vago em consultas SOLAP com o retorno lógico igual a *false*.

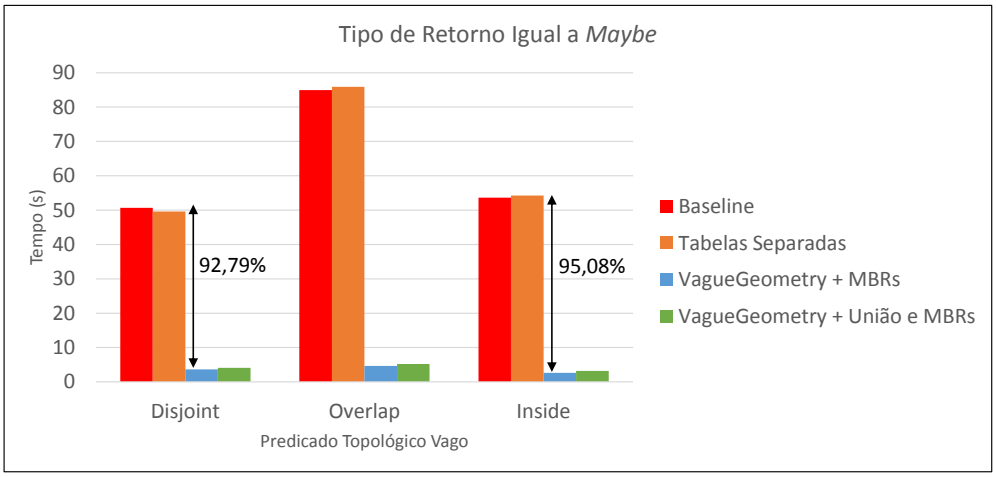


Figura 5.15: Tempo de execução médio somente considerando o processamento de cada predicado topológico vago em consultas SOLAP com o retorno lógico igual a *maybe*.

A construção dos objetos espaciais vagos da configuração *VagueGeometry + MBRs* demorou 0,24 segundos, enquanto a configuração *VagueGeometry + União e MBRs* demorou 0,43 segundos. O espaço de armazenamento requerido pelas configurações *Baseline* e *Tabelas Separadas* foi de 10,83 MB, para a configuração *VagueGeometry + MBRs* foi de 10,94 MB, e para a configuração *VagueGeometry + União e MBRs* foi de 7,63 MB. Neste caso, a configuração *VagueGeometry + União e MBRs* requereu menos espaço de armazenamento que as outras configurações, devido a compressão dos dados.

5.6 Considerações Finais

Neste capítulo, o TAD VagueGeometry foi avaliado experimentalmente para averiguar o desempenho do TAD VagueGeometry no processamento de consultas que continham predicados topológicos vagos. Primeiramente, a proposta inicial do TAD VagueGeometry, a qual foi detalhada no Capítulo 4, foi avaliada em um ambiente de banco de dados espaciais e comparada com o processamento dos predicados topológicos vagos no nível mais alto do SGBD PostgreSQL. Esta configuração foi chamada de *Baseline*, e utiliza somente as funcionalidades existentes do SGBD PostgreSQL/PostGIS para armazenar e processar os predicados. Dessa forma, o núcleo e conjectura na configuração *Baseline* são armazenados em colunas distintas. O resumo dos resultados desta primeira comparação é mostrada na Tabela 5.3, a qual exibe as reduções mínimas e máximas obtidas para cada tipo de retorno. O TAD VagueGeometry apresentou ganhos de 42,36% a 65,94% se comparado ao *Baseline*.

Posteriormente, foi medido o desempenho das melhorias propostas para o TAD VagueGeometry para diminuir ainda mais o tempo de processamento de consultas envolvendo predicados topológicos vagos. A primeira melhoria proposta armazena a união do núcleo com a conjectura de um objeto espacial vago. Enquanto a segunda melhoria proposta utiliza MBRs. A primeira melhoria não apresentou reduções significativas, devido a compressão realizada pelo SGBD PostgreSQL na armazenação de objetos VagueGeometry, uma vez que o tamanho desses objetos foram aumentados. Por outro lado, a combinação desta melhoria com o uso de MBRs

Tabela 5.3: Reduções mínimas e máximas obtidas pelo TAD VagueGeometry em relação ao *Baseline*, para cada tipo de retorno.

Tipo de Retorno	Redução Mínima	Redução Máxima
True	42,47%	65,83%
False	42,36%	65,33%
Maybe	42,98%	65,94%

Tabela 5.4: Reduções mínimas e máximas obtidas pela combinação das melhorias do TAD VagueGeometry em relação a sua proposta inicial, para cada tipo de retorno.

Tipo de Retorno	Redução Mínima	Redução Máxima
True	57,48%	77,94%
False	58,49%	76,48%
Maybe	57,24%	77,95%

Tabela 5.5: Reduções mínimas e máximas obtidas pela combinação das melhorias do TAD VagueGeometry em relação ao *Baseline*, o qual usa funcionalidades existentes em banco de dados espaciais.

Tipo de Retorno	Redução Mínima	Redução Máxima
True	81,63%	90,32%
False	82,11%	89,62%
Maybe	81,64%	90,34%

proporcionou ganhos de 57,24% a 77,95% se comparado ao TAD VagueGeometry proposto inicialmente. O resumo desses resultados é mostrado na Tabela 5.4, a qual exibe as reduções mínimas e máximas obtidas por cada tipo de retorno. É importante enfatizar que estas reduções foram em relação a proposta inicial do TAD VagueGeometry, logo, se o uso combinado das melhorias for comparado ao *Baseline*, foram obtidas reduções de 81,63% a 90,34%. O resumo dos ganhos da combinação das melhorias propostas do TAD VagueGeometry em relação ao *Baseline* é mostrado na Tabela 5.5.

O TAD VagueGeometry também foi avaliado em um ambiente de DWG que continha dados espaciais vagos. Neste ambiente, foi considerado o processamento de consultas SOLAP que continham predicados topológicos vagos. Esta avaliação experimental mostrou que o otimizador de consultas do SGBD PostgreSQL foi um fator importante, uma vez que determinou o momento em que o predicado topológico vago fosse executado. O otimizador de consultas também contou com a coleta de estatísticas do PostGIS para melhorar o plano de consultas nas configurações que reutilizavam as funcionalidades existentes do SGBD PostgreSQL/PostGIS. Além disso, demonstrou também que o predicado topológico vago correspondia em apenas de 0,12% a 3,86% do tempo total de processamento completo da consulta SOLAP. Isso indicou que os filtros, junções, agregações e ordenações requereram o maior tempo do processamento. Apesar desses fatores, o TAD VagueGeometry, que usava MBRs, apresentou reduções de 0,27% a 4,98% em relação ao *Baseline* no processamento completo das consultas SOLAP. Apesar da combinação das melhorias também ter apresentado reduções em relação ao *Baseline*, ela não foi a melhor configuração, devido ao volume de dados, o otimizador de consultas do SGBD PostgreSQL e a descompressão dos objetos no ato de sua recuperação. Na Tabela 5.6 são re-

Tabela 5.6: Reduções mínimas e máximas obtidas pelo TAD VagueGeometry com uso de MBRs em relação ao *Baseline* no processamento de consultas SOLAP, para cada tipo de retorno do predicado topológico vago.

Tipo de Retorno	Redução Mínima	Redução Máxima
True	0,53%	3,78%
False	0,27%	4,95%
Maybe	0,34%	4,98%

Tabela 5.7: Reduções mínimas e máximas obtidas pelo TAD VagueGeometry com uso de MBRs em relação ao *Baseline* no processamento dos predicados topológicos vagos das consultas SOLAP.

Tipo de Retorno	Redução Mínima	Redução Máxima
True	93,07%	95,20%
False	92,46%	94,58%
Maybe	92,79%	95,08%

sumidos os ganhos mínimos e máximos obtidos pela configuração *VagueGeometry + MBRs* em relação ao *Baseline*, para cada tipo de retorno e considerando o tempo de processamento completo das consultas SOLAP. Considerando apenas o tempo de processamento do predicado topológico vago das consultas SOLAP, esta configuração apresentou ganhos de 92,46% a 95,20% em relação ao *Baseline*. Na Tabela 5.7 são resumidos os ganhos mínimos e máximos obtidos para cada tipo de retorno, considerando apenas o tempo de processamento do predicado das consultas SOLAP.

Dessa forma, as avaliações experimentais conduzidas neste capítulo demonstraram que o TAD VagueGeometry unido com as melhorias propostas se sobressaiu com reduções significativas em relação as soluções existentes, tanto em ambientes de banco de dados espaciais bem como em ambiente de DWG. Como avaliação experimental adicional, o desempenho no processamento de predicados topológicos vagos em bancos de dados espaciais também foi averiguado considerando objetos espaciais vagos que tinham o formato retangular no núcleo e na conjectura. Esta avaliação experimental adicional não foi detalhada neste capítulo devido ao formato simples das geometrias (5 pontos por retângulo), e por isso, não apresenta uma complexidade similar a encontrada em objetos espaciais vagos que representam fenômenos do mundo real. Contudo, reduções de 66,86% a 79,56% foram obtidas pelo TAD VagueGeometry com as melhorias em relação ao *Baseline* no processamento dos predicados topológicos vagos no ambiente de banco de dados espaciais.

Capítulo 6

O TIPO ABSTRATO DE DADOS FUZZYGEOMETRY

Este capítulo detalha uma proposta adicional, o TAD FuzzyGeometry, o qual é uma extensão espacial do SGBD PostgreSQL. Definições dos tipos de dados do TAD FuzzyGeometry, representações, operações e exemplos são detalhados neste capítulo.

6.1 Considerações Iniciais

Com o estudo realizado sobre os modelos *fuzzy* para representação de dados espaciais vagos, adicionalmente foi implementado um TAD baseado neste modelo, denominado FuzzyGeometry. Assim, o TAD FuzzyGeometry é um resultado além do esperado e apresentado nesta dissertação. A documentação completa do TAD FuzzyGeometry pode ser acessada em `http://gbd.dc.ufscar.br/fuzzygeometry/`.

Este capítulo está organizado da seguinte forma. Na Seção 6.2 a implementação do TAD FuzzyGeometry é detalhada, o qual especifica os seus tipos de dados e as suas estruturas de dados. Na Seção 6.3 são descritas as operações do TAD FuzzyGeometry, inclusive suas representações textuais e binárias que permite o uso em aplicações do mundo real. Para exemplificar estas operações, na Seção 6.4 é descrito um exemplo de aplicação o qual usa o TAD FuzzyGeometry para manipular dados espaciais vagos baseado no modelo *fuzzy*. Por fim, na Seção 6.5 as considerações finais sobre o capítulo são feitas.

6.2 Especificação dos Tipos de Dados de FuzzyGeometry

A implementação do TAD FuzzyGeometry segue os mesmos princípios do TAD VagueGeometry, porém baseado no modelo *fuzzy*. Os modelos *fuzzy* dos quais o TAD FuzzyGeometry se baseia são os trabalhos de Dilo, By e Stein (2007), Schneider (2008), uma vez que definem os tipos de dados espaciais *fuzzy*, suas propriedades e operações. Em ambos os trabalhos

são definidos os tipos de dados espaciais *fuzzy* e suas operações. A implementação do TAD FuzzyGeometry foi realizada na linguagem C e reutilizou estrutura de dados do GEOS, além de ter código fonte aberto. O TAD FuzzyGeometry somente provê suporte para pontos *fuzzy* e linhas *fuzzy*. Na Figura 6.1 são mostrados os tipos de dados do TAD FuzzyGeometry: (i) *ponto fuzzy* (FUZZYPOINT), (ii) *multiponto fuzzy* (FUZZYMULTIPOINT), (iii) *linha fuzzy* (FUZZYLINE), e (iv) *multilinha fuzzy* (FUZZYMULTILINE). Assim, um objeto FuzzyGeometry pode ser instanciado como estes tipos de dados.

Em geral, esses tipos de dados espaciais *fuzzy* têm localização e representação no espaço com um grau de pertinência para cada ponto. A localização, a qual pode ser inexata, é representada por pares de coordenadas no espaço Euclidiano. O grau de pertinência atribui valores no intervalo real $]0, 1]$ para cada ponto, com o objetivo de determinar o quanto cada ponto pertence ao fenômeno. A seguir, serão detalhados cada tipo espacial *fuzzy* implementado. A estrutura do tipo de dado FuzzyGeometry, denominada FUZZYGEOM, é apresentada no trecho de código do Algoritmo 10, e detalhada como segue

- O elemento *type* armazena o tipo do dado vago que FUZZYGEOM representa, o qual pode ser FUZZYPOINT, FUZZYMULTIPOINT, FUZZYLINE ou FUZZYMULTILINE;
- O elemento *srid* armazena o SRID da geometria.
- O elemento *bbox* armazena o MBR convencional da geometria e sua estrutura é mostrada no Algoritmo 11. O MBR também foi armazenado uma vez que o seu uso pode diminuir significativamente o processamento de operações que envolvam dados espaciais vagos, conforme investigado no Capítulo 5.
- O elemento *data* armazena o restante das informações relacionadas às geometrias (pontos e linhas *fuzzy*).

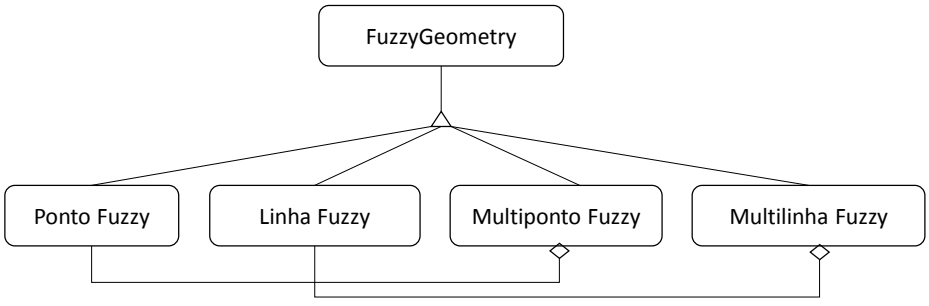


Figura 6.1: Os tipos de dados do TAD FuzzyGeometry

Algoritmo 10 Estrutura de dados para armazenar um objeto FuzzyGeometry em memória principal

```
1: typedef struct {
2:   uint8_t type;
3:   int32_t srid;
4:   BBOX *bbox;
5:   void *data;
6: } FUZZYGEOM;
```

Algoritmo 11 Estrutura de dados para armazenar o MBR de um FuzzyGeometry

```
1: typedef struct {
2:   double xmin;
3:   double xmax;
4:   double ymin;
5:   double ymax;
6: } BBOX;
```

Para facilitar as manipulações em memória primária para os tipos de dados de FuzzyGeometry, foi criada uma nova estrutura que armazena um vetor de pontos *fuzzy* (coordenadas x e y com seu grau de pertinência) para realizar operações genéricas de vetores serializados, tais como inserção, remoção, alteração e recuperação. Esta estrutura é denominada FPOINTARRAY. Todas as manipulações usam funções de manipulação direta de memória primária da biblioteca C, como o *memcpy* e *memmove*.

A estrutura FPOINTARRAY é mostrada no trecho de código do Algoritmo 12 e detalhada como segue. O elemento *serialized_fpointlist* refere-se aos pontos *fuzzy* serializados contendo de forma consecutiva as coordenadas x e y e seu respectivo grau de pertinência. Esses pontos são manipulados por meio de uma estrutura auxiliar denominada FPOINT, que tem como objetivo agrupar as coordenadas x e y com seu respectivo grau de pertinência. Dessa forma, o elemento *serialized_fpointlist* mantém um vetor de FPOINT serializado. Essa abordagem foi adotada para diminuir passos na serialização de FuzzyGeometry. Os outros elementos de FPOINTARRAY, *npoints* e *maxpoints*, são usados para o controle de quantidade de pontos, e referem-se ao número atual e ao número máximo de pontos, respectivamente.

6.2.1 Ponto *Fuzzy* e Multiponto *Fuzzy*

A definição do ponto *fuzzy* (FUZZYPOINT), o qual é de fato manipulado e convertido em FuzzyGeometry (de acordo com a Figura 6.1) é descrita no trecho de código do Algoritmo 13. A distinção entre esta estrutura e o ponto *fuzzy* manipulado de forma serializada pela estrutura FPOINTARRAY (Algoritmo 12) foi criada por razões de organização e distinção de funciona-

Algoritmo 12 Estrutura de dados FPOINTARRAY para manipulação interna dos tipos de dados do FuzzyGeometry

```
1: typedef struct {
2:   uint8_t *serialized_fpointlist;
3:   int npoints;
4:   int maxpoints;
5: } FPOINTARRAY;
6:
7: typedef struct {
8:   double x;
9:   double y;
10:  double u;
11: } FPOINT;
```

lidades específicas. Enquanto a estrutura FPOINT apenas manipula as operações em memória primária de vetores serializados em FPOINTARRAY, a estrutura FUZZYPOINT manipula de fato o ponto *fuzzy* para, por exemplo, ser usado nas operações geométricas de conjunto.

O ponto *fuzzy* de FuzzyGeometry tem uma estrutura similar à estrutura do FUZZYGEOM (Algoritmo 10) para facilitar conversões entre estruturas. Assim, os três primeiros elementos de FUZZYPOINT são os mesmos que os elementos de FUZZYGEOM. O último elemento, *point*, de FUZZYPOINT é um FPOINTARRAY. Esse elemento armazena o valor 1 nos elementos *maxpoints* e *npoints* (Algoritmo 12), desde que ele representa um ponto *fuzzy* simples. De forma contrária, no Algoritmo 14, o elemento *points* armazena o valor $n \in \mathbb{N}$ no elemento *npoints* e um número maior (por exemplo, $n * 2$) em *maxpoints* (Algoritmo 12), uma vez que ele representa um multiponto *fuzzy* (FUZZYMULTIPOINT).

Algoritmo 13 Estrutura de dados para armazenar um objeto do tipo *ponto fuzzy* em memória principal

```
1: typedef struct {
2:   uint8_t type;
3:   int32_t srid;
4:   BBOX *bbox;
5:   FPOINTARRAY *point;
6: } FUZZYPOINT;
```

6.2.2 Linha *Fuzzy* e Multilinha *Fuzzy*

A definição da linha *fuzzy*, além de um FPOINTARRAY, tem também o tipo de interpolação usada para o cálculo do grau de pertinência de um ponto pertencente à linha (*interpolation*). Até o presente momento, somente foi considerado o tipo de interpolação linear. Outras funções de

Algoritmo 14 Estrutura de dados para armazenar um objeto do tipo *multiponto fuzzy* em memória principal

```
1: typedef struct {
2:   uint8_t type;
3:   int32_t srid;
4:   BBOX *bbox;
5:   FPOINTARRAY *points;
6: } FUZZYMULTIPOINT;
```

interpolação podem ser implementadas desde que respeitem a propriedade de monotonicidade.

A estrutura da linha *fuzzy*, denominada FUZZYLINE, é descrita no trecho de código do Algoritmo 15. É importante enfatizar que cada segmento de linha é formado por um par consecutivos de pontos. Além disso, uma linha deve ter pelo menos 2 pontos e não pode conter segmentos que se intersectam. Finalmente, a definição de uma multilinha *fuzzy*, denominada FUZZYMULTILINE, é formada por um conjunto (vetor com elementos distintos) de FUZZYLINE. Sua estrutura é descrita no trecho de código do Algoritmo 16. Ressalta-se que uma multilinha *fuzzy* somente pode ter intersecções em pontos finais de cada linha *fuzzy*. Multilinha *fuzzy* que possuí linhas que se intersectam em pontos que não são finais, não são armazenadas, uma vez que são desconsideradas no processo de validação.

Algoritmo 15 Estrutura de dados para armazenar um objeto do tipo *linha fuzzy* em memória principal

```
1: typedef struct {
2:   uint8_t type;
3:   int32_t srid;
4:   BBOX *bbox;
5:   FPOINTARRAY *point;
6:   uint8_t interpolation;
7: } FUZZYLINE;
```

Algoritmo 16 Estrutura de dados para armazenar um objeto do tipo *multilinha fuzzy* em memória principal

```
1: typedef struct {
2:   uint8_t type;
3:   int32_t srid;
4:   BBOX *bbox;
5:   int ngeoms;
6:   int maxgeoms;
7:   FUZZYLINE **geoms;
8: } FUZZYMULTILINE;
```

6.2.3 A Estrutura para Armazenamento no SGBD PostgreSQL

A última definição necessária para a definição do TAD FuzzyGeometry é a estrutura de armazenamento que é manipulada diretamente pelo núcleo do SGBD PostgreSQL. Para isso, é necessário seguir as especificações do SGBD PostgreSQL descritas na Seção 2.2 do Capítulo 4. Como o TAD FuzzyGeometry tem um tamanho variável o primeiro elemento dessa estrutura deve armazenar, em um inteiro de 4 *bytes*, o tamanho do objeto espacial vago que será armazenado e o restante dos dados da estrutura devem estar serializados.

A nova estrutura, denominada FGSERIALIZED, é descrita no trecho de código do Algoritmo 17. Como definição, ela contém como primeiro elemento (*size*) o tamanho do objeto, o qual é manipulado somente utilizando macros da biblioteca interna do SGBD PostgreSQL. O segundo elemento (*srid*) armazena de forma serializada em um vetor SRID do objeto. O terceiro elemento (*data*) também armazena de forma serializada em um vetor os valores de cada objeto espacial *fuzzy* (por exemplo, uma linha *fuzzy*) a serem armazenados, os quais podem variar conforme cada tipo de dado. Tais dados são serializados utilizando aritmética de ponteiros e a função *memcpy* da biblioteca padrão da linguagem C.

Algoritmo 17 Estrutura de dados para armazenar um objeto FuzzyGeometry no SGBD PostgreSQL

```
1: typedef struct {
2:   uint32_t size;
3:   uint8_t srid[4];
4:   uint8_t data[1];
5: } FGSERIALIZED;
```

O alinhamento de 8 *bytes* é também seguida pela estrutura FGSERIALIZED. Os dois primeiros elementos desta estrutura compõe 8 *bytes*. O elemento *data* também segue este alinhamento conforme demonstrado pela ordem de serialização detalhada na Tabela 6.1. Para cada tipo de dado existe uma ordem de serialização. Em geral, primeiramente é serializado o MBR do objeto somente se o mesmo não for vazio, visando o acesso imediato. Em seguida, o tipo de dado do objeto FuzzyGeometry é serializado e, posteriormente, sua quantidade de pontos. Por fim, cada ponto é serializado na seguinte ordem: (i) coordenada x; (ii) coordenada y; e, (iii) seu respetivo grau de pertinência u. Na Tabela 6.1, a notação $<>$ representa que o elemento tem 4 *bytes*, enquanto que a notação $[\,]$ representa que o elemento tem um tamanho múltiplo de 8 *bytes*. Por fim, para manter o alinhamento de 8 *bytes*, é necessário adicionar um *padding* de 4 bytes para um objeto FuzzyGeometry do tipo linha *fuzzy*.

Tabela 6.1: Ordem de serialização de um objeto espacial vago do TAD FuzzyGeometry.

Tipo de Dado	Ordem de Serialização
Ponto *fuzzy* (FUZZYPOINT)	[MBR] <tipo de dado do objeto espacial vago> <número de pontos *fuzzy* (0 para vazio e 1 caso contrário)> [coordenada *x*] [coordenada *y*] [grau de pertinência *u*]
Multiponto *fuzzy* (FUZZYMULTIPOINT)	[MBR] <tipo de dado do objeto espacial vago> <número de pontos *fuzzy* (0 para vazio e 1 caso contrário)> [coordenada *x*] [coordenada *y*] [grau de pertinência *u*]
Linha *fuzzy* (FUZZYLINE)	[MBR] <tipo de dado do objeto espacial vago> <tipo de interpolação> <número de pontos *fuzzy* (0 para vazio e 1 caso contrário)> <*padding*> [coordenada *x*] [coordenada *y*] [grau de pertinência *u*]
Multilinha *fuzzy* (FUZZYMULTILINE)	[MBR] <tipo de dado do objeto espacial vago> <número de linhas *fuzzy*> para cada linha *fuzzy i* [serialização da linha *fuzzy i*]

6.2.4 A Definição no SGBD PostgreSQL

No Algoritmo 18 é descrito o comando SQL CREATE TYPE para criar o tipo de dado FuzzyGeometry no SGBD PostgreSQL. O parâmetro que indica o tamanho interno do dado (*internallength*) é variável. As funções de *input* e *output* são definidas conforme as especificações do SGBD PostgreSQL.

As funções *send* e *receive* foram definidas visando prover melhor desempenho para casos quando a entrada e saída forem binárias. Além disso, também foram definidas as funções *typmod_in* e *typmod_out* para tratar as restrições de tipos modificadores. O delimitador ':' também foi definido para vetores na linguagem SQL do tipo FuzzyGeometry. O método de armazenamento escolhido foi o *main*. A categoria do TAD FuzzyGeometry é do tipo geométrico e categorizado como 'G'. Por fim, o alinhamento dos dados serializados tem como tamanho padrão 8 *bytes* devido às coordenadas geográficas.

Algoritmo 18 A especificação do tipo FuzzyGeometry no SGBD PostgreSQL

```
CREATE TYPE FuzzyGeometry (
  internallength = variable,
  input = fg_in,
  output = fg_out
  send = fg_send,
  receive = fg_recv,
  typmod_in = fg_typmod_in,
  typmod_out = fg_typmod_out,
  delimiter = ':',
  category = 'G',
  alignment = double,
  storage = main
);
```

Da mesma forma que o TAD VagueGeometry, as funções para manipular o TAD FuzzyGeometry, foram definidas na linguagem C e assim associada para a linguagem SQL. Tal associação é criada por meio do comando SQL CREATE FUNCTION, o qual toma como parâmetro a função na linguagem C da extensão, de acordo com a documentação do SGBD PostgreSQL. Na Seção 6.3 são detalhados todas as operações do TAD FuzzyGeometry.

6.3 Operações do TAD FuzzyGeometry

Foram implementadas várias funções categorizadas como se segue:

- Funções de entrada e saída: recebe dados espaciais vagos em sua forma textual ou binária e os armazenam internamente, bem como o inverso. Estas funções são detalhadas na Seção 6.3.1;
- Operações genéricas: operações que podem ser chamadas independente do tipo do objeto FuzzyGeometry. Estas funções são detalhadas na Seção 6.3.2;
- Operações geométricas de conjuntos: operações de união, intersecção e diferença entre dados espaciais vagos. Estas funções são detalhadas na Seção 6.3.3.

Nas próximas seções, todas as operações do TAD FuzzyGeometry são especificadas e detalhadas. É importante notar que todas as operações usam o prefixo FG_ em suas assinaturas para denotar que é uma função do TAD FuzzyGeometry.

6.3.1 Funções de Entrada e Saída

Assim como para o TAD VagueGeometry, as funções de entradas de entrada e saída do TAD FuzzyGeometry usam representações textuais e binárias. Dessa forma, foi necessário definir as representações textuais dos tipos de dados espaciais *fuzzy*. As definições textuais de cada tipo de dado implementado são detalhadas a seguir. Em especial, foi implementado um conversor da representação textual para a representação interna de FuzzyGeometry. A abordagem utilizada consiste em uma representação híbrida entre a representação textual WKT e a representação de conjuntos *fuzzy*, principalmente quanto à utilização da barra para indicar o grau de pertinência associado. Objetos espaciais *fuzzy* vazios, os quais não contém nenhuma coordenada e graus de pertinências são especificados usando a palavra chave EMPTY depois do nome do tipo de dado. A representação textual é definida como *Fuzzy Well-Known Text* (FWKT). Seja x e y um par de coordenadas, u um grau de pertinência no intervalo real $]0,1]$ e $k, j \in \mathbb{N}$, foi definido a representação textual para o ponto *fuzzy* (i), multiponto *fuzzy* (ii), linha *fuzzy* (iii) e multilinha *fuzzy* (iv) como segue

(i) FUZZYPOINT$(u/x\ y)$

(ii) FUZZYMULTIPOINT$(u_1/x_1\ y_1, ..., u_k/x_k\ y_k)$

(iii) FUZZYLINESTRING$(u_1/x_1\ y_1, ..., u_k/x_k\ y_k)$

(iv) FUZZYMULTILINESTRING$((u_1/x_1\ y_1, ..., u_k/x_k\ y_k)_1, ..., (u_1/x_1\ y_1, ..., u_{k_j}/x_{k_j}\ y_{k_j})_j)$

Quando o SRID é armazenado juntamente a representação textual, a forma estendida de FWKT é formado (i.e. *Extended*-FWKT - EFWKT).

O formato binário de um objeto FuzzyGeometry consiste na transformação binária de cada elemento. Seja $id(A)$ uma função que extrai o identificador numérico 1 quando A é um ponto *fuzzy*, 2 quando A é uma linha *fuzzy*, 3 quando A é um multiponto *fuzzy* e 4 quando A é uma multilinha *fuzzy*, e *binary* uma função que extrai o formato binário a partir de um objeto numérico, além de n ser número de pontos de um objeto, l o número de linhas de uma multilinha *fuzzy*, e p um ponto *fuzzy* com coordenadas x e y com seu respectivo grau de pertinência u. A representação *Fuzzy Well-Known Binary* (FWKB) para ponto *fuzzy* (i), multiponto *fuzzy* (ii), linha *fuzzy* (iii) e multilinha *fuzzy* (iv) é definido como segue

(i) Se A é do tipo ponto *fuzzy*: $FWKB(A) = endianess + id(A) + binary(u) + binary(x) + binary(y)$

(ii) Se A é do tipo multiponto *fuzzy*: $FWKB(A) = endianess + id(A) + binary(n) + \sum_{p_i \in A}^{n}(binary(u)_{p_i} + binary(x)_{p_i} + binary(y)_{p_i})$

(iii) Se A é do tipo linha *fuzzy*: $FWKB(A) = endianess + id(A) + binary(n) + \sum_{p_i \in A}^{n}(binary(u)_{p_i} + binary(x)_{p_i} + binary(y)_{p_i})$

(iv) Se A é do tipo multilinha *fuzzy*: $FWKB(A) = endianess + id(A) + binary(l) + \sum_{l_i \in A}^{l} FWKB(A_{l_i})$

O símbolo de soma é usado para denotar a união entre os dados serializados e não para realizar a soma aritmética. Além disso, *endianess* indica como os *bytes* estão organizados em memória principal. Ademais, é possível adicionar o SRID na representação FWKB, formando assim o *Extended*-FWKB (EFWKB).

Com as representações textuais e binárias definidas, é possível inserir e visualizar objetos espaciais vagos por meio de funções específicas. As assinaturas das funções que transformam as representações FWKT, EFWKT, FWKB e EFWKB para o formato interno são listadas respectivamente a seguir

(i) *FG_FuzzyGeomFromText*(text *FWKT*, integer *SRID*) → FuzzyGeometry

(ii) *FG_FuzzyGeomFromEFWKT*(text *EFWKT*) → FuzzyGeometry

(iii) *FG_FuzzyGeomFromBinary*(bytea *FWKB*, integer *SRID*) → FuzzyGeometry

(iv) *FG_FuzzyGeomFromEFWKB*(bytea *EFWKB*) → FuzzyGeometry

As funções de saída, retornam as representações definidas nessa seção (FWKT, EFWKT, FWKB e EFWKB) e suas respectivas assinaturas são

(i) *FG_AsText*(FuzzyGeometry *fg*) → text

(ii) *FG_AsEFWKT*(FuzzyGeometry *fg*) → text

(iii) *FG_AsFWKB*(FuzzyGeometry *fg*) → bytea

(iv) *FG_AsEFWKB*(FuzzyGeometry *fg*) → bytea

6.3.2 Operações Genéricas

Operações genéricas foram definidas para manipular qualquer tipo de dado do TAD FuzzyGeometry, elas são o núcleo, fronteira, concentração e dilatação. Sejam *fg* um objeto FuzzyGeometry, $p > 1$ e $r \in]0,1[$, foram definidas as seguintes assinaturas

(i) *FG_Core*(FuzzyGeometry *fg*) → FuzzyGeometry

(ii) *FG_Boundary*(FuzzyGeometry *fg*) → FuzzyGeometry

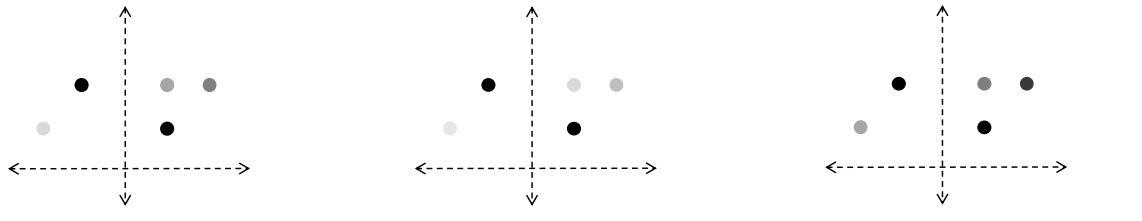
(iii) *FG_Concentration*(FuzzyGeometry *fg*, *p*) → FuzzyGeometry

(iv) *FG_Dilation*(FuzzyGeometry *fg*, *r*) → FuzzyGeometry

Em geral estas operações foram baseadas na teoria de conjuntos *fuzzy*. As operações (i) e (ii) extraem o núcleo (i.e. todos os pontos com grau de pertinência iguais a 1) e a fronteira vaga (i.e. todos os pontos com grau de pertinência menores que 1) de um objeto espacial *fuzzy*. As operações (iii) e (iv) modificam os graus de pertinência de um objeto FuzzyGeometry, aumentando ou diminuindo a vagueza espacial, respectivamente. Na Figura 6.2b é mostrado um exemplo de concentração sobre pontos *fuzzy* da Figura 6.2a com *p* igual a 2. Já na Figura 6.2c é mostrado um exemplo de dilatação sobre os pontos *fuzzy* da Figura 6.2a com *r* igual a 0.5. Nestas figuras, são mostrados as representações textuais FWKT de cada objeto FuzzyGeometry.

6.3.3 Operações Geométricas de Conjuntos

As operações geométricas de conjuntos são a união *fuzzy*, intersecção *fuzzy* e diferença *fuzzy*. Estas operações somente são processadas se, e somente se, os MBRs dos dois ob-



FUZZYMULTIPOINT(0.2/-2 1 + 1/-1 2 + 1/1 1 + 0.4/1 2 + 0.6/2 2)

(a)

FUZZYMULTIPOINT(0.04/-2 1 + 1/-1 2 + 1/1 1 + 0.16/1 2 + 0.36/2 2)

(b)

FUZZYMULTIPOINT(0.447/-2 1 + 1/-1 2 + 1/1 1 + 0.632/1 2 + 0.774/2 2)

(c)

Figura 6.2: Resultado da operação de concentração (b) e dilatação (c) sobre um objeto FuzzyGeometry (a) do tipo FUZZYMULTIPOINT, com suas respectivas representações textuais FWKT.

jetos FuzzyGeometry se intersectarem. Seja *fg* um objeto FuzzyGeometry, *t* uma t-norma (Seção 2.4.2.1), *s* uma t-conorma (Seção 2.4.2.1) e *d* um operador de diferença (Seção 2.4.2.1), foram definidas as seguintes assinaturas

(i) *FG_Union*(FuzzyGeometry *fg*, FuzzyGeometry *fg*, text *s*) → FuzzyGeometry

(ii) *FG_Union*(FuzzyGeometry *fg*) → FuzzyGeometry

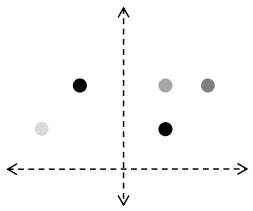
(iii) *FG_Intersection*(FuzzyGeometry *fg*, FuzzyGeometry *fg*, text *t*) → FuzzyGeometry

(iv) *FG_Difference*(FuzzyGeometry *fg*, FuzzyGeometry *fg*, text *d*) → FuzzyGeometry

A operação de união (i) entre objetos espaciais *fuzzy* é realizada pela união espacial e a união *fuzzy* entre os graus de pertinência. A união *fuzzy* pode ser executada usando uma t-conorma específica. Além disso, a união é somente executada para dados espaciais *fuzzy* do mesmo tipo. Na Figura 6.4b é mostrado o resultado da união *fuzzy*, considerando a soma probabilística como t-conorma, sobre os pontos *fuzzy* da Figura 6.3, com sua respectiva representação textual FWKT. A operação de união também pode ser utilizada como uma função de agregação (ii). Esta função de agregação considera a t-conorma padrão para o cálculo dos graus de pertinência.

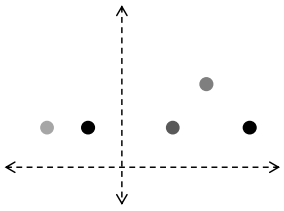
A operação de interseção (iii) entre dados espaciais *fuzzy* é realizada pela intersecção espacial e a intersecção *fuzzy* dos graus de pertinência. A intersecção *fuzzy* pode ser executada usando uma t-norma específica. Assim como na união, a intersecção pode ser somente chamada por tipos iguais de dados. Na Figura 6.4b é mostrado o resultado da intersecção *fuzzy*, considerando o produto como t-norma, sobre os pontos *fuzzy* da Figura 6.3, com sua respectiva representação textual FWKT.

A operação de diferença (iv) entre dados espaciais *fuzzy* é realizada pela diferença espacial e calculado os graus de pertinência de localizações em comum de acordo com um operador específico de diferença (e.g. diferença *fuzzy* e diferença absoluta). Assim como na união e intersecção, a diferença pode ser somente chamada por tipos iguais de dados. Na Figura 6.4c é mostrado o resultado da diferença *fuzzy*, considerando a diferença *fuzzy* como operador, sobre os pontos *fuzzy* da Figura 6.3, com sua respectiva representação textual FWKT.



FUZZYMULTIPOINT(0.2/-2 1 + 1/-1 2 + 1/1 1 + 0.4/1 2 + 0.6/2 2)

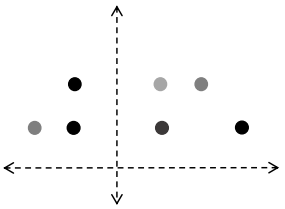
(a)



FUZZYMULTIPOINT(0.4/-2 1 + 1/-1 1 + 0.8/1 1 + 0.6/2 2 + 1/3 1)

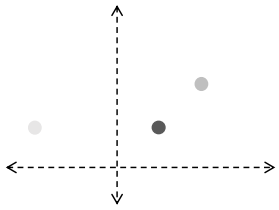
(b)

Figura 6.3: Dois objetos FuzzyGeometry (a) e (b) do tipo FUZZYMULTIPOINT com suas respectivas representações textuais FWKT.



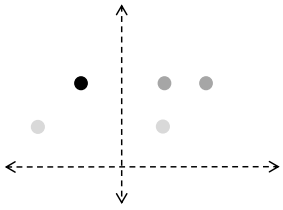
FUZZYMULTIPOINT(0.68/-2 1 + 1/-1 2 + 1/-1 1 + 1/1 1 + 0.4/1 2 + 0.84/2 2 + 1/3 1)

União



FUZZYMULTIPOINT(0.08/-2 1 + 0.8/1 1 + 0.36/2 2)

Intersecção



FUZZYMULTIPOINT(0.2/-2 1 + 1/-1 1 + 0.2/1 1 + 0.4/1 2 + 0.4/2 2)

Diferença

Figura 6.4: Resultado das operações geométricas de conjunto sobre os objetos FuzzyGeometry (a) e (b) da Figura 6.3 com suas respectivas representações textuais FWKT.

6.4 Exemplo de Aplicação

O objetivo dessa seção é demonstrar com exemplos, as operações e o uso do TAD FuzzyGeometry em uma aplicação. O mesmo exemplo de aplicação utilizado na Seção 4.5 é utilizado aqui. Porém, sem considerar as relações que continham regiões vagas, uma vez que o TAD FuzzyGeometry não provê suporte a este tipo de representação até o momento. O contexto deste exemplo, portanto, é gerenciar as pragas e rotas de animais em um ambiente agrícola. Nesse sentido, as seguintes relações são consideradas, agora considerando dados espaciais *fuzzy*

animal(*id*:integer, *rota*:fuzzyline, *descricao*:text, *animal*:text)

praga(*id*:integer, *praga*:fuzzymultipoint, *descricao*:text, *tipo*:text)

Como diferenciais em relação ao outro exemplo, o atributo *rota* na relação *animal* é representada por linhas *fuzzy* e o atributo *praga* da relação *praga* é representado por pontos *fuzzy*

indicando a possibilidade de um ponto ser uma praga em potencial.

Com as relações previamente definidas e com o TAD FuzzyGeometry, é possível criar, de maneira muito similar ao TAD VagueGeometry, as tabelas com os atributos espaciais vagos usando os tipos de dados do TAD FuzzyGeometry. O código SQL para criar a tabela da relação *animal* pode ser escrita como

```
CREATE TABLE animal(
id INTEGER PRIMARY KEY,
rota FUZZYGEOMETRY(FUZZYLINESTRING),
descricao TEXT,
animal TEXT);
```

É importante notar a restrição em parenteses do atributo *rota*, o qual aceitará somente linhas *fuzzy* tal como na restrição definida na especificação do exemplo de aplicação. A criação da relação *praga* é similar, porém, mudando a restrição para seu respectivo tipo de dado (FUZZYMULTIPOINT).

Após a criação das tabelas, é possível inserir objetos do tipo FuzzyGeometry. Assim como no TAD VagueGeometry, existem duas diferentes formas para inserir um objeto do tipo FuzzyGeometry em uma tabela. A primeira delas é usar a forma canônica provida pelo SGBD PostgreSQL, enquanto que a segunda forma é por meio do uso de funções específicas do TAD FuzzyGeometry (Seção 6.3.1). Considerando a tabela *praga*, um novo ponto *fuzzy* representando a praga do tipo *Z* é inserido usando o seguinte código SQL

```
INSERT INTO praga
VALUES (1,
FG_FuzzyGeomFromText('FUZZYMULTIPOINT(1/0 0, 0.5/10 10, 0.1/8 8)', 4326),
'severa', 'tipo Z')
```

Considerando que as tabelas têm todos os objetos inseridos, é possível executar consultas SQL sobre os objetos FuzzyGeometry. Uma possível consulta é retornar todas as partes em comum entre as rotas de todos os animais. Essa consulta usa a operação de intersecção entre os objetos distintos da tabela *animal* e pode ser escrita como

```
SELECT FG_AsText(FG_Intersection(A.rota, B.rota, 'standard t-norm')),
  A.animal, B.animal
```

```
FROM animal A, animal B
WHERE A.id <> B.id
```

Outra consulta, é a utilização da função de agregação da união entre todas as pragas severas, para assim tomar um controle específico

```
SELECT FG_AsText(FG_Union(praga))
FROM praga
WHERE tipo = 'severa'
```

Outro tipo de consulta é mudar o significado da vagueza espacial intensificando ou suavizando o fenômeno. Por exemplo, é possível utilizar a concentração para relaxar a rota de um animal específico. Como resultado, a seguinte consulta muda o grau de pertinência de todos os pontos da rota de um animal, intensificando a vagueza espacial (ou seja, aumentando o grau de incerteza). Assim, esta consulta pode ser escrita como

```
SELECT FG_AsText(FG_Concentration(rota, 2))
FROM animal
WHERE animal = 'X'
```

Por outro lado, o seguinte comando SQL usa a dilatação para aumentar o grau de pertinência das rotas de um outro animal. Assim, como resultado, a seguinte consulta aumenta o grau de certeza das rotas de um animal específico

```
SELECT FG_AsText(FG_Dilation(rota, 0.5))
FROM animal
WHERE animal = 'Y'
```

6.5 Considerações Finais

Neste capítulo, o TAD FuzzyGeometry foi apresentado. Este TAD foi adicionalmente implementado uma vez que o modelo *fuzzy* permite representar a vagueza espacial em mais níveis. Mais especificamente cada ponto pode ter um nível vagueza em um intervalo real $]0, 1]$.

O TAD FuzzyGeometry foi detalhado desde sua especificação internamente no SGBD PostgreSQL até suas operações. Além disso, foram apresentados representações textuais e binárias

para objetos espaciais vagos baseados no modelo *fuzzy*. Suas operações vão desde as baseadas na teoria de conjuntos *fuzzy* (operações genéricas) até as operações geométricas de conjunto.

Como principais vantagens, o TAD FuzzyGeometry é de código fonte aberto e implementado sobre outros projetos de código fonte aberto, tais como o SGBD PostgreSQL e o PostGIS. Apesar da atual implementação somente prover suporte para pontos *fuzzy* e linhas *fuzzy*, já é inferir que a teoria de conjuntos *fuzzy* proporciona um maior poder de representação da vagueza espacial que os modelos exatos. Por outro lado, este TAD claramente têm mais complexidade em suas operações e devido a isso maiores estudos são necessários para implementações das regiões *fuzzy* e outras operações, tais como os predicados topológicos *fuzzy*.

Capítulo 7

CONCLUSÕES

Este capítulo concluí esta pesquisa de mestrado e aponta os trabalhos futuros para a continuidade deste trabalho.

Esta dissertação apresentou conceitos sobre a vagueza espacial e suas representações. Dados espaciais vagos são propostos para representar a vagueza espacial, a qual está presente em representações espaciais de fenômenos do mundo real que contêm localização incerta, ou fronteiras inexatas ou interiores não bem definidos. Com dados espaciais vagos é possível representar tais fenômenos, os quais não são possíveis de se representar utilizando somente dados espaciais *crisp*.

Contudo, até então, sistemas gerenciadores de banco de dados espaciais não ofereciam um tipo abstrato de dados (TAD) para manipular dados espaciais vagos. Consequentemente, não havia suporte também em operações SOLAP sobre DWGs que continham dados espaciais vagos para auxiliar na tomada de decisão estratégica considerando a vagueza espacial. Um TAD é de extrema importância para uma aplicação uma vez que é responsável por facilitar a manipulação de dados complexos, escondendo toda a complexidade envolvida do usuário final. Outra vantagem é que um TAD beneficia a legibilidade da consulta, tornando-a mais enxuta e simples.

Nesse sentido, esta pesquisa de mestrado visou enfocar neste problema por meio da investigação de modelos para representar dados espaciais vagos e da proposta de TADs para sua incorporação em banco de dados espaciais e em DWGs. Em geral, existem vários tipos de representações de dados espaciais vagos, onde os modelos exatos foram mais investigados nesta dissertação.

Os modelos exatos foram mais investigados devido a reutilização de conceitos e estruturas já bem conhecidas em banco de dados espaciais e, consequentemente, em DWGs. Com isso, esta pesquisa de mestrado propôs um novo TAD, denominado como VagueGeometry, para ma-

nipular dados espaciais vagos baseado no modelo exato VASA (PAULY; SCHNEIDER, 2010). Este TAD foi especificado, desenvolvido, documentado e validado por meio de testes nas operações e avaliação experimental. É importante notar que no processo de validação, foi identificado inconsistências na definição da operação de intersecção envolvendo dados espaciais vagos de tipos diferentes e uma proposta de correção foi definida e implementada.

Em relação a avaliação experimental, o TAD VagueGeometry foi avaliado em dois ambientes. O primeiro deles visou avaliar o desempenho no processamento de consultas envolvendo predicados topológicos vagos em SGBDEs. O TAD VagueGeometry foi comparado com a forma de armazenamento de objetos espaciais vagos somente utilizando as funcionalidades existentes em SGBDEs. Como resultado, o TAD VagueGeometry apresentou ganhos de 42,36% a 65,94%. Adicionalmente, com base nestes resultados, melhorias no TAD VagueGeometry foram propostas para melhorar ainda mais o desempenho no processamento de predicados topológicos vagos. A primeira proposta de melhoria, a qual pré-armazena a união do núcleo e conjectura de objetos espaciais vagos, não apresentou reduções. Porém, ao se combinar esta melhoria com o uso de MBRs, conseguiu-se reduzir de 57,24% a 77,95% em relação a proposta inicial do TAD VagueGeometry.

O segundo ambiente da avaliação experimental visou avaliar a incorporação de dados espaciais vagos em DWGs. Dessa forma, investigou-se o uso do TAD VagueGeometry em consultas SOLAP que continha predicados topológicos vagos. Os resultados mostraram que o TAD VagueGeometry proporcionou ganhos de 0,27% a 4,98% no processamento de consultas SOLAP. Além disso, constatou-se que o processamento dos predicados topológicos vagos corresponderam apenas de 0,12% a 3,86% do tempo de execução da consulta SOLAP. Assim, as operações de junções, filtros, agregações e ordenações das consultas demandaram mais tempo de processamento que os predicados. Outro fator importante para este fator, foi o otimizador de consultas, que determinava em que momento o predicado seria executado. Quando considerado somente o tempo de execução dos predicados nas consultas SOLAP, o TAD VagueGeometry, com o uso de MBRs, apresentou ganhos de 92,46% a 95,20% em relação as soluções existentes.

No melhor do nosso conhecimento, o TAD VagueGeometry, é o primeiro TAD completo, quando comparado aos trabalhos correlatos, que disponibiliza operações essenciais para manipular dados espaciais vagos eficientemente. Sua completude se deve ao fato de prover suporte as mais importantes operações espaciais que um tipo de dado espacial deve possuir, como por exemplo, predicados topológicos, operadores geométricos de conjunto, operadores numéricos e operadores específicos.

Além da investigação do modelo exato, foi também considerado o estudo dos modelos

fuzzy para representação da vagueza espacial, devido a possibilidade de se modelar em vários níveis, a vagueza espacial. Como resultado, um novo TAD foi proposto, o qual foi denominado como FuzzyGeometry e baseado em diversos trabalhos (e.g. Schneider (1999), Dilo, By e Stein (2007)). Sua primeira versão foi especificada, desenvolvida e documentada nesta dissertação. Apesar de não prover todas as operações importantes, como os predicados topológicos, o TAD FuzzyGeometry demonstra a possibilidade de representar diversos fenômenos do mundo real e a realização de consultas considerando a vagueza espacial baseada na teoria de conjuntos *fuzzy*.

Como trabalhos futuros, existem diversos pontos a serem estudados e investigados listados a seguir

- Investigação do otimizador de consultas do SGBD para o processamento de consultas SOLAP em DWGs que envolvem dados espaciais vagos. A inclusão de estatísticas sobre objetos espaciais vagos pode impactar no momento da criação do plano de consultas, e com isso melhorar ou não, o processamento de consultas. Estatísticas em objetos espaciais *crisp* já são computadas para este fim, como o feito pela extensão espacial PostGIS. Logo, a coleta de estatísticas de objetos espaciais vagos será também investigada.

- Avaliação experimental no processamento de junção espacial envolvendo objetos espaciais vagos. Este tipo de consulta é amplamente utilizada em aplicações e requerem um alto processamento. Devido a isso, é importante um estudo do comportamento do TAD VagueGeometry neste tipo de consulta.

- Utilização de dados reais, além da variação no volume de dados e número de objetos no núcleo e conjectura na avaliação experimental, tanto para o ambiente de banco de dados espaciais como em DWGs.

- Continuação no desenvolvimento do TAD FuzzyGeometry e a sua incorporação em DWGs. Esta incorporação engloba os predicados topológicos *fuzzy* para o processamento de consultas SOLAP. Um modelo de dados espaciais *fuzzy* que também pode ser considerado nesse desenvolvimento é a álgebra espacial plateau (SCHNEIDER, 2014).

- Visualização de objetos espaciais vagos em ferramentas SOLAP, uma vez que a vagueza espacial pode ter diversas partes disjuntas e níveis de vagueza.

REFERÊNCIAS

ALTMAN, D. Fuzzy set theoretic approaches for handling imprecision in spatial analysis. *International Journal of Geographical Information Systems*, v. 8, n. 3, p. 271–289, 1994.

BEJAOUI, L. et al. Qualified topological relations between spatial objects with possible vague shape. *Int. J. Geogr. Inf. Sci.*, Taylor & Francis, Inc., Bristol, PA, USA, v. 23, n. 7, p. 877–921, jul. 2009. ISSN 1365-8816. Disponível em: <http://dx.doi.org/10.1080/13658810802022814>.

BEJAOUI, 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. Disponível em: <http://dx.doi.org/10.1007/s10707-010-0104-5>.

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, p. 1–25, 2012. ISSN 0717-5000.

CHEN, T. et al. iBLOB: Complex object management in databases through intelligent binary large objects. In: DEARLE, A.; ZICARI, R. (Ed.). *Objects and Databases*. Springer Berlin Heidelberg, 2010, (Lecture Notes in Computer Science, v. 6348). p. 85–99. ISBN 978-3-642-16091-2. Disponível em: <http://dx.doi.org/10.1007/978-3-642-16092-9_10>.

CHENG, R.; KALASHNIKOV, D.; PRABHAKAR, S. Querying imprecise data in moving object environments. *Knowledge and Data Engineering, IEEE Transactions on*, v. 16, n. 9, p. 1112–1127, Sept 2004. ISSN 1041-4347.

CHENG, R.; KALASHNIKOV, D. V.; PRABHAKAR, S. Evaluating probabilistic queries over imprecise data. In: *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*. New York, NY, USA: ACM, 2003. (SIGMOD '03), p. 551–562. ISBN 1-58113-634-X. Disponível em: <http://doi.acm.org/10.1145/872757.872823>.

CHENG, R. et al. Efficient indexing methods for probabilistic threshold queries over uncertain data. In: *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*. VLDB Endowment, 2004. (VLDB '04), p. 876–887. ISBN 0-12-088469-0. Disponível em: <http://dl.acm.org/citation.cfm?id=1316689.1316765>.

CIFERRI, C. et al. Cube algebra: A generic user-centric model and query language for olap cubes. *Int. J. Data Warehous. Min.*, IGI Global, Hershey, PA, USA, v. 9, n. 2, p. 39–65, abr. 2013. ISSN 1548-3924. Disponível em: <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*. Tese (Doutorado) — Universidade Federal de Pernambuco, 2002.

CLEMENTINI, E.; DI FELICE, P. A comparison of methods for representing topological relationships. *Inf. Sci. Appl.*, Elsevier Science Inc., New York, NY, USA, v. 3, n. 3, p. 149–178, maio 1995. ISSN 0020-0255. Disponível em: <http://dl.acm.org/citation.cfm?id=204201.204203>.

CLEMENTINI, E.; DI FELICE, P.; OOSTEROM, P. van. A small set of formal topological relationships suitable for end-user interaction. In: ABEL, D.; CHIN OOI, B. (Ed.). *Advances in Spatial Databases*. Springer Berlin Heidelberg, 1993, (Lecture Notes in Computer Science, v. 692). p. 277–295. ISBN 978-3-540-56869-8. Disponível em: <http://dx.doi.org/10.1007/3-540-56869-7_16>.

CLEMENTINI, E.; FELICE, P. D. A model for representing topological relationships between complex geometric features in spatial databases. *Information Sciences*, v. 90, n. 1-4, p. 121 – 136, 1996. ISSN 0020-0255. Disponível em: <http://www.sciencedirect.com/science/article/pii/0020025595002898>.

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. Disponível em: <http://www.sciencedirect.com/science/article/pii/0097849394900078>.

COHN, A.; GOTTS, N. The 'egg-yolk' representation of regions with indeterminate boundaries. In: *Geographic objects with indeterminate boundaries*. [S.l.]: Francis Taylor, 1995. p. 171–187.

CORTI, P. et al. *PostGIS Cookbook*. [S.l.]: Packt Publishing, 2014. ISBN 1849518661, 9781849518666.

DAVID, P.; SOMODEVILLA, M.; PINEDA, I. Fuzzy spatial data warehouse: A multidimensional model. In: *Current Trends in Computer Science, 2007. ENC 2007. Eighth Mexican International Conference on*. [S.l.: s.n.], 2007. p. 3–9.

DB2 Spatial Extender. 2014. Disponível em: <http://www-03.ibm.com/software/products/en/db2spaext>.

DELIPETREV, B.; JONOSKI, A.; SOLOMATINE, D. P. Development of a web application for water resources based on open source software. *Comput. Geosci.*, Pergamon Press, Inc., Tarrytown, NY, USA, v. 62, p. 35–42, jan. 2014. ISSN 0098-3004. Disponível em: <http://dx.doi.org/10.1016/j.cageo.2013.09.012>.

DILO, A. *Representation of and reasoning with vagueness in spatial information: A system for handling vague objects*. Tese (Doutorado) — International Institute for Geo-information Science & Earth Observation, 2006.

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 978-3-540-33288-6. Disponível em: <http://dx.doi.org/10.1007/3-540-33289-8_12>.

DILO, A.; BY, R. A. de; STEIN, A. A system of types and operators for handling vague spatial objects. *International Journal of Geographical Information Science*, Taylor & Francis,

Inc., Bristol, PA, USA, v. 21, n. 4, p. 397–426, jan. 2007. ISSN 1365-8816. Disponível em: <http://dx.doi.org/10.1080/13658810601037096>.

DUBOIS, D.; PRADE, H.; PRADE, H. *Fundamentals of Fuzzy Sets*. Springer US, 2000. (The Handbooks of Fuzzy Sets). ISBN 9780792377320. Disponível em: <http://books.google.com.br/books?id=WhqPtfVEnSoC>.

ESR, I. *ESRI Shapefile Technical Description*. [S.l.], jul. 1998. Disponível em: <http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf>.

GAEDE, V.; GÜNTHER, O. Multidimensional access methods. *ACM Comput. Surv.*, ACM, New York, NY, USA, v. 30, n. 2, p. 170–231, jun. 1998. ISSN 0360-0300. Disponível em: <http://doi.acm.org/10.1145/280277.280279>.

GEOS - Geometry Engine Open Source. 2014. Disponível em: <http://trac.osgeo.org/geos/>.

GKATZOFLIAS, D.; MELLIOS, G.; SAMARAS, Z. Development of a web gis application for emissions inventory spatial allocation based on open source software tools. *Comput. Geosci.*, Pergamon Press, Inc., Tarrytown, NY, USA, v. 52, p. 21–33, mar. 2013. ISSN 0098-3004. Disponível em: <http://dx.doi.org/10.1016/j.cageo.2012.10.011>.

GUESGEN, H. W. From the egg-yolk to the scrambled-egg theory. In: *Proceedings of the Fifteenth International Florida Artificial Intelligence Research Society Conference*. AAAI Press, 2002. p. 476–480. ISBN 1-57735-141-X. Disponível em: <http://dl.acm.org/citation.cfm?id=646815.708590>.

GÜTING, R. H. An introduction to spatial database systems. *The VLDB Journal*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, v. 3, n. 4, p. 357–399, out. 1994. ISSN 1066-8888. Disponível em: <http://dl.acm.org/citation.cfm?id=615204.615206>.

JAMSHIDI, M.; VADIEE, N.; ROSS, T. J. (Ed.). *Fuzzy Logic and Control: Software and Hardware Applications*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1993. ISBN 0-13-334251-4.

KIMBALL, R.; ROSS, M. *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling*. 2nd ed. New York, NY, USA: John Wiley & Sons, Inc., 2002. ISBN 0471200247, 9780471200246.

KLEMENT, E.; MESIAR, R.; PAP, E. Families of t-norms. In: *Triangular Norms*. Springer Netherlands, 2000, (Trends in Logic, v. 8). p. 101–119. ISBN 978-90-481-5507-1. Disponível em: <http://dx.doi.org/10.1007/978-94-015-9540-7_4>.

KRAIPEERAPUN, P. *Implementation of vague spatial objects*. Dissertação (Mestrado) — International Institute for Geo-Information Science and Earth Observation, 2004.

LI, R. et al. Uncertain spatial data handling: Modeling, indexing and query. *Comput. Geosci.*, Pergamon Press, Inc., Tarrytown, NY, USA, v. 33, n. 1, p. 42–61, jan. 2007. ISSN 0098-3004. Disponível em: <http://dx.doi.org/10.1016/j.cageo.2006.05.011>.

MALINOWSKI, E.; ZIMÁNYI, E. *Advanced data warehouse design. From conventional to spatial and temporal applications.* [S.l.]: Berlin: Springer, 2008. xxi + 435 p. ISBN 978-3-540-74404-7/hbk.

MATEUS, R. C. et al. How does the spatial data redundancy affect query performance in geographic data warehouses? *JIDM*, v. 1, n. 3, p. 519–534, 2010.

MOHAN, P. et al. Should sdbms support a join index?: A case study from crimestat. In: *Proceedings of the 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. New York, NY, USA: ACM, 2008. (GIS '08), p. 37:1–37:10. ISBN 978-1-60558-323-5. Disponível em: <http://doi.acm.org/10.1145/1463434.1463481>.

O'NEIL, P. et al. Performance evaluation and benchmarking. In: NAMBIAR, R.; POESS, M. (Ed.). Berlin, Heidelberg: Springer-Verlag, 2009. cap. The Star Schema Benchmark and Augmented Fact Table Indexing, p. 237–252. ISBN 978-3-642-10423-7. Disponível em: <http://dx.doi.org/10.1007/978-3-642-10424-4_17>.

OPEN Geospatial Consortium (OGC). 2014. Disponível em: <http://www.opengeospatial.org/>.

ORACLE Spatial. 2014. Disponível em: <http://docs.oracle.com/cd/B28359_01/app-dev.111/b28400/sdo_intro.htm#SPATL010>.

PAULY, A.; SCHNEIDER, M. Quality Aspects in Spatial Data Mining. In: ____. USA: CRC Press, 2008. cap. Querying Vague Spatial Objects in Databases with VASA, p. 3–14.

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. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0306437909000519>.

POSTGIS - Spatial and Geographic objects for PostgreSQL. 2014. Disponível em: <http://postgis.net/>.

POSTGRESQL: Documentation. 2014. Disponível em: <http://www.postgresql.org/docs/9.4/static/docguide.html>.

PROENÇA, F. R. *Geração de Dados Espaciais Vagos Baseada em Modelos Exatos*. Dissertação (Mestrado) — Universidade Federal de São Carlos, 2013.

SCHNEIDER, M. Uncertainty management for spatial datain databases: Fuzzy spatial data types. In: GÜTING, R.; PAPADIAS, D.; LOCHOVSKY, F. (Ed.). *Advances in Spatial Databases*. Springer Berlin Heidelberg, 1999, (Lecture Notes in Computer Science, v. 1651). p. 330–351. ISBN 978-3-540-66247-1. Disponível em: <http://dx.doi.org/10.1007/3-540-48482-5_20>.

SCHNEIDER, M. Fuzzy topological predicates, their properties, and their integration into query languages. In: *ACM SIGSPATIAL GIS*. New York, NY, USA: ACM, 2001. p. 9–14. ISBN 1-58113-443-6. Disponível em: <http://doi.acm.org/10.1145/512161.512165>.

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. p. 490–515.

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, n. 3, p. 148–170, 2014.

SCHNEIDER, M.; BEHR, T. Topological relationships between complex spatial objects. *ACM Trans. on Database Systems*, v. 31, n. 1, p. 39–81, 2006.

SCHOCKAERT, S. et al. Fuzzy region connection calculus: Representing vague topological information. *Int. J. Approx. Reasoning*, Elsevier Science Inc., New York, NY, USA, v. 48, n. 1, p. 314–331, abr. 2008. ISSN 0888-613X. Disponível em: <http://dx.doi.org/10.1016/j.ijar.2007.10.001>.

SIQUEIRA, T. et al. The impact of spatial data redundancy on solap query performance. *Journal of the Brazilian Computer Society*, Springer-Verlag, v. 15, n. 2, p. 19–34, 2009. ISSN 0104-6500. Disponível em: <http://dx.doi.org/10.1007/BF03194499>.

SIQUEIRA, T. 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. Disponível em: <http://dx.doi.org/10.1007/s10707-013-0186-y>.

SIQUEIRA, T. et al. Benchmarking spatial data warehouses. In: BACH PEDERSEN, T.; MOHANIA, M.; TJOA, A. (Ed.). *Data Warehousing and Knowledge Discovery*. Springer Berlin Heidelberg, 2010, (Lecture Notes in Computer Science, v. 6263). p. 40–51. ISBN 978-3-642-15104-0. Disponível em: <http://dx.doi.org/10.1007/978-3-642-15105-7_4>.

SIQUEIRA, T. et al. Querying vague spatial information in geographic data warehouses. In: GEERTMAN, S.; REINHARDT, W.; TOPPEN, F. (Ed.). *Advancing Geoinformation Science for a Changing World*. Springer Berlin Heidelberg, 2011, (Lecture Notes in Geoinformation and Cartography). p. 379–397. ISBN 978-3-642-19788-8. Disponível em: <http://dx.doi.org/10.1007/978-3-642-19789-5_19>.

SIQUEIRA, T. L. L. et al. Towards vague geographic data warehouses. In: XIAO, N. et al. (Ed.). *Geographic Information Science*. Springer Berlin Heidelberg, 2012, (Lecture Notes in Computer Science, v. 7478). p. 173–186. ISBN 978-3-642-33023-0. Disponível em: <http://dx.doi.org/10.1007/978-3-642-33024-7_13>.

SIQUEIRA, T. L. L. et al. Investigating the effects of spatial data redundancy in query performance over geographical data warehouses. In: *GeoInfo*. [S.l.: s.n.], 2008. p. 1–12.

SOMODEVILLA, M.; PETRY, F. Indexing mechanisms to query fmbrs. In: *Fuzzy Information, 2004. Processing NAFIPS '04. IEEE Annual Meeting of the*. [S.l.: s.n.], 2004. v. 1, p. 198–202 Vol.1.

SPATIAL Extensions for MySQL. 2014. Disponível em: <http://dev.mysql.com/doc/refman/5.0/en/spatial-extensions.html>.

STEFANOVIC, N.; HAN, J.; KOPERSKI, K. Object-based selective materialization for efficient implementation of spatial data cubes. *IEEE Trans. on Knowl. and Data Eng.*, IEEE Educational Activities Department, Piscataway, NJ, USA, v. 12, n. 6, p. 938–958, nov. 2000. ISSN 1041-4347. Disponível em: <http://dx.doi.org/10.1109/69.895803>.

TAO, Y. et al. Indexing multi-dimensional uncertain data with arbitrary probability density functions. In: *In Proc. VLDB*. [S.l.: s.n.], 2005. p. 922–933.

VERSTRAETE, J.; DE TRÉ, G.; HALLEZ, A. Bitmap based structures for the modeling of fuzzy entities. *Control and Cybernetics*, POLISH ACAD SCIENCES SYSTEMS RESEARCH INST, v. 35, n. 1, p. 147–164, 2006. ISSN 0324-8569.

YUEN, S. M. et al. Superseding nearest neighbor search on uncertain spatial databases. *Knowledge and Data Engineering, IEEE Transactions on*, v. 22, n. 7, p. 1041–1055, July 2010. ISSN 1041-4347.

ZADEH, L. A. Fuzzy sets. *Information and Control*, v. 8, p. 338–353, 1965.

ZINN, D.; BOSCH, J.; GERTZ, M. Modeling and querying vague spatial objects using shapelets. In: *Proceedings of the International Conference on Very Large Data Bases*. Vienna, Austria: VLDB Endowment, 2007. (VLDB '07), p. 567–578. ISBN 978-1-59593-649-3. Disponível em: <http://dl.acm.org/citation.cfm?id=1325851.1325917>.