UNIVERSIDADE DE BRASÍLIA
INSTITUTO DE GEOCIÊNCIAS
PROGRAMA DE PÓS-GRADUAÇÃO EM GEOCIÊNCIAS APLICADAS E GEODINÂMICA
ÁREA DE CONCENTRAÇÃO GEOPROCESSAMENTO E ANÁLISE AMBIENTAL

INTEGRIDADE TOPOLÓGICA EM SISTEMAS DE BANCOS DE DADOS ESPACIAIS

Luiz Claudio Oliveira de Andrade

Dissertação de Mestrado do Curso de Pós-Graduação em Geoprocessamento e Análise Ambiental, orientada pelo Dr. Edilson de Souza Bias, aprovada em 04 de outubro de 2018.

UNB
Brasília
2018

PUBLICADO POR:

Universidade de Brasília - UNB
Gabinete do Diretor (GB)
Instituto de Geociências (IG)
CEP 70.910-900
Brasília - DF - Brasil
Tel.:(61) 3107-3300
E-mail: igd@unb.br

UNIVERSIDADE DE BRASÍLIA
INSTITUTO DE GEOCIÊNCIAS
PROGRAMA DE PÓS-GRADUAÇÃO EM GEOCIÊNCIAS APLICADAS E GEODINÂMICA
ÁREA DE CONCENTRAÇÃO GEOPROCESSAMENTO E ANÁLISE AMBIENTAL

INTEGRIDADE TOPOLÓGICA EM SISTEMAS DE BANCOS DE DADOS ESPACIAIS

Luiz Claudio Oliveira de Andrade

Dissertação de Mestrado do Curso de Pós-Graduação em Geoprocessamento e Análise Ambiental, orientada pelo Dr. Edilson de Souza Bias, aprovada em 04 de outubro de 2018.

UNB
Brasília
2018

Dados Internacionais de Catalogação na Publicação (CIP)

Andrade, Luiz Claudio Oliveira de.

Cutter INTEGRIDADE TOPOLÓGICA EM SISTEMAS DE BANCOS DE DADOS ESPACIAIS / Luiz Claudio Oliveira de Andrade. – Brasília : UNB, 2018.

xiii + 91 p. ; ()

Dissertação (**Mestrado em Geoprocessamento e Análise Ambiental**) – Universidade de Brasília, Brasília, 2018.

Orientador : Edilson de Souza Bias.

1. Validação. 2. Topologia 3. Python. 4. PostgreSQL. 5. PostGIS 6. QGIS I. Título.

CDU 000.000

Mestrado em Geoprocessamento e Análise Ambiental

Dr. Edilson de Souza Bias _____

Orientador

Dr. Ricardo Seixas Brites _____

Membro interno

Dra. Linda Soraya Issmael _____

Membro externo

"A vida será mais complicada se você possuir uma curiosidade ativa, além de aumentarem as chances de você entrar em apuros, mas será mais divertida".

Edward Speyer
em *"Seis Caminhos a Partir de Newton"*, 1994

Primeiramente a ***Deus*** *pela sua misericórdia e a minha esposa* ***Waleska*** *e a meus filhos* ***Danilo, Jonas e Lívia****. Sem vocês, minha vida não seria completa.*

RESUMO

Neste trabalho é analisada a eficiência e aplicabilidade de processos de validação topológica desenvolvidos no DSGTools, extensão para QGIS desenvolvida pelo Exército Brasileiro. O DSGTools foi desenvolvido originalmente para atender somente os produtos cartográficos feitos em conformidade com a Especificação Técnica para a Estruturação de Dados Geoespaciais Vetoriais (ET-EDGV) versão 2.1.3, porém, atualmente, é capaz de lidar com qualquer base de dados geoespaciais armazenados em PostgreSQL/PostGIS, além de permitir a produção de dados conforme a recentemente aprovada ET-EDGV versão 3.0. Os processos abordados neste trabalho foram desenvolvidos a partir da determinação de erros topológicos que frequentemente ocorrem durante a etapa de vetorização de um produto cartográfico e foram divididos em dois grupos: processos de identificação e processos de correção. Para desenvolver os processos, foi desenvolvido um *framework* dentro do DSGTools. Esse *framework* permite que os processos de identificação criem registros detalhados de cada problema topológico encontrado e, da mesma forma, permite que os processos de correção os sanem de maneira automatizada. Parte dos processos foi desenvolvida puramente em Python e outra parte em Python com o auxílio de consultas SQL. Os resultados desta pesquisa apresentam um caso de teste no qual um produto cartográfico da DSG, uma Carta Topográfica 1:25.000, feita de acordo com a ET-EDGV 2.1.3, é validado, mostrando os resultados positivos atingidos por meio dos processos presentes no DSGTools.

Palavras-chave: validação. topologia. DSGTools. QGIS. qualidade. PostgreSQL. PostGIS. Python. EDGV. topológica.

TOPOLOGICAL INTEGRITY IN SPATIAL DATABASE SYSTEMS

ABSTRACT

In this work it is analyzed the efficiency and aplicability of topological validation processes developed in DSGTools, a QGIS plugin developed by the Brazilian Army. DSGTools was originally developed to attend only cartographic products made in accordance with the Technical Specification for Vector Geospatial Data Structuring (ET-EDGV) version 2.1.3, but, currently, it is capable to handle with any geospacial database stored in PostgreSQL/PostGIS, as well as allowing data production in accordance with the recently approved ET-EDGV version 3.0. The processes addressed in this work were developed from the determination of topological errors that frequently occur during the vectorization step of a cartographic product and were divided in two sets: identification processes and correction processes. To develop the processes, it was developed a framework within DSGTools. This framework allows the identification processes to create detailed records for each topological error encountered and, in the same way, allows the correction processes to solve them in a automated way. Part of the processes was developed purely in Python and another part was developed in Python with the aid of SQL queries. The results of this work present a test case where a cartographic product provided by DSG, a 1:25,000 Topographic Chart, made in accordance with ET-EDGV 2.1.3, is validated, showing the positive results achieved by the processes present in DSGTools.

Keywords: validation. topology. DSGTools. QGIS. quality. PostgreSQL. PostGIS. Python. EDGV. topological.

LISTA DE FIGURAS

Pág.

SUMÁRIO

Pág.

1 INTRODUÇÃO

A Diretoria de Serviço Geográfico do Exército Brasileiro (DSG), por meio de suas Organizações Militares Diretamente Subordinadas (Centros de Geoinformação), tem mais de 100 anos de experiência em mapeamento topográfico no Brasil. Atualmente, após constantes evoluções tecnológicas, toda a linha de produção cartográfica da DSG é apoiada em Sistemas de Informações Geográficas (SIG) e conta com diversas fases de produção, dentre elas: Aquisição Vetorial, Validação e Edição (CORREIA, 2011).

A Validação é a fase responsável pela garantia da consistência lógica dos dados produzidos, ou seja, é a fase responsável por eliminar erros de integridade topológica (MARAS et al., 2010) nos dados vetoriais produzidos. No caso específico de produtores de geoinformação em nível federal, é nesta fase que são garantidas as restrições topológicas estabelecidas na Especificação Técnica para Estruturação de Dados Geoespaciais Vetoriais versão 2.1.3 (ET-EDGV 2.1.3) (CONCAR, 2010a), padrão de estruturação de dados da Infraestrutura Nacional de Dados Espaciais (INDE) (BRASIL, 2008).

A ET-EDGV é uma das especificações de maior importância para a INDE e, conforme estabelecido em seu Plano de Ação (CONCAR, 2010b) (com fulcro no nº 2 do §1º e no §3º do art. 15, do Capítulo VIII, do Decreto-Lei nº 243, de 28 de fevereiro de 1967), coube ao Exército Brasileiro elaborar a especificação técnica que define os padrões para a aquisição da geometria dos dados geoespaciais vetoriais, atributos correlacionados e os respectivos metadados, que são essenciais à perfeita individualização das instâncias. Neste sentido, foi criada a Especificação Técnica para a Aquisição de Dados Geoespaciais Vetoriais (ET-ADGV) (Exército Brasileiro, 2011) que está disponível, para o público em geral, no sítio `http://www.geoportal.eb.mil.br/portal/inde2?id=140`

Vale ressaltar que qualquer produtor de geoinformação pode fazer uso de tais normas para balizar uma produção cartográfica com qualidade e perfeitamente interoperável com a INDE.

Sendo assim, a Validação é de suma importância para que os dados advindos de qualquer linha de produção cartográfica possam ser de fato plenamente utilizáveis. Isto faz com que estudos na área de garantia da qualidade em dados geoespaciais vetoriais sejam de grande importância, tanto por motivos de ordem prática quanto teórica.

Neste sentido, considerando que a DSG, assim como outros produtores de geoinformação, ainda apoia parte de sua produção cartográfica em soluções proprietárias, particularmente a fase de validação topológica, tem se estudado formas de implantar o uso do *software* livre em sua cadeia produtiva.

Tal estudo é baseado nas diretrizes do Governo Eletrônico, que foram concebidas em decorrência do Decreto de 29 de outubro de 2003 (JARDIM, 2005), particularmente no que diz respeito a diretriz: Utilização do *software* livre como recurso estratégico. E, portanto, é o foco da pesquisa desenvolvida neste trabalho.

No caso particular da DSG, ao se entrar na era do SIG, a fase de validação foi estruturada com uma solução de *software* proprietária. Tal solução é composta de um banco de dados orientado a objetos chamado *GOTHIC* e de um aplicativo de SIG denominado *LAMPS2* (HARDY, 2001), produzidos pela empresa Laser-Scan. O fato da solução ser proprietária impede o completo conhecimento técnico-científico da solução.

Além da impossibilidade de completo conhecimento técnico-científico, a manutenção desta solução é dificultada, pois faz uso de uma linguagem de programação procedural praticamente desconhecida, chamada LULL. A linguagem é tão pouco usada que nem é mencionada no índice TIOBE (BARCELOS et al., 2017), índice que aponta as linguagens de programação mais utilizadas no mundo.

Com base nestas informações, pode-se entender o motivo que fez a DSG levar mais de 10 anos para personalizar a solução *GOTHIC/LAMPS2* para suprir a produção cartográfica com dados estruturados por meio da ET-EDGV 2.1.3. Porém, a ET-EDGV 2.1.3 já foi atualizada pela ET-EDGV 3.0 (CONCAR, 2018) para incluir a possibilidade de mapeamentos em grandes escalas e, com isso, seria necessário customizar novamente a solução *GOTHIC/LAMPS2* para ser capaz de trabalhar com a nova norma. Isto, somado ao fato desta solução ter sido descontinuada pela fabricante, torna contraproducente apoiar uma linha de produção cartográfica na referida solução.

Sendo assim, visando criar uma solução para validação topológica, apoiada em *software* livre, capaz de lidar com a ET-EDGV 3.0 e que pudesse ser utilizada por qualquer produtor de geoinformação vetorial, se chegou ao objetivo geral deste trabalho: *Desenvolver uma solução corporativa para validação topológica apoiada em software livre.*

Tal objetivo geral, para ser cumprido, é composto dos seguintes objetivos específicos:

a) Propor e implementar processos de validação que garantam a integridade topológica da bancos de dados geográficos PostgreSQL/PostGIS;

b) Propor e implementar fluxos de controle de qualidade e de consistência lógica dos dados no ambiente do Banco de Dados;

c) Propor uma metodologia com fluxo de processos para realizar a validação topológica para uso em um contexto corporativo; e

d) Testar e disponibilizar as ferramentas para produção no DSGTools.

O *DSGTools* é uma extensão para o aplicativo livre de SIG *QGIS* que foi desenvolvido pela DSG (DSG, 2015). Esta extensão foi criada com o intuito de prover a todos os seus usuários, de maneira transparente, a criação de bancos de dados, em *PostgreSQL/PostGIS* e *SQLite/Spatialite*, em conformidade com a ET-EDGV 2.1.3. Junto a criação de bancos em conformidade com a ET-EDGV 2.1.3, foram desenvolvidas no DSGTools diversas outras ferramentas que entregam formas simples de lidar com tarefas costumeiramente complexas, otimizando, por tanto, a produção de geoinformação.

Atualmente, o DSGTools já permite a criação de bancos de dados conforme a ET-EDGV 3.0, sendo a única extensão do QGIS capaz de trabalhar com as normas estabelecidas pela CONCAR. Ademais, o uso do QGIS com o DSGTools atende às diretrizes do Governo Eletrônico, estando alinhado com o posicionamento estratégico do Governo Federal.

Sendo assim, como solução, acredita-se que a produção de ferramentas de validação topológica em *software* livre, particularmente no *DSGTools*, permite a criação de uma solução corporativa, capaz de ser usada para garantir a consistência lógica de dados geoespaciais vetoriais, armazenados em Sistemas Gerenciadores de Bancos de Dados, particularmente o *PostgreSQL* com sua extensão espacial *PostGIS*.

Os capítulos restantes desta dissertação estão organizados da seguinte maneira:

- Capítulo 2: O capítulo REVISÃO BIBLIOGRÁFICA aborda os principais conceitos referentes aos erros topológicos que afetam a qualidade de dados geoespaciais vetoriais.

- Capítulo 3: No capítulo MATERIAL E MÉTODOS é feita uma descrição detalhada dos processos desenvolvidos, no DSGTools, no contexto deste trabalho para tratar erros topológicos.

- Capítulo 4: No capítulo RESULTADOS E DISCUSSÃO são apresentados os resultados do trabalho, apresentando o uso das ferramentas de validação desenvolvidas no DSGTools e também é apresentado um caso de teste onde são propostos fluxogramas para validação de camadas que compõem o conceito de cobertura terrestre.

- Capítulo 5: Neste capítulo é feita a conclusão, encerrando o presente trabalho.

2 REVISÃO BIBLIOGRÁFICA

2.1 Dados geoespaciais

Dados geoespaciais podem ser representados digitalmente por meio de duas estruturas de dados, a matricial e a vetorial (CÂMARA; MONTEIRO, 2001).

A estrutura de dados matricial é representada por uma matriz de células que formam uma grade. Cada célula pode ser definida por valores de um sistema binário numérico para guardar características de parte do que está sendo representado (MARAS et al., 2010).

A estrutura vetorial tem como elemento base o ponto. Um conjunto de pontos criam linhas e um conjunto de linhas criam polígonos. Sendo assim, pode-se afirmar que ponto, linha e polígono são as primitivas geométricas de uma estrutura de dados vetorial (OGC, 2010).

Tendo em vista que esta pesquisa está relacionada somente a dados vetoriais, é neste tipo de estrutura que todo o estudo estará apoiado.

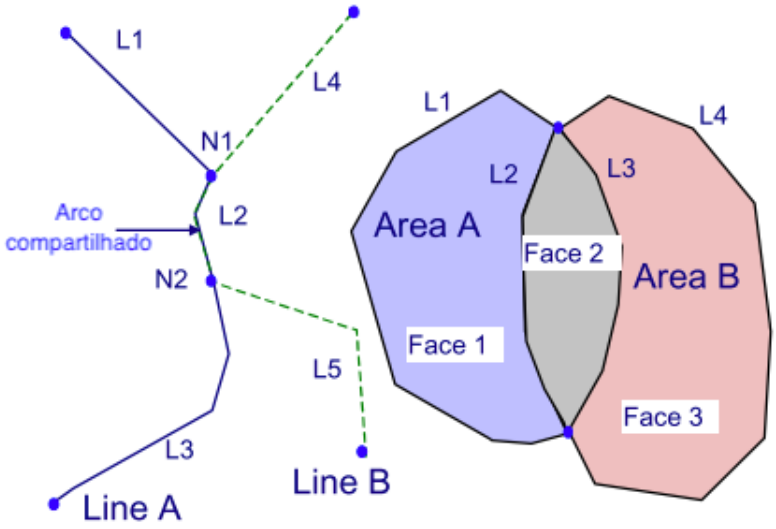
2.2 Dados vetoriais

Os dados vetoriais, quanto a forma de armazenamento, podem ser subdivididos em duas classes: não topológica e topológica.

A forma não topológica, também conhecida como espaguete (MARAS et al., 2010), armazena feições diretamente por meio das primitivas geométricas ponto, linha e polígono. Pontos são elementos zero-dimensionais definidos por uma coordenada. Linhas são elementos uni-dimensionais formados por uma sequência de coordenadas. Polígonos são elementos bi-dimensionais fechados que são formados por linhas que iniciam e terminam no mesmo ponto (MARAS et al., 2010).

A forma topológica armazena as feições por meio de nós, arcos e faces. Pontos são compostos por nós. Um arco é composto com um conjunto de coordenadas que iniciam e terminam em um nó. Face é o espaço bi-dimensional cercado por arcos (MARAS et al., 2010). Na figura 2.1 é possível ver linhas e polígonos estruturados topologicamente

Figura 2.1 - Estrutura topológica



Fonte: Adaptado de (HARDY, 2001)

Independente da forma em que os dados vetoriais estão estruturados, o conhecimento dos relacionamentos entre geometrias é importante para a execução de análises espaciais. Estes relacionamentos são definidos como topológicos (EGENHOFER et al., 1993).

Estes relacionamentos são determinados por meio da análise do interior (A^0), limite (∂A) e exterior (A^-) de uma geometria A com o interior (B^0), limite (∂B) e exterior (B^-) de outra geometria B. Da mesma forma, por meio do interior, exterior e limite de uma geometria se pode definir as primitivas geométricas (SERVIGNE et al., 2000), como segue:

a) Ponto P: $P = \partial A = A^0$, ou seja, para um ponto, seu limite é igual ao seu interior;

b) Linha L: ∂L = dois pontos finais de L e $L^0 = L - \partial L$, ou seja, o seu interior é a linha menos os seus dois pontos finais; e

c) Área A: ∂A = interseção de A com seu exterior A^- e A^0 = união de todos os conjuntos abertos em A.

A figura 2.2 exemplifica tais conceitos para o caso de áreas.

Figura 2.2 - Exmplificação dos conceitos de de Egenhofer

Fonte: (EGENHOFER; HERRING, 1990)

Considerando tais definições, por meio da matriz de 9 interseções dimensionalmente estendida (DE-9IM) (CLEMENTINI; DI FELICE, 1995), que foi feita como uma extensão da matriz de 9 interseções (EGENHOFER et al., 1993), é possível se determinar matematicamente os relacionamentos entre duas geometrias.

Sendo assim, a figura 2.3 exemplifica os 8 relacionamentos que podem ser definidos matematicamente entre duas regiões no espaço bidimensional R^2 (EGENHOFER et al., 1993), a saber:

a) Disjunto;

b) Contém;

c) Dentro;

d) Igual;

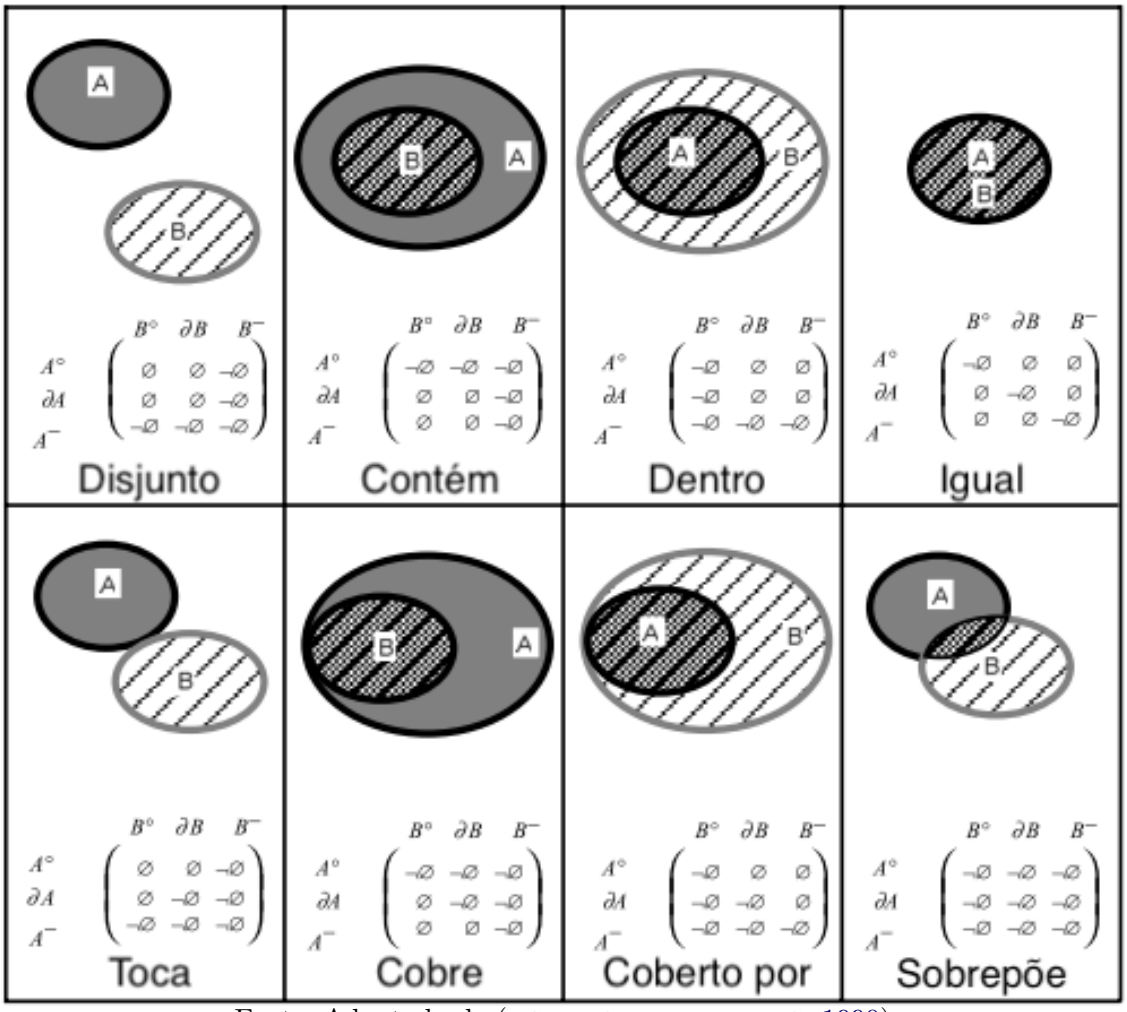
e) Toca;

f) Cobre;

g) Coberto por; e

h) Sobrepõe.

Figura 2.3 - Relacionamentos topológicos no R^2.



Fonte: Adaptado de (EGENHOFER; HERRING, 1990)

2.3 Qualidade de dados vetoriais

O conceito de qualidade de dados geoespaciais está normatizado na ISO 19115:2003 (ISO, 2003), abordando as seguintes classes de qualidade, conforme tabela 2.1

Tabela 2.1 - Classes de qualidade da ISO 19115:2003.

Classe	**Descrição**
Completeza	Presença e ausência de feições, seus atributos e seus relacionamentos.
Consistência lógica	Grau de aderência às regras lógicas da estrutura de dados.
Acurácia posicional	Acurácia de posição das feições
Acurácia temática	Acurácia dos atributos quantitativos e a correção dos atributos não quantitativos e das classificações de feições e seus relacionamentos.
Acurácia temporal	correção de eventos ou sequências ordenadas.

A Consistência Lógica é necessária para garantir a completa qualidade de uma base de dados geoespaciais (HASHEMI; ABBASPOUR, 2015). Este trabalho, devido ao foco na integridade topológica, ficará restrito à esta classe, que se divide nas seguinte subclasses:

a) Consistência conceitual: Aderência às regras do conceitual definido;

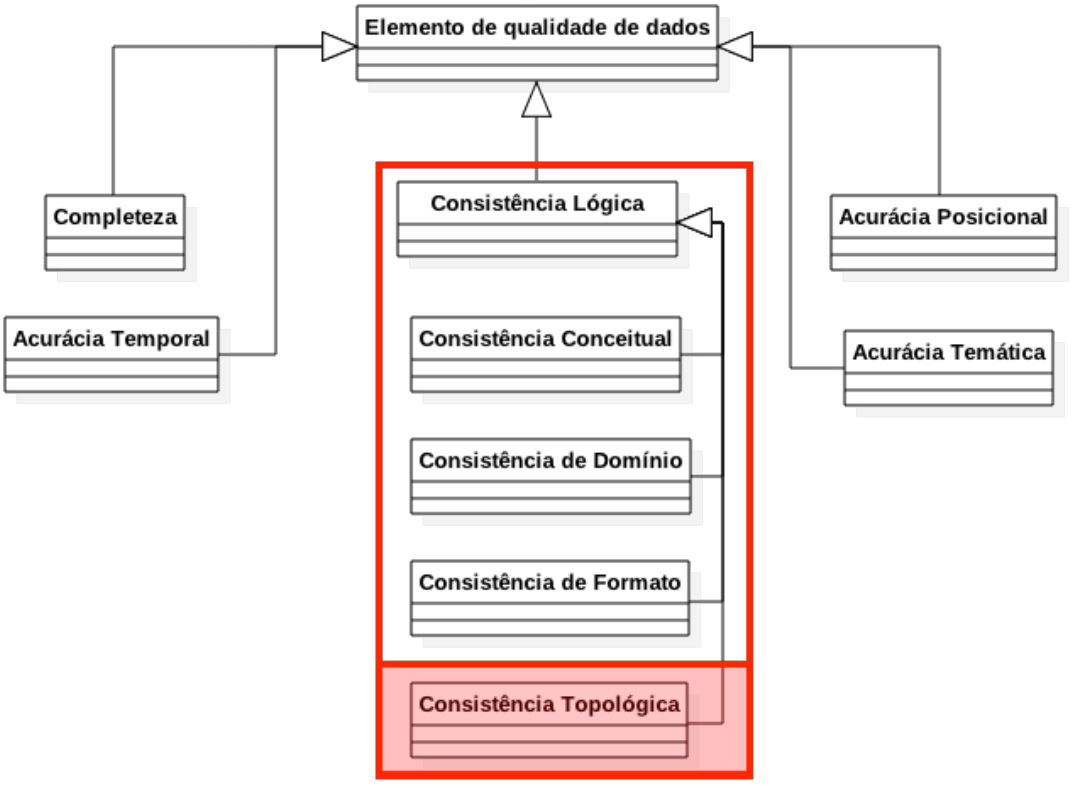
b) Consistência de domínio: Aderência dos valores (atributos) ao domínio estipulado;

c) Consistência de formato: Grau no qual os dados são armazenados de acordo com a estrutura física estipulada; e

d) Consistência topológica: Exatidão das características topológicas de uma base de dados (foco deste trabalho).

A figura 2.4 mostra um extrato das classes e subclasses da qualidade de dados estipuladas pelo ISO 19115:2003, para uma melhor compreensão, sendo que no destaque dela pode-se ver o foco deste trabalho.

Figura 2.4 - Classes e subclasses de qualidade de dados (extrato da ISO)



A consistência topológica é a condição necessária para que as análises estipuladas na DE-9IM sejam possíveis. Dentro do escopo da consistência topológica busca-se garantir que os dados geoespaciais vetoriais não estejam eivados de erros gerais como:

a) Geometrias inválidas;

b) Geometrias Não-simples;

c) Pontas de linhas soltas por excesso ou falta; e

d) Geometrias duplicadas.

Da mesma forma, as regras topológicas estabelecidas no modelo de dados utilizado devem ser obedecidas para evitar erros como:

a) Geometrias multiparte;

b) Geometrias com vãos ou sobreposições;

c) Geometrias com segmentos que formam ângulos muito pequenos;

d) Geometrias com vértices muito próximo a suas arestas;

e) Geometrias com áreas muito pequenas; e

f) Geometrias com comprimento muito pequeno.

2.4 Erros geométricos e topológicos

Dados obtidos por meio de processos de vetorização, que podem ser manuais ou automatizados, necessitam de correções geométricas e topológicas. Tais correções visam garantir a consistência topológica dos dados, para que seja possível fazer uso eficiente dos dados em sistemas de informações geográficas e outras aplicações espaciais (KARAS et al., 2010).

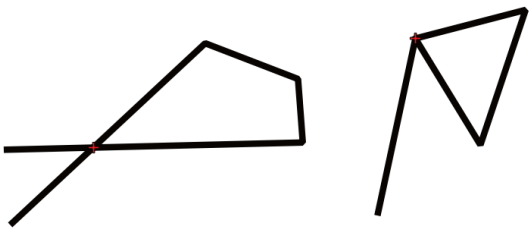
Nos subitens a seguir, os erros são mostrados em maior detalhe.

2.4.1 Geometrias não-simples

De acordo com as especificações da Simple Feature Access (SFS) (OGC, 2010) uma geometria simples não possui pontos anômalos, como auto-interseção ou auto-tangência. A figura 2 mostra exemplos de geometrias do tipo linha que são não-simples.

Tendo em vista a importância de sanar este tipo de erro, Laggner e Orthen (2012) propõem uma forma de reparar geometrias não simples.

Figura 2.5 - Exemplos de linhas não simples



2.4.2 Geometrias inválidas

A definição de geometria válida está ligada a geometrias do tipo linha e polígono. Para geometrias do tipo linha ser válida significa ser simples. Para o caso de geometrias do tipo polígono, geometrias formadas por 1 limite exterior e 0 ou mais limites interiores (OGC, 2010), ser válida significa atender aos seguintes itens conforme elencado por (LEDOUX et al., 2012):

a) os anéis que formam os limites exteriores e interiores devem ser simples;

b) os anéis não devem se cruzar mas podem se intersectar em um ponto tangente;

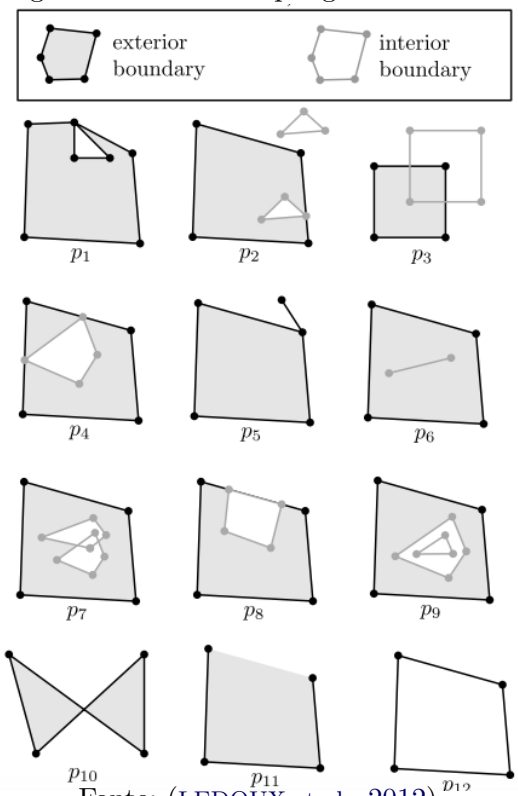
c) um polígono não deve ter linhas de corte ou pontas (externas ou internas);

d) o interior de cada polígono deve ser um conjunto de pontos conectados; e

e) os anéis internos não podem estar fora do anel exterior e nem dentro de outros anéis interiores.

Com isso pode-se chegar a uma lista não exaustiva de polígonos inválidos conforme mostrado na figura 2.6.

Figura 2.6 - Lista de polígonos inválidos



Fonte: (LEDOUX et al., 2012)

Tendo em vista que a maioria das aplicações de geoprocessamento são sensíveis à geometrias inválidas, Laggner e Orthen (2012) propõem uma forma de repará-las.

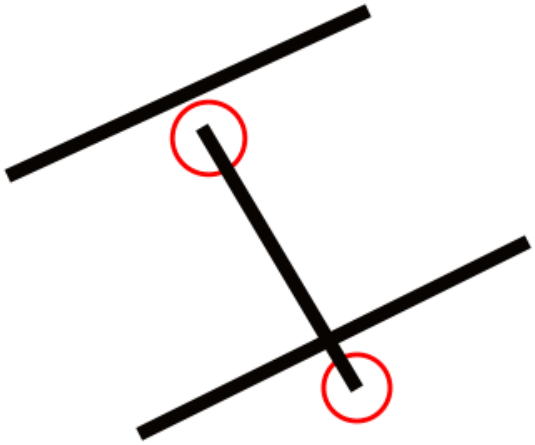
2.4.3 Geometrias duplicadas

Este erro é autoexplicativo, está relacionado a presença de geometrias duplicadas para representar a mesma feição e pode afetar a execução de análises espaciais. Servigne et al. (2000) propõem uma forma de lidar com este tipo de erro por meio da deleção de uma das duplicatas.

2.4.4 Linhas soltas por excesso ou falta

Este tipo de problema não é relacionado a validade ou simplicidade de uma geometria. Em sentido estrito não é um erro mas para algumas aplicações, como roteamento, é impeditivo de uso. Uma aplicação de roteamento necessita de uma rede conectada para que as rotas possam ser calculadas (KARAS et al., 2010). Na figura 2.7 pode ser visto este tipo de erro.

Figura 2.7 - Pontas soltas por excesso e por falta

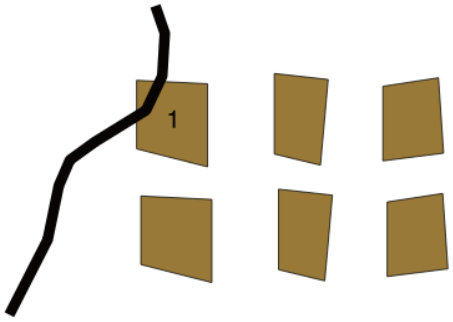


Para sanar este tipo de erro, Karas et al. (2010) propõem uma solução por meio do cálculo da interseção seguida da remoção dos excessos ou do prolongamento, caso necessário.

2.4.5 Geometrias multiparte

Especificações de qualidade podem estipular a proibição da existência de geometrias multiparte (WADEMBERE; OGAO, 2014). Isso ocorre pois geometrias multiparte podem levar a problemas de lentidão na execução de consulta espaciais. Algumas estruturas de indexação espacial são baseadas no conceito de mínimo retângulo envolvente, fazendo com que a primeira fase de uma consulta espacial (determinação de candidatos) seja rápida, porém a segunda parte da consulta (filtragem) pode ser prejudicada em termos de performance computacional devido a complexidade da geometria (NGUYEN, 2009). Sendo assim, no caso de consultas espaciais com geometrias multiparte, é mais eficiente separar a geometria multiparte e recriar a indexação espacial para que, ao invés de um grande retângulo mínimo envolvente que leve a uma geometria complexa, sejam criados diversos retângulos mínimos envolventes que retornam partes menores da geometria original. A figura 2.8 exemplifica o caso citado. Considere que se deseja determinar se uma estrada intersecta uma camada de polígonos. Caso a geometria seja separada, o teste de interseção somente será feito no polígono número 1.

Figura 2.8 - Consulta de interseção de linha com polígono



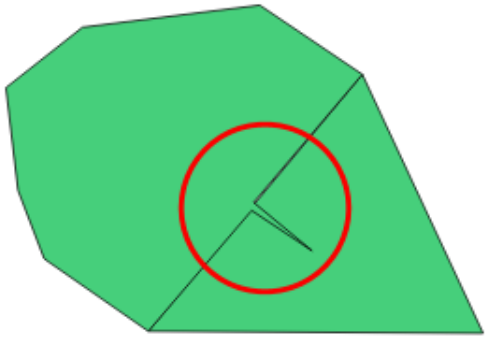
2.4.6 Geometrias com vãos ou sobreposições entre elas

Em diversas situações não se deseja que polígonos de uma camada tenham vãos ou sobreposições entre si (BANSAL, 2011). Por exemplo, a ET-EDGV 2.1.3 estabelece diversos casos em que os polígonos de determinadas classes devem tocar polígonos de outra classe. Da mesma forma ao se compor o mapeamento de uma cobertura terrestre não se deseja que haja vãos entre os polígonos que a compõem. Segundo Dangermond (apud MARAS et al., 2010), estes tipo de erro são normalmente decorrentes do processo de digitalização. Erros desse tipo afetam resultados de análises espaciais, principalmente as relacionadas com o uso do predicado topológico *Toca*. Sendo assim, Laggner e Orthen (2012) propõem uma forma de reparar este erro.

2.4.7 Geometrias com ângulos pequenos

A presença de ângulos pequenos em linhas e polígonos também deve evitada. Tal fato fere o conceito de geometria limpa (OOSTEROM et al., 2005). Segmentos que formam ângulos pequenos são indicativos de vértices em excesso que podem ser removidos sem prejudicar o formato geral da geometria. Este tipo de vértice, cujos segmentos anterior e próximo formam um ângulo abaixo de um limite tolerado, ao ser removido pode inclusive auxiliar na resolução de problemas de relacionamento topológico entre feições vizinhas. A figura 2.9 mostra um exemplo deste caso. Vale ressaltar que o mesmo conceito pode ser estendido para feições lineares.

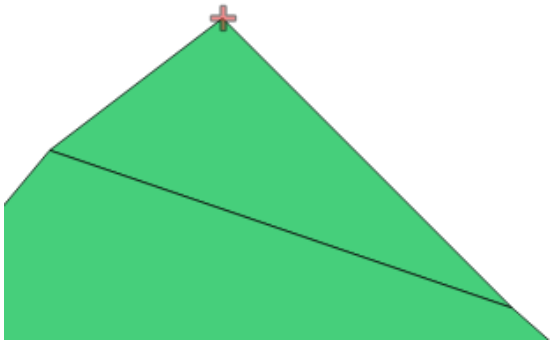
Figura 2.9 - Polígono com segmentos que formam ângulo abaixo do limite



2.4.8 Geometrias com vértices próximos a suas arestas

Este tipo de problema também está intimamente ligado ao conceito de geometria limpa (OOSTEROM et al., 2005). De acordo com a escala do produto deve haver uma filtragem para remoção de vértices próximos a arestas de acordo com um valor de tolerância. A figura 2.10 mostra este caso, onde o vértice em destaque está a uma distância menor que a distância limite da aresta mostrada. Vale ressaltar que o mesmo conceito pode ser usado para feições lineares.

Figura 2.10 - Vértice próximo a aresta



Uma solução para este tipo de problema pode ser baseada na remoção dos vértices em excesso. Tais algoritmos são classificados como algoritmos de simplificação. Douglas e Peucker (1973) propuseram um dos mais conhecidos algoritmos de simplificação, sendo largamente usado até hoje.

2.4.9 Geometrias com áreas pequenas ou comprimentos pequenos

Este tipo de problema também está ligado aos critérios de aquisição das geometrias. Normalmente os valores mínimos de área e comprimento estão relacionados à escala do produto que se está produzindo. Sendo assim, de acordo com o valor mínimo estabelecido, deve ser possível identificar estas geometrias que possuam área ou comprimento abaixo do valor limite para correção, que pode ser um processo de remoção ou de dissolução de geometria menor na geometria maior (desde que haja uma relação de interseção entre elas).

2.5 Fragmentação do dado

Até o presente momento, foram apresentados os erros topológicos que comprometem a qualidade de uma base de dados vetorial. Porém, cabe ressaltar que outros fatores devem ser levados em conta na garantia de integridade topológica. Fatores estes ligados à forma como o dado é estruturado e armazenado (HOOP et al., 1993), que fazem com que soluções voltadas para a garantia de integridade topológica devam ser capazes de lidar com diferentes formas de estruturar o dado.

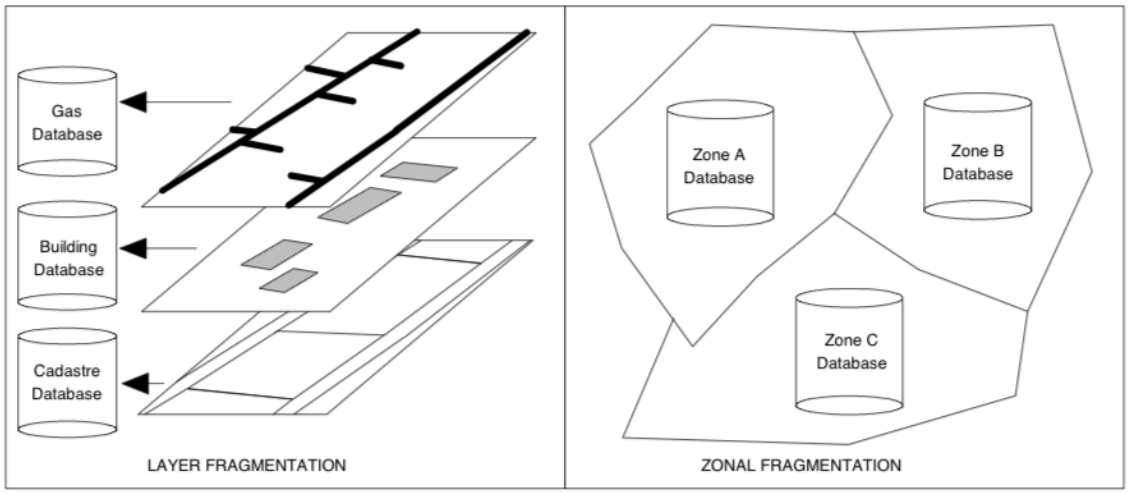
Neste contexto, dados geoespaciais podem ser armazenados em bases de dados por meio de dois tipos de fragmentação (LAURINI, 1998), a saber:

a) Fragmentação em camadas; e

b) Fragmentação zonal.

A fragmentação em camadas é a forma de divisão do dado em camadas de informação diferentes e tem por objetivo uma estruturação temática do dado. A fragmentação zonal é a forma de divisão do dado em regiões, comumente ligada a competência de mapeamento territorial. Como exemplo disso, pode-se citar o caso de órgãos de mapeamento estatal num contexto federal. A figura 2.11 exemplifica visualmente ambas as fragmentações.

Figura 2.11 - Fragmentação em camadas e fragmentação zonal



Fonte: (LAURINI, 1998)

A fragmentação em camadas é um desafio à integridade topológica, pois há casos em que regras topológicas devem ser definidas entre camadas diferentes. Como exemplo disso, pode-se citar as camadas de vegetação da ET-EDGV 2.1.3, que, por uma escolha temática, fragmenta tipos diferentes de vegetação em camadas separadas, mas que, ao mesmo tempo, estipula que não deve haver vãos nem sobreposições entre elas.

Atualmente, quando se fala em *software* livre, é possível lidar com a fragmentação em camadas, mas não é algo simples ao usuário. Soluções proprietárias permitem criar regras entre camadas separadas, mas sempre utilizando a estrutura interna estipulada que, costumeiramente, não é interoperável. Sendo assim, a solução desenvolvida neste trabalho buscou, também, lidar com este tipo de fragmentação.

3 MATERIAL E MÉTODOS

3.1 Material

Os aplicativos utilizados neste trabalho são os seguintes:

a) Editor de código Visual Studio Code versão 1.20 para o edição dos códigos Python desenvolvidos neste trabalho;

b) Aplicativo de SIG QGIS versão 2.18;

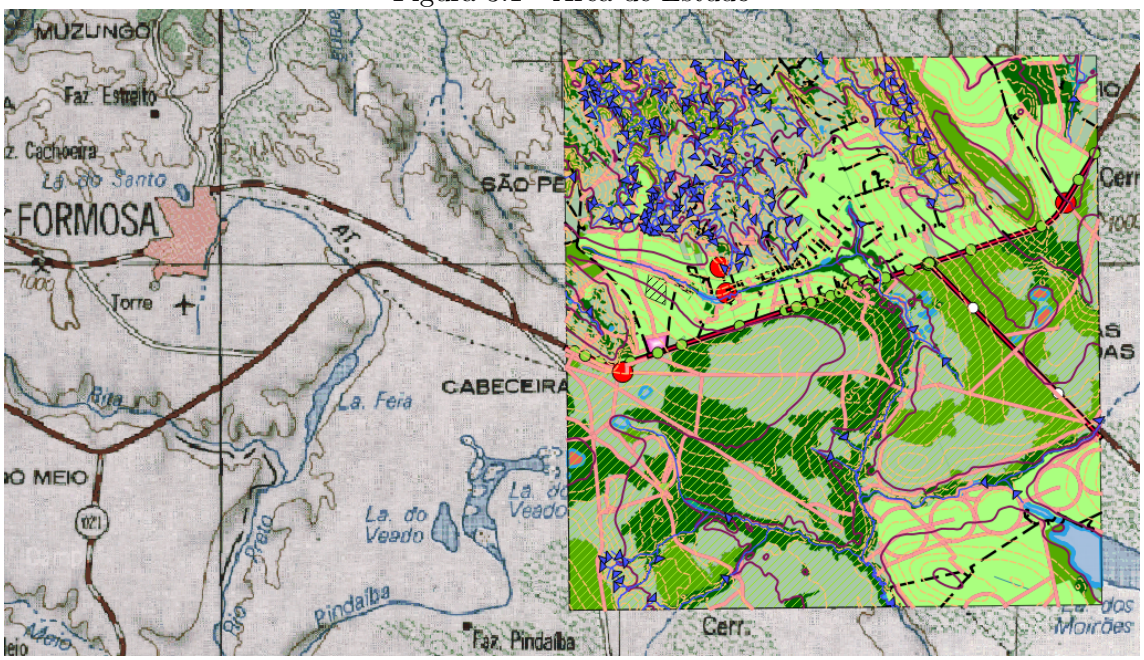
c) SGBD PostgreSQL versão 9.6;

d) Extensão Espacial do PostgreSQL, PostGIS versão 2.1; e

e) DSGTools, versão de desenvolvimento (DSG, 2015).

A área de estudo é referente à Carta Topográfica de MI 2216-2-NO, cujo índice de nomenclaura é SD-23-Y-C-V-2-NO. É um produto vetorial na escala 1:25.000, a leste da cidade de Formosa-GO, obtido por meio do 2º Centro de Geoinformação, Organização Militar Diretamente Subordinada à DSG (OMDS/DSG). Possui uma área de $185km^2$ que engloba a parte norte do Campo de Instrução de Formosa. A figura 3.1 mostra a situação do MI 2216-2-NO.

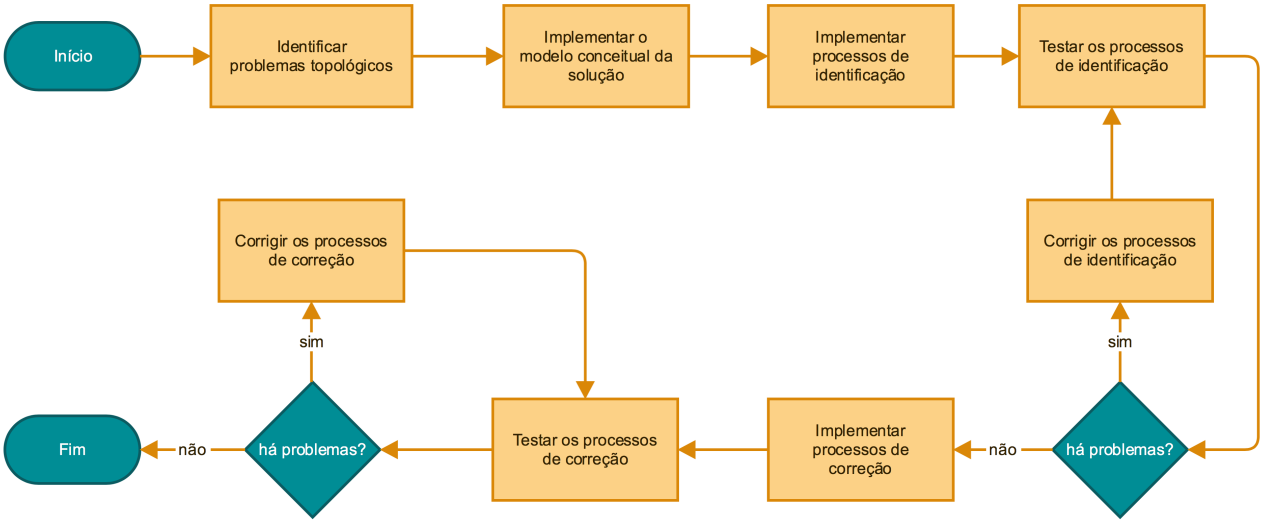
Figura 3.1 - Área de Estudo



3.2 Métodos

3.2.1 Fluxograma de Desenvolvimento

Figura 3.2 - Fluxo metodológico do trabalho



O desenvolvimento do presente trabalho se iniciou com a definição de quais problemas topológicos deveriam ser solucionados no DSGTools para que a extensão pudesse se tornar uma solução corporativa capaz de garantir a integridade topológica de dados vetoriais armazenados em PostgreSQL/PostGIS. Tais problemas são determinados pela ocorrência dos erros geométricos e topológicos listados no capítulo 2, na seção 2.4.

Neste contexto, foi definido e implementado, no DSGTools, um modelo conceitual para o *framework* de validação que foi desenvolvido. Este modelo é a base na qual os processos de validação do plugin.

Em seguida, foi definido que os processos de validação implementados no framework deveriam ser divididos em dois grupos, a saber:

a) Processos de identificação; e

b) Processos de correção.

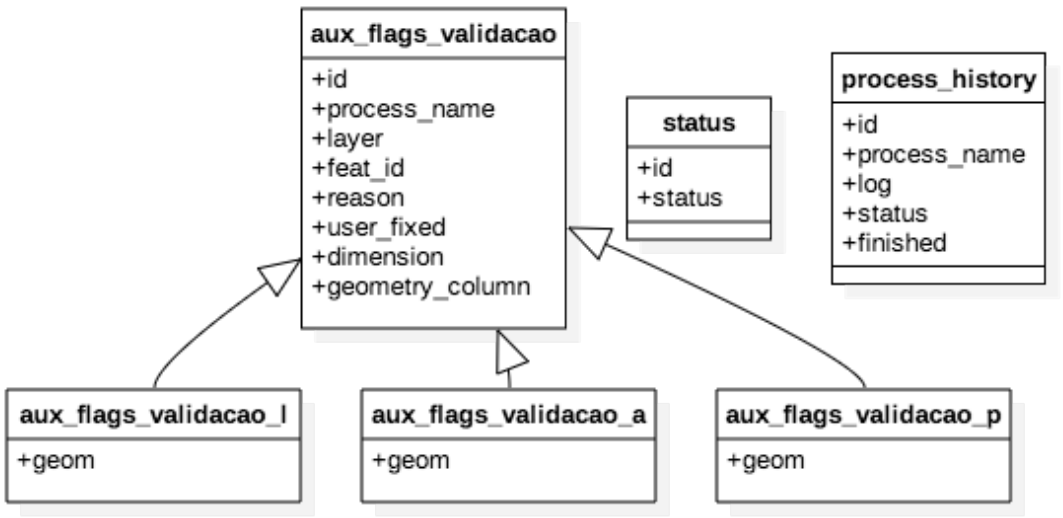
Para cada grupo se adotou o procedimento de implementação, testes e correções.

Ao fim de todos os testes, o conjunto de ferramentas de validação do DSGTools foi terminado, finalizando o presente fluxograma. Na figura 3.2 é possível ver o fluxograma para uma melhor compreensão.

3.2.2 Modelo Conceitual da Solução

Com o objetivo de implementar processos capazes de solucionar os problemas topológicos apresentados na seção Erros geométricos e topológicos, foi criado um framework que foi implementado no DSGTools, adicionando, desta forma, um conjunto de ferramentas de validação ao conjunto de ferramentas previamente existente. Este *framework* é apoiado em um esquema do SGBD PostgreSQL que é criado na primeira vez que um banco é selecionado para ser validado topologicamente no DSGTools. O modelo conceitual apresentado na figura 3.3 foi feito a partir do código existente no DSGTools, durante a execução desta pesquisa.

Figura 3.3 - Modelo conceitual do framework de validação



As tabelas aux_flags_validacao_p, aux_flags_validacao_l e aux_flags_validacao_a herdam de aux_flags_validacao e são as tabelas responsáveis pelo registro dos problemas de tipo ponto, linha e área identificados, respectivamente. O registro contempla o nome do processo que levantou a flag, o nome da camada que estava sendo verificada pelo processo, o id da feição com o problema identificado, o motivo

do problema topológico, se o problema já foi sanado, a dimensão da flag e a coluna geométrica testada. A coluna geométrica é guardada pois uma tabela pode possuir mais de uma coluna geométrica.

A tabela status guarda os possíveis estados de um processo: Não rodado ainda (*not yet ran*), terminado (*finished*), falhou (*failed*), rodando (*running*) e terminado com flags (*finished with flags*). Por fim, a tabela process_history registra todo o histórico de execução de processos, guardando o nome do processo executado, somado a seu registro pós execução, situação pós execução e data-hora do término de execução do processo.

3.2.3 Processos de Validação

Os processos de validação desenvolvidos no DSGTools foram divididos em dois grupos: processos de identificação e processos de correção. Os processos de identificação são os responsáveis por identificar os problemas topológicos criando registros para os problemas encontrados. Os processos de correção são os processos que objetivam corrigir automaticamente os problemas topológicos encontrados. Em termos arquiteturais, todos os processos herdam de um processo chamado *ValidationProcess*. Este processo é o responsável por fazer, entre outras, as seguintes tarefas:

a) guardar registros de problemas;

b) atualizar a camada original após a execução de processos de correção;

c) carregar as camadas no QGIS que serão processadas antes da execução; e

d) registrar o tempo de execução de cada processo.

Nos itens a seguir os processos de identificação e correção serão abordados com maiores detalhes.

3.2.3.1 Processos de Identificação

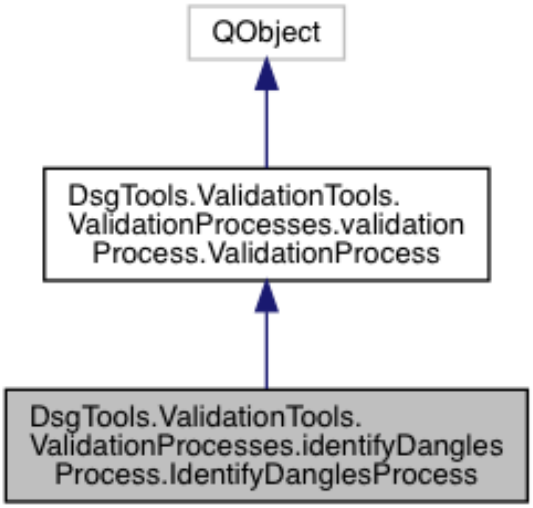
Os processos de identificação são responsáveis por identificar e registrar os problemas encontrados. O registro dos problemas é feito por meio de flags que são armazenadas nas três tabelas que herdam de *aux_flags_validacao*.

3.2.3.1.1 Processo de Identificação de Pontas Soltas

Este processo é responsável por identificar pontas soltas em feições lineares, sendo útil para identificar problemas de conectividade que podem ser impeditivos para

algumas aplicações, como roteamento. A base deste método é um algoritmo que percorre os pontos iniciais e finais de cada geometria e identifica em quantas geometrias estes pontos estão presentes. Se apenas uma geometria possuir um dado ponto inicial ou final, este ponto é classificado como ponta solta.

Figura 3.4 - Diagrama de hierarquia do processo Identificar pontas soltas



Este processo é baseado unicamente em um algoritmo Python que é implementado em seu método de execução. O pseudoalgoritmo 1 mostra a base deste processo.

Ambos foram desenvolvidos no contexto da pesquisa.

Algoritmo 1: Identificador de pontas soltas

```
endVerticesDict: mapa chave/valor
início
    para cada feição ∈ camada faça
        geom ← geometria da feição
        se geometria é multiparte então
            para cada parte ∈ partes de geom faça
                preencher endVerticesDict
            fim
        fim
        senão
            preencher endVerticesDict
        fim
    fim
    para cada ponto ∈ lista de chaves de endVerticesDict faça
        se tamanho da lista de ids de feicao de ponto > 1 então
            continuar
        fim
        geometry geometria do ponto em wkb
        featid único id presente na lista de ponto
        criar flag com geometry e featid pois ponto é uma ponta solta
    fim
fim
```

O preenchimento de *endVerticesDict* é feito por meio do algoritmo 2.

Algoritmo 2: PREENCHIMENTO DE DADOS NO ENDVERTICESDICT

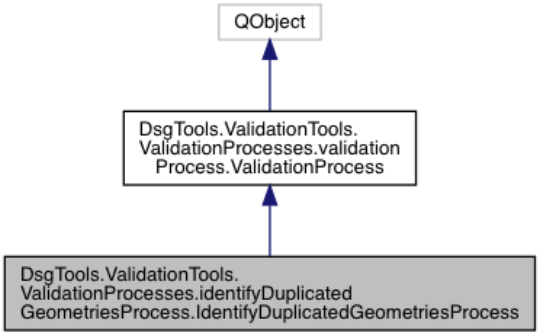
Entrada: *feição*

```
1  geom ← geometria da feição
2  startPoint ← obter ponto inicial de geom
3  endPoint ← obter ponto final de geom
4  se startPoint não estiver na lista de chaves de endVerticesDict então
5  |  adicionar startPoint em endVerticesDict
6  |  criar uma lista vazia de valores para startPoint
7  fim
8  adicionar o id da feicao na lista de valores de startPoint
9  se endPoint não estiver na lista de chaves de endVerticesDict então
10 |  adicionar endPoint em endVerticesDict
11 |  criar uma lista vazia de valores para endPoint
12 fim
13 adicionar o id da feicao na lista de valores de endPoint
```

3.2.3.1.2 Identificar Geometrias Duplicadas

Este processo é responsável por identificar geometrias duplicadas, sendo executado por meio de uma consulta SQL que é rodada diretamente nas tabelas que estão sendo testadas. Tal processo já estava presente no DSGTools.

Figura 3.5 - Diagrama de hierarquia do processo Identificar geometrias duplicadas



A SQL executada é a apresentada na consulta 3.1:

Consulta 3.1 - Identificar geometrias duplicadas

```
select * from
```

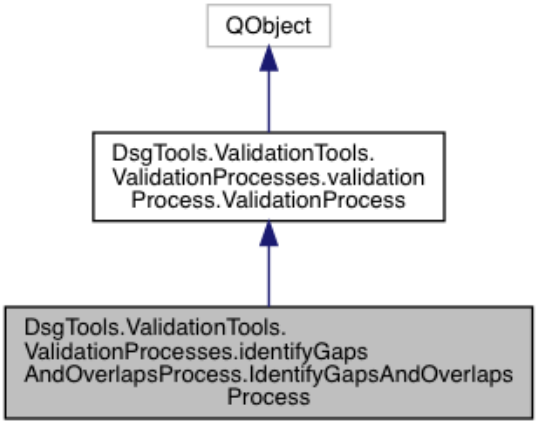
```
(SELECT coluna_chave,
ROW_NUMBER()
OVER(PARTITION BY coluna_geometrica ORDER BY coluna_chave asc) AS Row,
coluna_geometrica FROM ONLY esquema.tabela
) dups
where dups.Row > 1
```

Esta consulta cria grupos de geometrias iguais e para cada linha do grupo é atribuído um número de linha que se inicia em 1 (um). Sendo assim, ao final da consulta haverá grupos de geometrias duplicadas, linhas do número 2 (dois) em diante, que podem ser selecionadas ao se buscar linhas com número maior que um. Das linhas com número maior que um são obtidos os identificadores (coluna_chave) e as geometrias para que sejam criadas as flags.

3.2.3.1.3 Identificar Vãos e Sobreposições

Este processo é capaz de identificar vãos e sobreposições ao se analisar um conjunto de camadas do tipo polígono com uma camada de referência (geralmente a moldura que engloba o conjunto de camadas). Inicialmente, o conjunto de camadas que será analisado passa por um processo de unificação para que todas suas geometrias sejam analisadas como se pertencessem a uma única camada.

Figura 3.6 - Diagrama de hierarquia do processo Identificar Vãos e Sobreposições



A busca por vãos e sobreposições é feito por meio de duas consultas SQL, que foram desenvolvidas no contexto desta pesquisa. Uma para buscar as regiões que representam vãos e outra para buscar as áreas de sobreposição.

Os vãos com a camada de referência são determinados por meio da SQL apresentada

na consulta 3.2:

Consulta 3.2 - Identificar vão com a referência

```
select (ST_Dump(ST_SymDifference(a.geom, b.geom))).geom from
(select ST_Union(coluna_geometrica) as geom from esquema.tabela_ref) as a,
(select ST_Union(geom) as geom from validation.coverage_temp) as b
```

Esta consulta calcula a diferença simétrica entre a união das geometrias que compõem a camada de referência e a união das geometrias da camada unificada. Como o resultado tende a ser uma geometria multiparte, é feito o uso do *ST_Dump* para separar cada vão tenha um registro individualizado.

As sobreposições são determinadas por meio da SQL apresentada na consulta 3.3 que faz uso da função *ST_GeoTableSummary* presente no *PostGIS Addons* (RACINE, 2013):

Consulta 3.3 - Identificar sobreposições

```
select (ST_Dump(foo.geom)).geom as geom from
(select
(ST_GeoTableSummary(
'esquema', 'tabela_unificada', 'coluna_geometrica', 'coluna_chave', 10, 'S3')
).geom) as foo
where ST_IsEmpty(foo.geom) = 'f'
```

Da mesma forma, os vão são determinados por meio da SQL apresentada na consulta 3.4 que faz uso da função *ST_GeoTableSummary*.

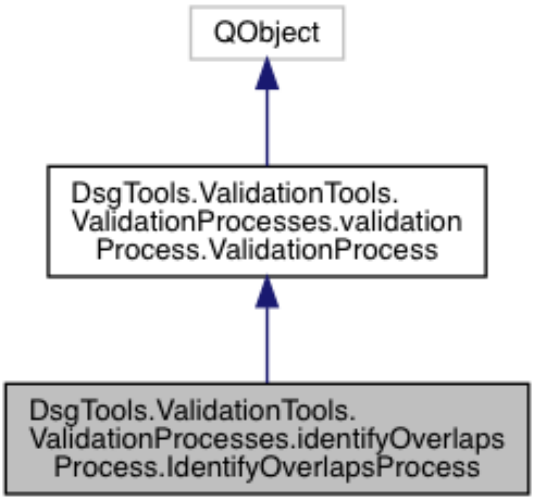
Consulta 3.4 - Identificar vãos

```
select (ST_Dump(foo.geom)).geom as geom from
(select
(ST_GeoTableSummary(
'esquema', 'tabela_unificada', 'coluna_geometrica', 'coluna_chave', 10, 'S4')
).geom) as foo
where ST_IsEmpty(foo.geom) = 'f'
```

3.2.3.1.4 Identificar Sobreposições

Este processo é responsável apenas pela determinação de sobreposições e uma dada camada. Pode ser interpretado com um caso simplificado do processo Identificar Vãos e Sobreposições, tendo seu funcionamento executado unicamente por meio da SQL apresentada na consulta 3.3, que foi desenvolvida no contexto desta pesquisa.

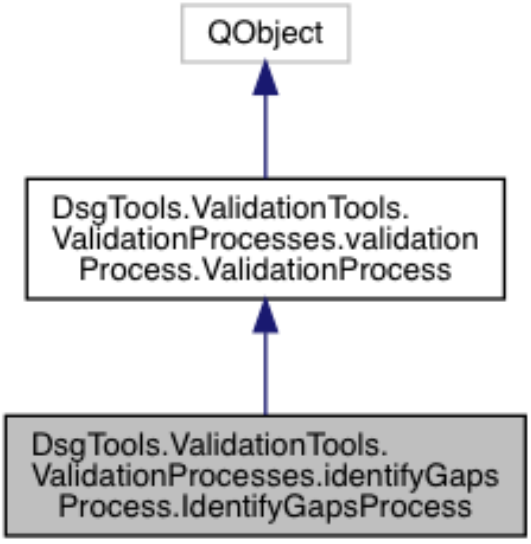
Figura 3.7 - Diagrama de hierarquia do processo Identificar vãos



3.2.3.1.5 Identificar Vãos

Este processo é responsável apenas pela determinação de vãos e uma dada camada. Pode ser interpretado com um caso simplificado do processo Identificar vãos e sobreposições, tendo seu funcionamento executado unicamente por meio da SQL apresentada na consulta 3.4, que foi desenvolvida no contexto desta pesquisa.

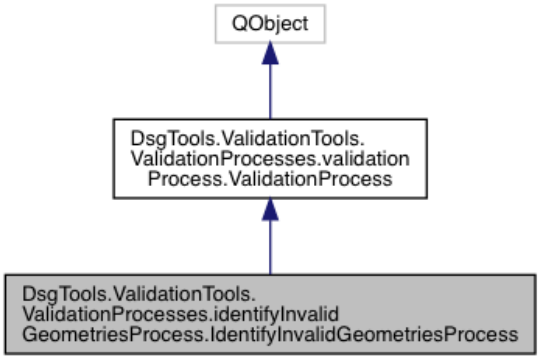
Figura 3.8 - Diagrama de hierarquia do processo Identificar vãos



3.2.3.1.6 Identificar Geometrias Inválidas

Este processo identifica feições que possuem geometrias que ferem o conceito de validade apresentado na seção 2.4.2. Tal processo já estava presente no DSGTools.

Figura 3.9 - Diagrama de hierarquia do processo Identificar geometrias inválidas



Este processo é executado por meio da SQL apresentada na consulta 3.5 que faz uso das funções *ST_IsValid*, que verifica a validade de uma geometria, e da função *ST_IsValidDetail*, que detalha a invalidade fornecendo o motivo e a localização do problema, a saber:

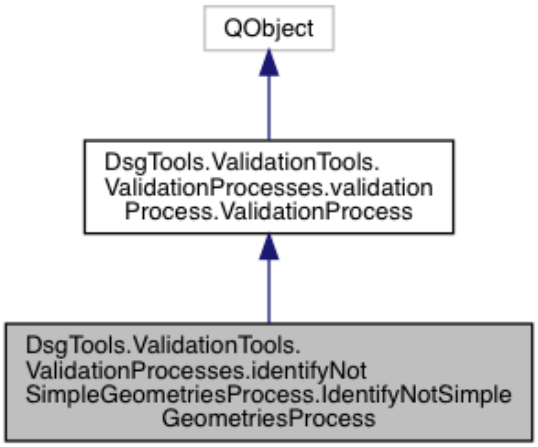
Consulta 3.5 - Identificar geometrias inválidas

```
select distinct
        f."coluna_chave" as "coluna_chave",
        (reason(ST_IsValidDetail(f."coluna_geometrica",0))),
        (location(ST_IsValidDetail(f."coluna_geometrica",0))) as "coluna_geometrica
            "
from
        (select "coluna_chave", "coluna_geometrica" from only "esquema"."tabela"
where
        ST_IsValid("coluna_geometrica") = 'f') as f
```

3.2.3.1.7 Identificar Geometrias Não Simples

Este processo identifica feições que possuem geometrias que ferem o conceito de simplicidade. Tal processo já estava presente no DSGTools.

Figura 3.10 - Diagrama de hierarquia do processo Identificar geometrias não simples



Este processo é executado por meio da SQL apresentada na consulta 3.6:

Consulta 3.6 - Identificar geometrias não simples

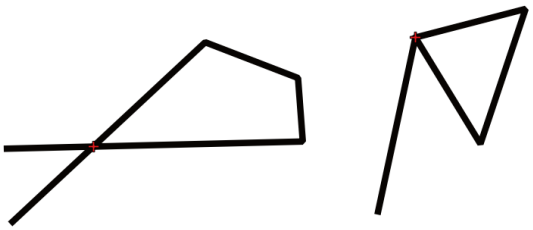
```
select
foo."coluna_chave" as "coluna_chave",
ST_MULTI(st_startpoint(foo."coluna_geometrica")) as "coluna_geometrica"
from
(select
"coluna_chave" as "coluna_chave",
(ST_Dump
(ST_Node(
        ST_SetSRID(ST_MakeValid( "coluna_geometrica"),ST_SRID( "coluna_geometrica")
            )))
).geom as "coluna_geometrica"
from "esquema"."tabela"
where ST_IsSimple("coluna_geometrica") = 'f'
) as foo
where
st_equals(
st_startpoint(foo."coluna_geometrica"),st_endpoint(foo."coluna_geometrica")
)
```

Esta consulta se inicia com a determinação das geometrias não simples da tabela que está sob análise. Esta etapa é feita por meio do seguinte trecho da consulta:

```
where ST_IsSimple("coluna_geometrica") = 'f'
```

Isto faz com que geometrias como as mostradas na figura 3.11 sejam selecionadas.

Figura 3.11 - Geometrias não simples

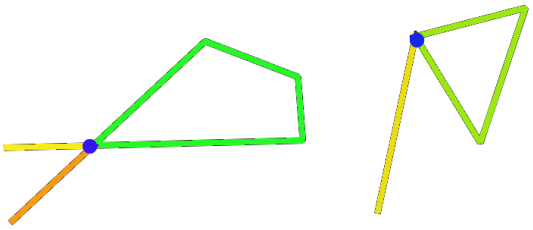


As geometrias selecionadas com a cláusula *where* acima são as que devem ser analisadas. Posteriormente é necessário determinar a localização do problema que leva a geometria a ser classificada como não simples. Sendo assim, inicialmente, é necessário tornar a geometria válida. Isso é feito por meio do seguinte trecho da consulta 6:

```
ST_SetSRID(ST_MakeValid( "coluna_geometrica"),ST_SRID( "coluna_geometrica"))
```

Este trecho gera uma geometria multiparte válida com o mesmo SRID (código de sistema de referência espacial) que a geometria original (a geometria não simples de entrada). Em seguida, na geometria validada são criados nós, porém permanecendo como uma geometria multiparte. Finalizando, as partes da geometria multiparte são separadas com o uso do *ST_Dump*. Isto é feito para que seja possível analisar as partes individualmente. Gerando o que se pode ver na figura 3.12.

Figura 3.12 - Geometrias simples e separadas



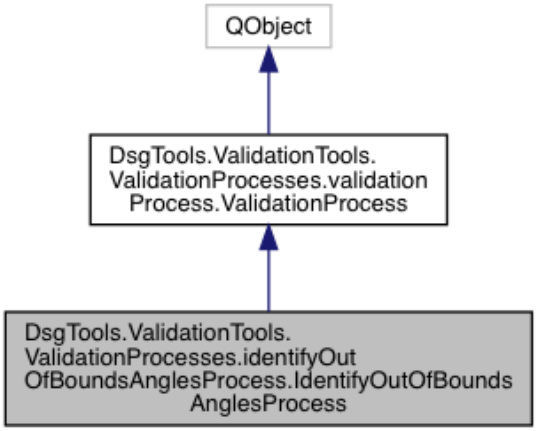
Finalmente, a determinação da localização do problema que torna a geometria não simples é feito por meio da seleção do ponto inicial das partes que possuem ponto inicial igual ao ponto final (partes verdes da figura 3.12), tendo em vista que isto

determina um ponto de auto-interseção na geometria original, devendo, portanto, ser registrado como uma *flag*.

3.2.3.1.8 Identificar Ângulos Fora dos Limites

Este processo é por identificar ângulos fora de um limites estabelecido, tanto para camadas do tipo polígono como para camadas do tipo linha. O processo itera sobre os todos pares de segmentos consecutivos possíveis calculando o ângulo formado pelos pares. Tal processo já estava presente no DSGTools.

Figura 3.13 - Diagrama de hierarquia do processo Identificar geometrias não simples



Este processo é baseado em uma consulta SQL para camadas do tipo linha que segue os seguintes passos:

a) Inicialmente, esta consulta seleciona apenas as linhas que possuem mais de 2 vértices, isto é executado por meio do uso do seguinte trecho de consulta, chamado de A:

```
(SELECT
coluna_chave as coluna_chave,
(ST_Dump(coluna_geometrica)).geom as coluna_geometrica

FROM only
esquema.tabela
) AS linestrings
WHERE ST_NPoints(linestrings.coluna_geometrica) > 2
```

b) Posteriormente, são criados pontos referentes a pares de segmentos adjacentes, *pt1*, *anchor* e *pt2*, por todo o comprimento da linha selecionada que são selecionados junto ao id da linha. Isto é feito por meio do seguinte trecho da consulta, chamado de B:

```
SELECT
ST_PointN(
coluna_geometrica, generate_series(1, ST_Npoints(coluna_geometrica)-2)
) as pt1,
ST_PointN(
coluna_geometrica, generate_series(2, ST_Npoints(coluna_geometrica)-1)
) as anchor,
ST_PointN(
coluna_geometrica, generate_series(3, ST_Npoints(coluna_geometrica))
) as pt2,
linestrings.coluna_chave as coluna_chave

FROM A
```

c) Em seguida, são calculados os ângulos que os segmentos adjacentes formam com o ponto *anchor*. Isto é feito no seguinte trecho da consulta que cria *result*:

```
WITH result AS (SELECT points.coluna_chave,
points.anchor,
(degrees(ST_Azimuth(points. anchor, points.pt1) -
ST_Azimuth(points.anchor, points.pt2)
)::decimal + 360
) % 360 as angle

FROM
(B) as points)
```

d) Finalmente, é aplicada uma cláusula *where* final para que os ângulos que estejam fora do limite sejam selecionados juntamente com o ponto anchor e o com o id da linha correspondente. Isto é feito por meio do seguinte trecho da consulta:

```
select distinct
```

```
coluna_chave,
anchor,
angle
from result
where
(result.angle % 360) <
lista_ids or result.angle >
(360.0 - (lista_ids % 360.0))
```

Para as camadas do tipo polígono, foram implementadas as seguintes mudanças:

a) O conceito aplicado para as camadas do tipo polígono é o mesmo. Vale ressaltar que a operação inicial, trecho de consulta A, deve ser modificada para que os anéis do polígono sejam analisados. Isto é feito com alteração:

```
(SELECT
coluna_chave as coluna_chave,
(ST_Dump(
ST_Boundary(ST_ForceRHR((ST_Dump(coluna_geometrica)).geom)))
).geom as coluna_geometrica

FROM only
esquema.tabela
) AS linestrings
WHERE ST_NPoints(linestrings.coluna_geometrica) > 2
```

b) Por fim, a última alteração que deve ser feita na consulta é na definição dos pontos *pt1*, *anchor* e *pt2*, trecho de consulta B, que devem levar em consideração que os anéis de um polígono são uma linha fechada e simples e, portanto, o ponto inicial é igual ao ponto final. Tal fato se reflete na seguinte alteração:

```
SELECT
ST_PointN(
coluna_geometrica, generate_series(1, ST_Npoints(coluna_geometrica)-2)
) as pt1,
ST_PointN(
coluna_geometrica, generate_series(2, ST_Npoints(coluna_geometrica)-1)
```

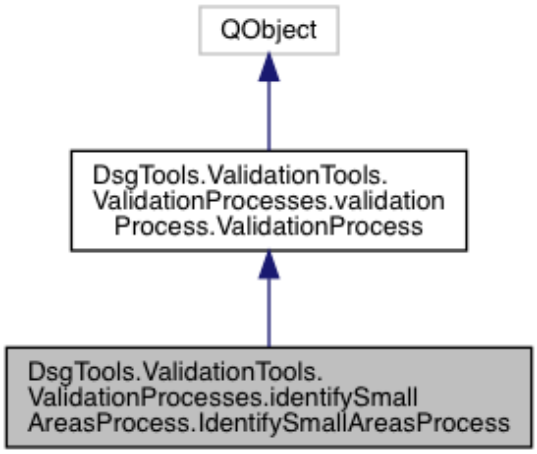
```
) as anchor,
ST_PointN(
coluna_geometrica, generate_series(3, ST_Npoints(coluna_geometrica))
) as pt2,
linestrings.coluna_chave as coluna_chave

FROM A
```

Estas alterações permitem que a consulta seja efetuada em camada do tipo polígono.

3.2.3.1.9 Identificar Áreas Pequenas

Este processo é responsável por identificar feições que possuem polígonos com áreas abaixo de uma tolerância estipulada, gerando *flags* para cada uma delas. Tal processo já estava presente no DSGTools.

Figura 3.14 - Diagrama de hierarquia do processo Identificar áreas pequenas



Este processo é feito exclusivamente em Python. Visando uma melhor compreensão, o código Python foi traduzido para pseudocódigo no contexto desta pesquisa,

gerando o algoritmo 3.

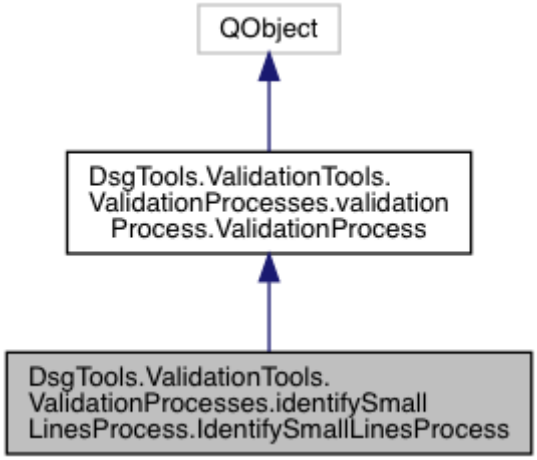
Algoritmo 3: IDENTIFICADOR DE ÁREAS PEQUENAS

```
1  início
2      para cada feição ∈ camada faça
3          geom ← geometria da feição
4          se área de geom < tolerância então
5              geometry ← geometria em wkb
6              featid ← id da feição
7              criar flag com geometry e featid pois a área é pequena
8          fim
9      fim
10 fim
```

3.2.3.1.10 Identificar Linhas pequenas

Este processo é responsável por identificar feições que possuem linhas com comprimento abaixo de uma tolerância estipulada, gerando *flags* para cada uma delas. Tal processo já estava presente no DSGTools.

Figura 3.15 - Diagrama de hierarquia do processo Identificar linhas pequenas



Este processo é feito exclusivamente em Python e, para melhor compreensão, foi

traduzido para pseudocódigo gerando o algoritmo 4:

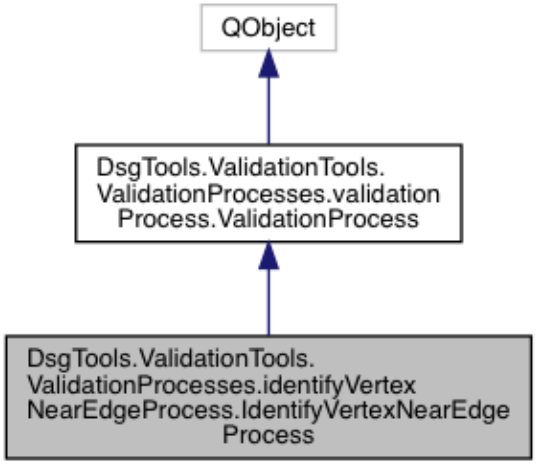
Algoritmo 4: Identificador de linhas pequenas

```
1  início
2      para cada feição ∈ camada faça
3          geom ← geometria da feição
4          se comprimento de geom < tolerância então
5              geometry ← geometria em wkb
6              featid ← id da feição
7              criar flag com geometry e featid pois a área é pequena
8          fim
9      fim
10 fim
```

3.2.3.1.11 Identificar Vértices próximo a Arestas

Este processo identifica qualquer ocorrência de vértices de uma geometria cuja a distância para uma aresta da mesma seja menor que uma tolerância. Tal processo já estava presente no DSGTools.

Figura 3.16 - Diagrama de hierarquia do processo Identificar vértices próximo a arestas



Este processo é baseado em duas consultas SQL. A primeira consulta é responsável pela criação de duas tabelas temporárias, uma tabela com os segmentos que formam as geometrias da tabela que está sobre análise e outra tabela com os vértices da mesma tabela. A primeira consulta, para o caso de camadas do tipo polígono, segue

os passos vistos a seguir:

a) Inicialmente deve-se selecionar as linhas referentes aos anéis dos polígonos. Isto acontece por meio do seguinte trecho de consulta, chamada de A:

```
SELECT
coluna_chave as coluna_chave,
(
ST_Dump(ST_Boundary(coluna_geometrica))
).geom
FROM only
esquema.tabela
```

b) Em seguida, são criados os pontos que formarão os segmentos. Isto acontece por meio do seguinte trecho de consulta, chamado de B:

```
SELECT

ST_PointN(
coluna_geometrica,
generate_series(1, ST_Npoints(coluna_geometrica)-1)) as sp,

ST_PointN(
coluna_geometrica,
generate_series(2, ST_NPoints(coluna_geometrica))) as ep,

linestrings.coluna_chave as coluna_chave
FROM
(A) AS linestrings
```

c) Com os pontos, os segmentos podem ser de fato criados. Isto ocorre por meio do seguinte trecho de consulta, chamado de C:

```
drop table if exists seg;
create temp table seg as
(SELECT
segments.coluna_chave as coluna_chave,
ST_MakeLine(sp,ep) as coluna_geometrica
FROM
(B) AS segments)
```

d) Agora a tabela de pontos pode ser criada. Ela contém todos os vértices da camada e é feita por meio do seguinte trecho de consulta, chamado de D:

```
drop table if exists pontos;
create temp table pontos as
select
coluna_chave as coluna_chave,
(ST_DumpPoints(coluna_geometrica)).geom as coluna_geometrica
from only
esquema.tabela
```

e) Finalmente, são criados os índices para agilizar a consulta. Isto pode ser visto a seguir:

```
create index pontos_gist on pontos using gist (coluna_geometrica);
create index seg_gist on seg using gist (coluna_geometrica)
```

Continuando o processo, é efetuada a consulta propriamente dita, ou seja, a busca por vértices, geometrias da tabela pontos, que possuam distância para os segmentos da tabela *seg* menores que uma tolerância estabelecida, excluindo os pontos que pertençam ao próprio segmento. Isto é feito por meio da consulta 3.7:

Consulta 3.7 - Identificar vértices próximos a arestas

```
select
pontos.coluna_chave,
ST_SetSRID(pontos.coluna_geometrica,epsg) as coluna_geometrica
from
pontos, seg
where
ST_DWithin(seg.coluna_geometrica, pontos.coluna_geometrica, tol)
and ST_Distance(seg.coluna_geometrica, pontos.coluna_geometrica) > 0
```

Cada vértice selecionado por meio da consulta é usado para, junto aos ids, criar as *flags* deste processo.

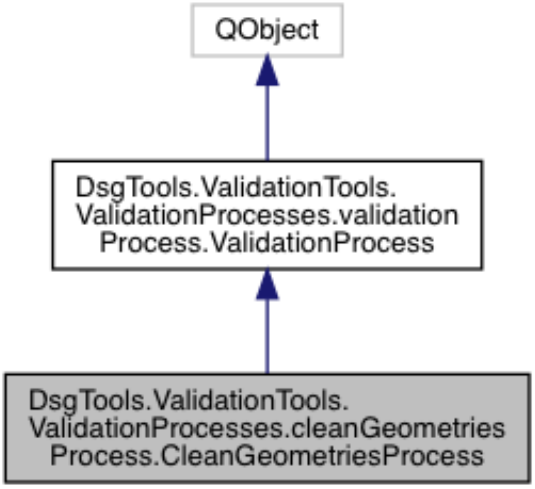
3.2.3.2 Processos de Correção

Os processos de correção são os responsáveis por tratar os problemas topológicos presentes nas geometrias armazenadas no banco. Alguns dos processos desenvolvidos para correção trabalham com base nas flags levantadas pelos processos de identificação, outros não necessitam de flags, tendo em vista que atuam em todas as geometrias presentes nas camadas que estão sendo processadas.

3.2.3.2.1 Limpar Geometrias

Este processo é responsável por lidar com geometrias que violam o conceito de geometria limpa apresentado na revisão bibliográfica. Este processo é baseado nas ferramentas vetoriais do *GRASS* (GRASS Development Team, 2018) que estão disponíveis por meio da *API Python* do *QGIS*. Tal processo já estava presente no DSGTools.

Figura 3.17 - Diagrama de hierarquia do processo Limpar geometrias

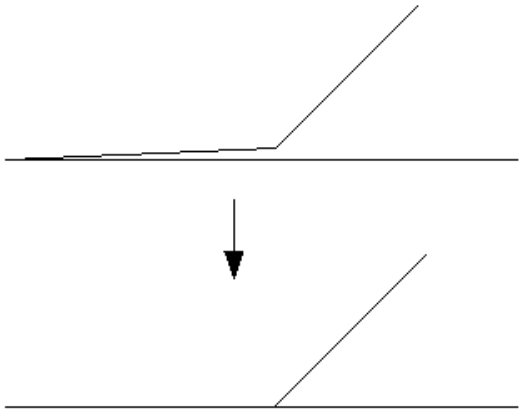


O processamento é feito com o uso do *v.clean* do *GRASS*, usando as seguintes ferramentas em sequência:

3.2.3.2.1.1 RMSA (Remove small angles)

Esta ferramenta é responsável pela eliminação de ângulos muito pequenos formados por segmentos consecutivos, tanto em linhas quanto em polígonos (GRASS Development Team, 2018). O resultado do uso desta ferramenta pode ser visto na figura 3.18. De acordo com a documentação do *GRASS*, o *RMSA* deve ser seguido do *break* e do *rmdupl*.

Figura 3.18 - Efeito do RMSA



Fonte: (GRASS Development Team, 2018)

3.2.3.2.1.2 BREAK

O break quebra linhas e anéis de polígonos em interseções e também quebra linhas e anéis de polígonos que formam uma volta colapsada (GRASS Development Team, 2018). Esta ferramenta deve ser seguida do *rmdupl*.

3.2.3.2.1.3 RMDUPL (Remove duplicates)

O rmdupl remove geometrias que possuem o mesmo conjunto de vértices (GRASS Development Team, 2018). Esta ferramenta deve ser usada depois do *break*.

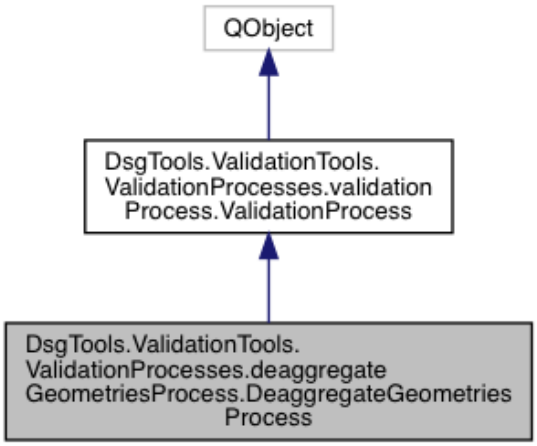
3.2.3.2.1.4 RMDANGLE (Remove dangles)

Uma linha ou anel é considerado como ponta solta se nenhuma outra linha está tocando pelo menos um de seus nós finais (GRASS Development Team, 2018). Este conceito é o mesmo apresentado no algoritmo desenvolvido para identificar pontas soltas. No GRASS esta ferramenta funciona com o uso de uma tolerância, removendo todas as pontas soltas menores que a tolerância estipulada. Se a tolerância for menor que zero, exemplo -1, todas as pontas soltas são removidas.

3.2.3.2.2 Desagregar Geometrias

Este processo é responsável por tornar geometrias multiparte e geometrias de parte única. Isto é feito para resolver o problema apresentado no item Geometrias multiparte. Tal processo já estava presente no DSGTools.

Figura 3.19 - Diagrama de hierarquia do processo Desagregar geometrias



Este processo é executado por meio do pseudoalgoritmo 5, que foi traduzido do código Python, previamente existente, no contexto desta pesquisa:

Algoritmo 5: DESAGREGADOR DE GEOMETRIAS

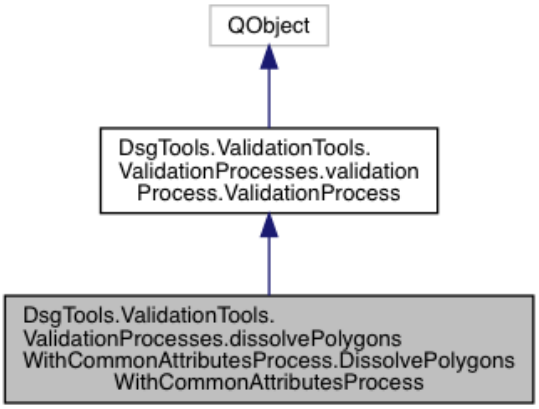
```
addList = []
início
    para cada feição ∈ camada faça
        geom ← geometria da feição
        se geom é vazia então
            continuar
        fim
        se partes de geom > 1 então
            partes ← lista de partes de geom
            para i de 1 até tamanho de partes faça
                se parte não é nulo então
                    newFeat ← nova feição a partir de feição
                    newFeat.geometry ← partes[i]
                    adiciona newFeat em addList
                fim
            fim
            feat.geometry ← partes[0]
            atualizar feat na camada
            adicionar feições de addList na camada
        fim
    fim
fim
```

Este algoritmo percorre as feições da camada em análise e, para cada feição cuja geometria possua número de partes maior que 1, é feito o processo de desagregação, que consiste em criar novas feições com as partes de índice 1 em diante. A feição original tem sua geometria atualizada para a parte de índice 0.

3.2.3.2.3 Dissolver Polígonos Com Atributos Comuns

Este processo funciona em camadas do tipo área e executa o processo de dissolução de fronteiras para feições que possuam atributos comuns (MARTINEZ-LLARIO et al., 2009). Este processo usa o algoritmo de dissolução disponível na API do QGIS. Tal processo já estava presente no DSGTools. Os pseudocódigos 6, 7, 8 e 9 foram traduzidos no contexto desta pesquisa para uma melhor compreensão.

Figura 3.20 - Diagrama de hierarquia do processo Dissolver geometrias com atributos comuns



Para auxiliar o uso de um valor de área mínima como tolerância no processo, foi criando um algoritmo para criar uma camada temporária a partir da camada original, combinando todos os atributos escolhidos pelo usuário em um único atributo chamado tupple, que consiste de todos os atributos originais combinados por meio de uma separação por vírgulas.

O uso de um valor de área como tolerância para o processo faz com que somente feições com geometrias que possuam área menor que o estipulado sejam processados. Isto levou ao desenvolvimento de um algoritmo para ajustar o atributo tupple para uso na ferramenta de dissolução do QGIS.

O algoritmo 6 é apresentado em etapas a seguir:

Algoritmo 6: Dissolver polígonos com atributos comuns

```
1  início
2      criação de listas
3      ajuste de atributo d_id
4      atualização do atributo tupple na camada temporária
5  fim
```

A parte de criação de listas é feita por meio do algoritmo 7:

Algoritmo 7: Criação das listas

```
1  início
2      criar campo d_id na camada
3      criar lista de feições pequenas (smallFeatureList)
4      criar lista de feições grandes (bigFeatureList)
5      criar índice espacial para as feições grandes (bigFeatIndex)
6      para cada feição ∈ camada faça
7          d_id da feição ← id da feição
8          se área da geometria da feição < tolerância então
9              adiciona feição em smallFeatureList
10         fim
11         senão
12             adiciona feição em bigFeatureList
13             adiciona feição em bigFeatIndex
14         fim
15     fim
16 fim
```

A parte de ajuste de atributo d_id é feita por meio do algoritmo 8:

Algoritmo 8: Ajuste do atributo d_id

```
1  início
2      para cada feição ∈ smallFeatureList faça
3          candidatos ← lista de candidatos obtidos em bigFeatIndex
4          para cada canditato ∈ candidatos faça
5              se atributo d_id = id da feição pequena e feição pequena intersecta
                candidato e atributo tupple de feição pequena = atributo tupple de
                candidato então
6                  atributo d_id de feição pequena = id do candidato
7              fim
8          fim
9      fim
10 fim
```

Finalmente, a parte de atualização da camada temporária é feita por meio do algoritmo 9:

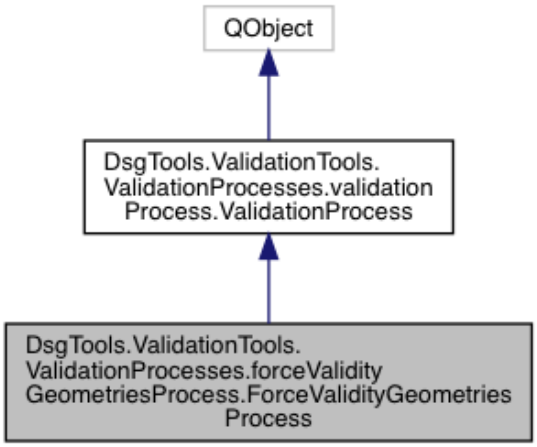
Algoritmo 9: Atualização da camada temporária

```
1  início
2      para cada feição ∈ smallFeatureList+bigFeatureList faça
3          adicionar d_id em tupple
4          atualizar feição na camada com o novo valor de tupple
5      fim
6  fim
```

3.2.3.2.4 Forçar Validade de Geometrias

Este processo é responsável por lidar com feições que possuem geometrias que violam o conceito de validade apresentado na seção 2.4.2. Este processo tem como requisito a execução do processo do item 3.2.3.1.6, pois funciona baseado nas *flags* levantadas por aquele processo. Tal processo já estava presente no DSGTools.

Figura 3.21 - Diagrama de hierarquia do processo Forçar validade de geometrias



Este processo é feito por meio da consulta 3.8:

Consulta 3.8 - Forçar validade de geometrias

```
update esquema.tabela set
coluna_geometrica = ST_Multi(result.coluna_geometrica)
from

(select distinct
parts.coluna_chave,
ST_Union(parts.coluna_geometrica) as coluna_geometrica
from
esquema.tabela as source,

(select
coluna_chave as coluna_chave,
ST_Multi(
(ST_Dump(ST_SetSRID(ST_MakeValid(coluna_geometrica), srid))).geom
) as coluna_geometrica
from
esquema.tabela
where
coluna_chave in (lista_ids)
) as parts
where
ST_GeometryType(parts.coluna_geometrica) = ST_GeometryType(source.coluna_geometrica
    )
group by parts.coluna_chave
) as result
where
result.coluna_chave = esquema.tabela.coluna_chave
```

O funcionamento desta consulta em linhas gerais segue as seguintes etapas:

a) Selecionar as geometrias originais que foram previamente identificadas

como inválidas

b) Tornar estas geometrias válidas com o uso do *ST_MakeValid* seguido do *ST_Dump*. Isto pode gerar um conjunto de geometrias válidas correspondente a uma geometria inválida.

c) Selecionar somente as partes que possuam mesmo tipo geométrico das geometrias originais. Isto é feito para evitar casos como os de polígonos com vértices repetidos. O *ST_MakeValid* vai retornar uma geometria multiparte que consiste do polígono, sem os vértices repetidos, somada aos vértices repetidos. O uso do posterior do *ST_Dump* gerará geometrias de parte única para cada parte, sendo somente uma destas partes do tipo polígono, fazendo com que as partes do tipo ponto não sejam necessárias.

d) O resultado acima é agrupado pelo id para que grupos de geometrias com mesmo id sejam usadas na função agregadora *ST_Union* individualmente, podendo criar uma geometria multiparte, porém válida.

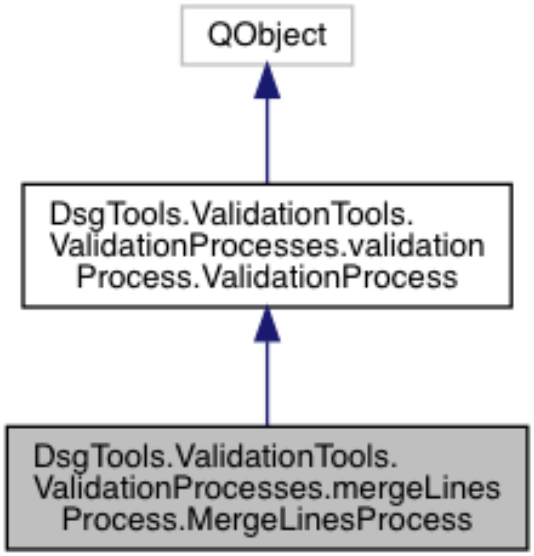
e) O resultado da função *ST_Union* é usado para atualizar a tabela original.

Cabe ressaltar que este processo pode ser seguido pelo processo Desagregar Geometrias.

3.2.3.2.5 Fundir Linhas

Este processo é responsável por unir feições que possuem geometrias do tipo linha que possuem um mesmo conjunto de atributos. Este processo é útil para a garantia de integridade topológica em redes, fazendo com que linhas com mesmo atributo sejam unidas.

Figura 3.22 - Diagrama de hierarquia do processo Fundir Linhas



Este processo é puramente baseado em código Python, que foi desenvolvido e traduzido para os pseudocódigos 10, 11 e 12, no contexto desta pesquisa:

Algoritmo 10: FUNDIR LINHAS

Entrada: atributosProibidos

```
1 início
2 |  criar do mapa chave/valor de feições (featuresDict)
3 |  unir linhas
4 fim
```

A etapa de criação do mapa chave/valor utilizado no algoritmo é feito por meio do

algoritmo 11:

Algoritmo 11: Criação do mapa chave/valor de feições

```
featuresDict =
columns = Vazio início
    para cada feição ∈ camada faça
        se não há nada em columns então
            preencher columns com as colunas da camada
        fim
        attributes = []
        para cada column ∈ columns faça
            se column ∈ atributosProibidos então
                continuar
            fim
            obter o valor do atributo column e adicionar em attributes
        fim
        unificado = unificação de todos os valores de attributes em uma string
         se unificado ∉ chaves de atributosProibidos então
            adicionar unificado como chave de atributosProibidos
            criar uma lista vazia para unificado em atributosProibidos
        fim
        adicionar feição na lista de unificado em atributosProibidos
    fim
fim
```

A etapa que é responsável por realizar a união das linhas é feita por meio do algo-

ritmo 12:

Algoritmo 12: UNIÃO DAS LINHAS

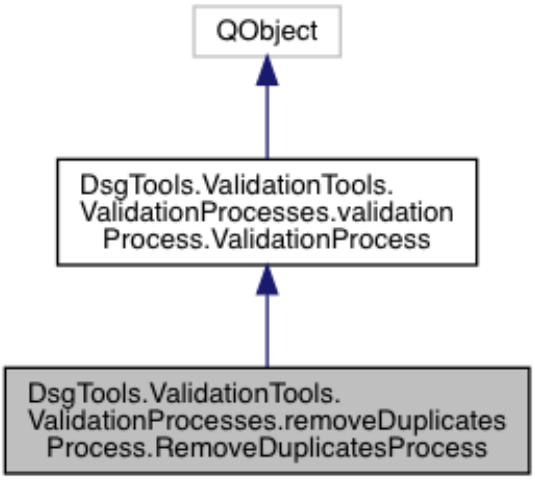
```
idsToRemove = []
início
    para cada chave ∈ chaves de featuresDict faça
        features ← featuresDict[chave]
        para cada feature ∈ features faça
            se feature.id() ∈ idsToRemove então
                continuar
            fim
            geom ← feature.geometry()
            para cada other ∈ features faça
                se other.id() = feature.id() então
                    continuar
                fim
                se feature.id() ∈ idsToRemove então
                    continuar
                fim
                se geom toca other.geometry() então
                    geom ← geom unificada à other.geometry()
                    atualizar geometria de feature na camada com geom
                fim
            fim
        fim
    fim
fim
```

Em seguida, automaticamente, visando criar nós nas interseções entre as linhas, é executado o processo Limpar Geometrias.

3.2.3.2.6 Remover Geometrias Duplicadas

Este processo remove as geometrias identificadas como duplicadas pelo processo Identificar Geometrias Duplicadas. Tal processo já estava presente no DSGTools.

Figura 3.23 - Diagrama de hierarquia do processo Remover geometrias duplicadas



Para funcionar, das *flags* de geometrias duplicadas o processo obtém os ids que necessitam ser removidos e executa a seguinte consulta 3.9:

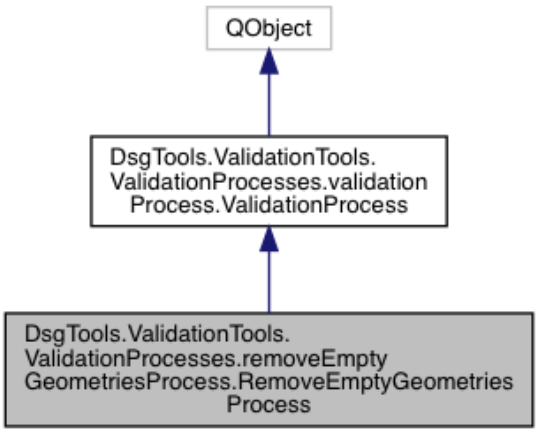
Consulta 3.9 - Remover feições

```
DELETE FROM "esquema"."tabela"
WHERE coluna_chave in (lista_ids_para_remoção)
```

3.2.3.2.7 Remover Geometrias Vazias

Este processo remove as geometrias vazias (STOLZE, 2003) das tabelas sob análise. Ele não necessita das *flags* pois a identificação e remoção são feitas de maneira unificada devido ao fato de não ser possível criar *flags* com posição para este tipo de problema. Tal processo já estava presente no DSGTools.

Figura 3.24 - Diagrama de hierarquia do processo Remover geometrias vazias



O processo executa a consulta 3.10 para remover geometrias vazias.

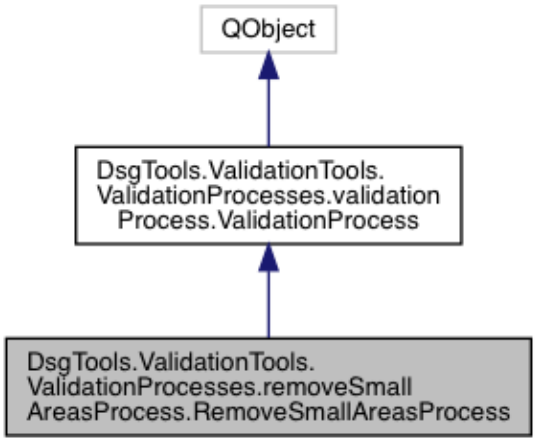
Consulta 3.10 - Remover geometrias vazias

```
DELETE FROM "esquema"."tabela"
WHERE st_isempty("coluna_geométrica") = TRUE
```

3.2.3.2.8 Remover Áreas Pequenas

Este processo remove as geometrias identificadas como duplicadas pelo processo Identificar Áreas Pequenas. Tal processo já estava presente no DSGTools.

Figura 3.25 - Diagrama de hierarquia do processo Remover áreas pequenas

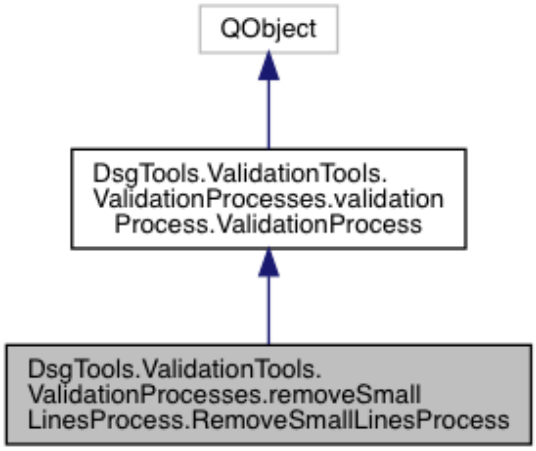


Para funcionar, das *flags* de geometrias com áreas pequenas o processo obtém os ids que necessitam ser removidos e executa a consulta 3.9.

3.2.3.2.9 Remover Linhas Pequenas

Este processo remove as geometrias identificadas como duplicadas pelo processo Identificar Linhas pequenas. Tal processo já estava presente no DSGTools.

Figura 3.26 - Diagrama de hierarquia do processo Remover linhas pequenas

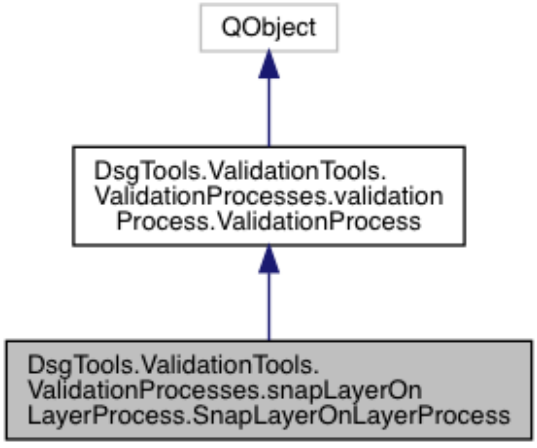


Para funcionar, das *flags* de geometrias com linhas pequenas o processo obtém os ids que necessitam ser removidos e executa a consulta 3.9:

3.2.3.2.10 Colar Camada em Camada

Este processo é capaz de colar uma lista de camadas em uma camada de referência, um processo comumente conhecido como *snap*. Este processo é capaz de realizar *snap* de vértices em segmentos sem gerar vãos ou sobreposições. Tal processo já estava presente no DSGTools.

Figura 3.27 - Diagrama de hierarquia do processo Colar camada em camada



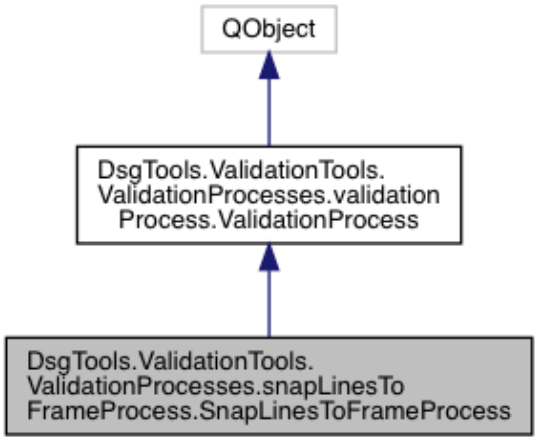
Completamente implementado em Python, o algoritmo desenvolvido neste trabalho

é uma tradução do já existente no QGIS. O código original, em C++, está nos arquivos qgsgeometrysnapper.h e qgsgeometrysnapper.cpp que estão disponíveis no repositório *online* do QGIS na pasta *QGIS/src/analysis/vector/*.

3.2.3.2.11 Colar Linhas na Moldura

Este processo é responsável por colar linhas em uma camada selecionada como moldura. A ação de colar uma camada em outra pode ser feito por meio do processo Colar Camada em Camada, porém este método realiza a operação de *snap* por meio de um prolongamento das linhas até que as mesmas toquem a camada definida como moldura. Tal processo já estava presente no DSGTools.

Figura 3.28 - Diagrama de hierarquia do processo Colar linhas na moldura



Este processo é feito por meio da execução de duas consultas SQL. A primeira consulta é responsável por trabalhar com os pontos iniciais das linhas e a segunda é responsável por trabalhar com os pontos finais das linhas. A idéia geral de funcionamento das consultas pode ser entendida por meio dos passos apresentados para lidar com os pontos iniciais das linhas.

a) Inicialmente, os pontos iniciais das linhas são usados para criar segmentos que os liguem até a camada definida como moldura. Isto é feito com o uso do *ST_ShortestLine*. Isto é feito por meio da seguinte consulta SQL, chamada de *A*:

```
select
a."coluna_chave" as "coluna_chave",
```

```
a."coluna_geometrica" as "coluna_geometrica",
ST_ShortestLine(st_startpoint((ST_Dump(a."coluna_geometrica")).geom),
ST_Boundary(m."coluna_geometrica_moldura")) as from_start

from
"esquema"."tabela" a, "esquema_moldura"."tabela_moldura" m
```

b) A segunda etapa consiste em atualizar a camada para que seus pontos iniciais sejam os pontos referentes aos pontos finais dos segmentos criados por meio de *A*, desde que o comprimento de *from_start* seja menor que a tolerância estipulada. Isso se dá no trecho de SQL a seguir, chamado de *B*:

```
select
short."coluna_chave",
St_SetPoint((ST_Dump(short."coluna_geometrica")).geom,
0,
ST_EndPoint(from_start)) as newline

from (A) as short

where
ST_Length(from_start) < tolerancia
```

c) Em seguida, as geometrias tem que ser reconstruídas com o uso do *ST_Union* devido ao uso do *ST_Dump*. Isso ocorre no seguinte trecho de consulta SQL, chamado de *C*:

```
select
simplelines."coluna_chave" as "coluna_chave",
ST_Union(simplelines.newline) as "coluna_geometrica"
from (B) as simplelines group by simplelines."coluna_chave"
```

d) Finalizando, a camada é atualizada com as geometrias reconstruídas. Isso se dá no seguinte trecho de SQL:

```
update "esquema"."tabela" as classe set
"coluna_geometrica" = ST_Multi(agrupado."coluna_geometrica")
from (C) as agrupado
```

```
where
where classe."coluna_chave" = agrupado."coluna_chave"
```

Para lidar com os pontos finais das linhas, os seguintes passos foram tomados:

a) A consulta A foi alterada para ficar da seguinte forma:

```
select
a."coluna_chave" as "coluna_chave",
a."coluna_geometrica" as "coluna_geometrica",
ST_ShortestLine(
st_endpoint((ST_Dump(a."coluna_geometrica")."coluna_geometrica"
),
ST_Boundary(m."coluna_geometrica_moldura")) as from_start,
ST_NPoints((ST_Dump(a."coluna_geometrica")."coluna_geometrica") as index
from "esquema"."tabela" a, "esquema_moldura"."tabela_moldura" m
```

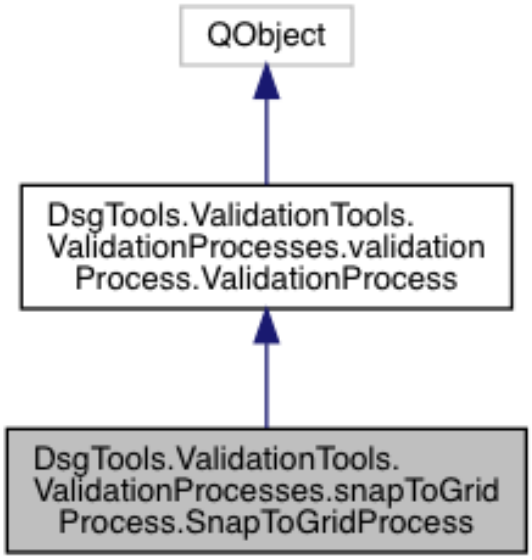
b) A consulta B foi alterada para ficar da seguinte forma:

```
select
short."coluna_chave",
St_SetPoint((ST_Dump(short."coluna_geometrica")).geom,
short.index - 1,
ST_EndPoint(from_start)) as newline

from (A) as short

where
ST_Length(from_start) < tolerancia
```

3.2.3.2.12 Colar na Grade

Este processo é responsável por colar todos os vértices de uma geometria em uma malha regular com espaçamento definido pelo usuário. O processo é baseado em uma consulta SQL que faz uso da função ST_SnapToGrid. Tal processo já estava presente no DSGTools.

Figura 3.29 - Diagrama de hierarquia do processo Colar na grade



O processo é executado por meio da consulta 3.11.

Consulta 3.11 - Colar na grade

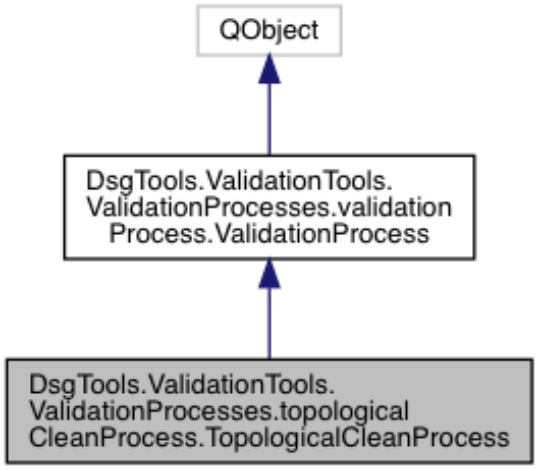
```
update "esquema"."tabela" set
"coluna_geometrica" =
ST_SetSRID(ST_SnapToGrid("coluna_geometrica",espaçamento),srid)
```

Sendo assim, na prática, este processo pode ser utilizado para efetuar arredondamentos nas coordenadas dos vértices das geometrias de uma camada. Podendo, inclusive, ter efeito na precisão das coordenadas.

3.2.3.2.13 Limpeza Topológica

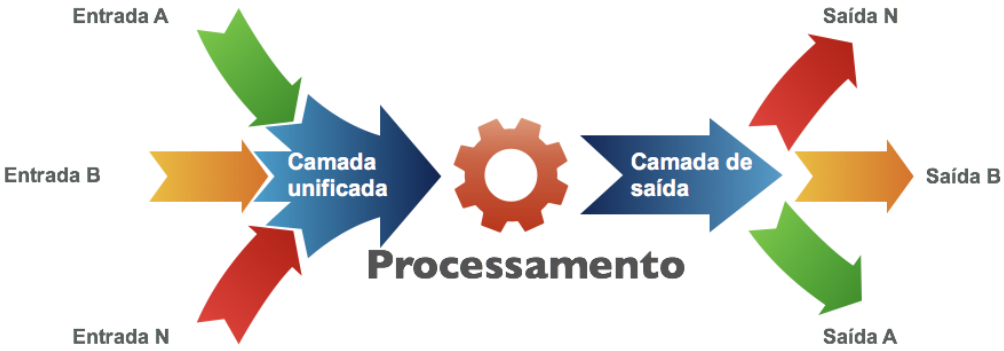
Este processo é igual ao processo Limpar Geometrias quanto a execução, porém difere quanto a entrada de dados. No processo Limpar Geometrias a entrada das camadas para processamento é individualizada, enquanto que neste processo a entrada é unificada. Este processo já estava presente no DSGTools, porém sofreu melhorias durante a execução da pesquisa.

Figura 3.30 - Diagrama de hierarquia do processo Limpeza topológica



O processo de unificação das camadas é responsável por criar uma camada temporária com todas as geometrias das camadas selecionadas para processamento. A camada temporária criada possui como atributos o id da feição original e o nome da camada original. Isto é feito para que seja possível, a partir da camada unificada, atualizar as camadas originais após o processamento. A figura 3.31 mostra o esquema de funcionamento do processo.

Figura 3.31 - Funcionamento do processo Limpeza topológica

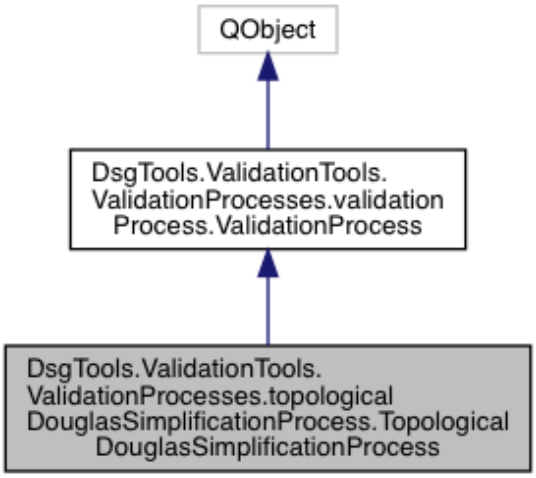


3.2.3.2.14 Simplificação Topológica de Douglas Peucker

Este processo é baseado no mesmo funcionamento que o processo Limpeza Topológica. A diferença reside no fato de se simplificar as geometrias das camadas proces-

sadas por meio do uso do algoritmo de simplificação Douglas Peucker (DOUGLAS; PEUCKER, 1973) implementado na ferramenta *v.generalize*. Este processo se torna útil ao se desejar simplificar múltiplas camadas de maneira unificada sem que sejam criados vão e sobreposições. Tal processo já estava presente no DSGTools.

Figura 3.32 - Diagrama de hierarquia do processo Limpeza topológica de Douglas Peucker



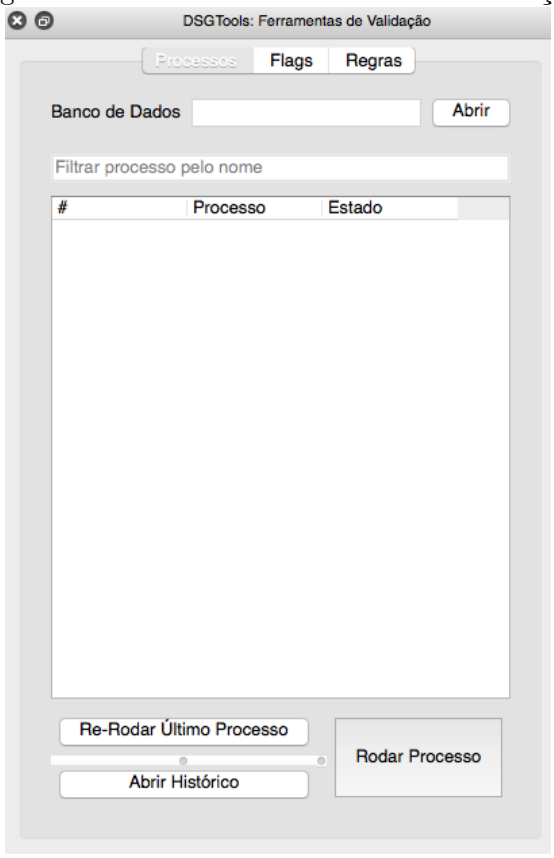
4 RESULTADOS E DISCUSSÃO

Neste capítulo são apresentados os resultados obtidos por meio das implementações apresentadas no capítulo 3. Em seguida, é apresentado um caso de teste onde fluxos de processos propostos são executados e discutidos.

4.1 Utilização da Caixa de Ferramentas de Validação do DSGTools

Dos processos mencionados neste trabalho, havia um conjunto já existente no DSGTools (DSG, 2015) que foram sujeitos a uma profunda análise de funcionamento. Outra parte dos processos apresentados foram implementados no contexto desta pesquisa, já sendo disponibilizados no DSGTools, podendo ser acessados na Caixa de Ferramentas de Validação do DSGTools (figura 4.1).

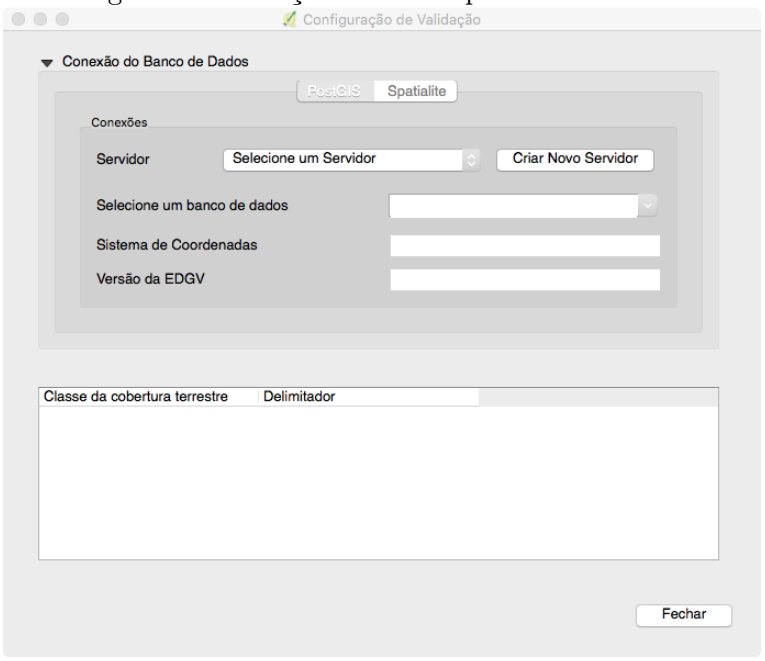
Figura 4.1 - Caixa de Ferramentas de Validação



Para iniciar o trabalho, o banco de dados que terá suas tabelas com dados geoespaciais vetoriais validadas deve ser escolhido pelo usuário. Isto pode ser visto por meio

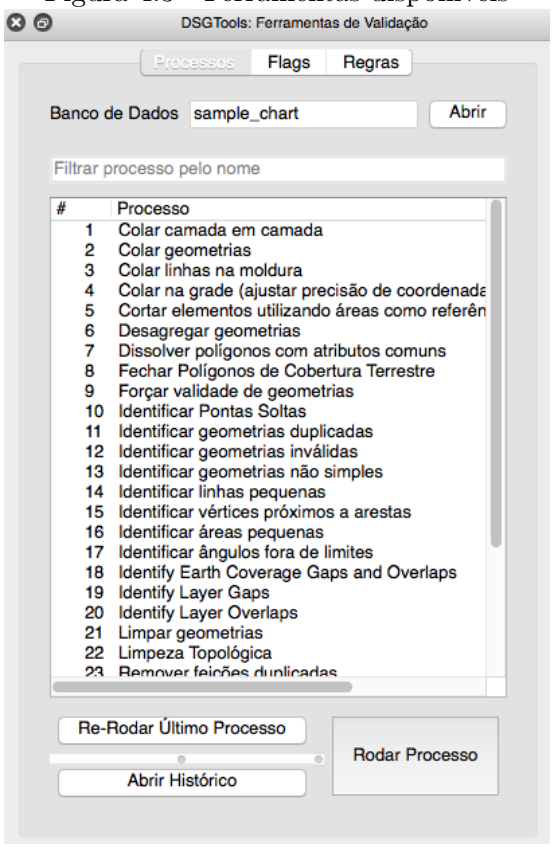
da figura 4.2.

Figura 4.2 - Seleção de banco para ser validado



Após a seleção do banco, a Caixa de Ferramentas é populada com a lista de processos disponíveis, que já podem ser executados para processar os dados vetoriais armazenados no SGBD.

Figura 4.3 - Ferramentas disponíveis

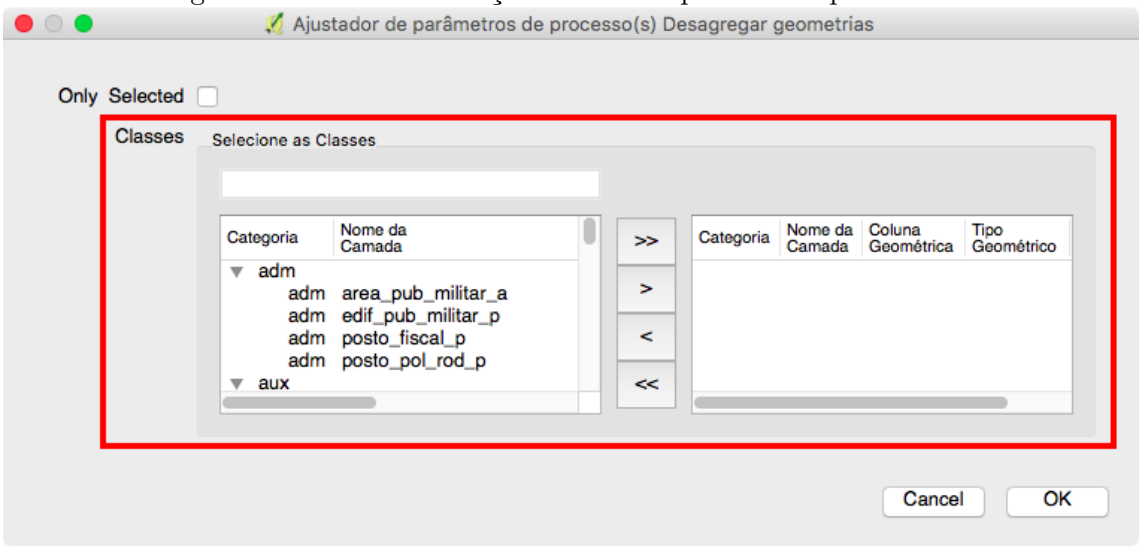


4.1.1 Execução de processos

Após o carregamento do banco, os processos já podem ser executados pelo usuário. Isto é feito por meio da seleção do processo seguido do clique no botão *Rodar Processo*, que pode ser visto na figura 4.3. Cada processo tem um ajustador de parâmetros necessários que é disponilizado ao usuário por meio de uma janela que é aberta assim que se clica no botão *Rodar Processo*.

Uma parte comum a todos os processos é a seleção das tabelas que serão processadas. Múltiplas tabelas podem ser selecionadas para processamento, e podem, de acordo com o processo desejado, serem processadas individualmente ou em conjunto, por meio da criação de uma tabela unificada (caso do Limpeza Topológica). Cabe ressaltar que somente tabelas que possuem geometrias são disponibilizadas ao usuário, tabelas ainda vazias não são mostradas para seleção. A seleção de tabelas pode ser vista na área em destaque da figura 4.4.

Figura 4.4 - Caixa de seleção de tabelas para serem processadas



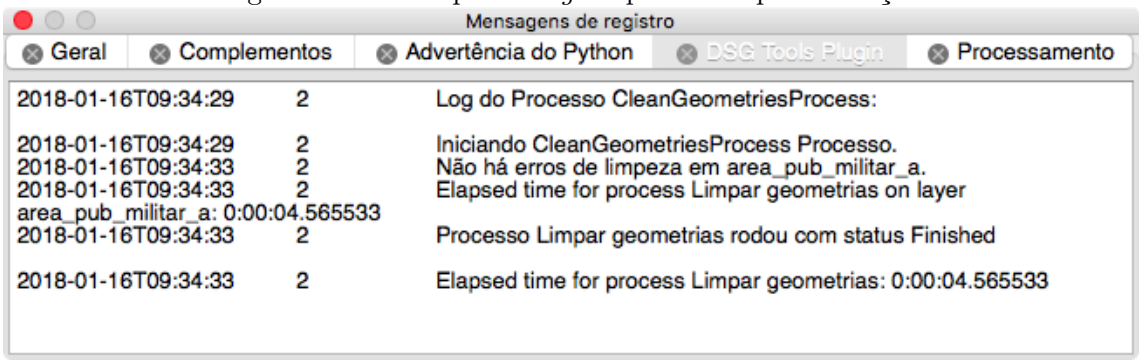
Os demais parâmetros dos processos são disponibilizados de acordo com cada processo. Um manual do DSGTools foi elaborado para indicar como cada processo funciona, quais seus efeitos e quais parâmetros são necessários para sua execução. Tal manual pode ser acessado por meio do menu mostrado na área em destaque da figura 4.5 (botão Ajuda) ou por meio da url `https://github.com/dsgoficial/DsgTools/wiki`.

Figura 4.5 - Caixa de seleção de tabelas para serem processadas

Após a completa seleção dos parâmetros, o processo inicia sua execução. Ao término

da execução, aparecerá ao usuário uma mensagem indicando o estado final da execução. Caso o processo seja executado com sucesso, é possível obter informações de tempo de execução total e tempo de execução por tabela, como se pode ver na figura 4.6, caso contrário, o *log* do QGIS mostrará informações referentes ao problema de execução encontrado.

Figura 4.6 - Exemplo de *log* do processo após execução

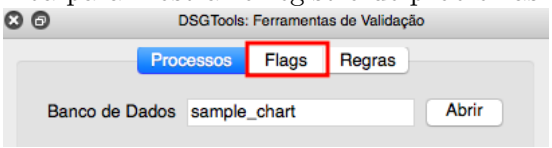


Por fim, caso seja necessário rodar novamente o último processo com os mesmos parâmetros usados anteriormente, é possível clicar no botão *Re-rodar último processo*.

4.1.2 Registro de Identificação de Problemas

Os processos de identificação, ou seja, os processo listados na seção 3.2.3.1, criam registros para cada problema identificado por eles. O acesso aos registros é feito por meio da aba *Flags* que pode se vista no destaque da figura .

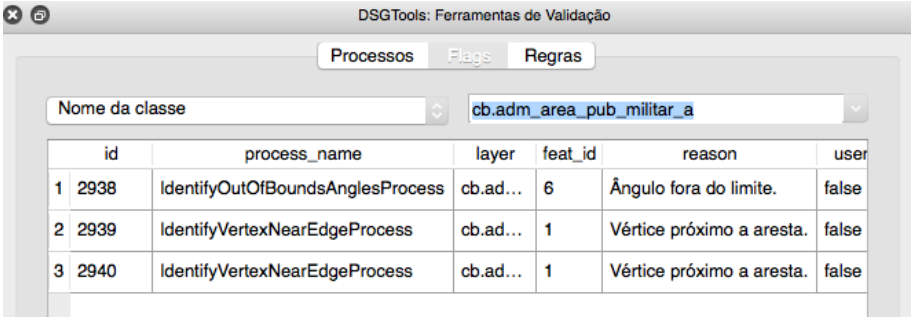
Figura 4.7 - Aba para mostrar o registro de problemas identificados



Ao se clicar na aba *Flags* é possível ver todos os registros de problemas identificados. A seleção de problemas pode ser filtrada pelo nome da tabela (tabelas são chamadas de classes no DSGTools, devido à influência da ET-EDGV) ou pelo nome do processo. Na figura 4.8 é possível ver o registro de problemas para a tabela chamada

cb.adm_area_pub_militar_a. Nesta figura é possível ver que a registro de problemas levantados pelos processos Identificar Ângulos Fora dos Limites e Identificar Vértices próximo a Arestas.

Figura 4.8 - Exemplo de registros para a tabela *cb.adm_area_pub_militar_a*



Cada registro de problema pode ser analisado em detalhes por meio da ferramenta *Zoom para flag*, que é acessível por meio do clique com o botão direito do *mouse* em cima da *flag*. As figuras 4.9 e 4.10 mostram, respectivamente, a seleção e visualização de uma *flag* levantada pelo processo Identificar Ângulos Fora dos Limites.

Figura 4.9 - Seleção de uma *flag*

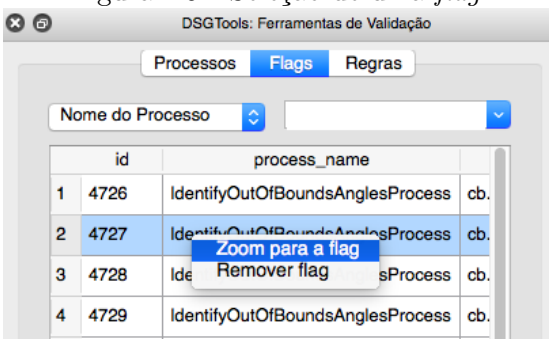
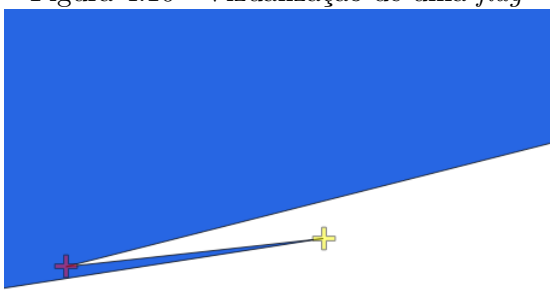


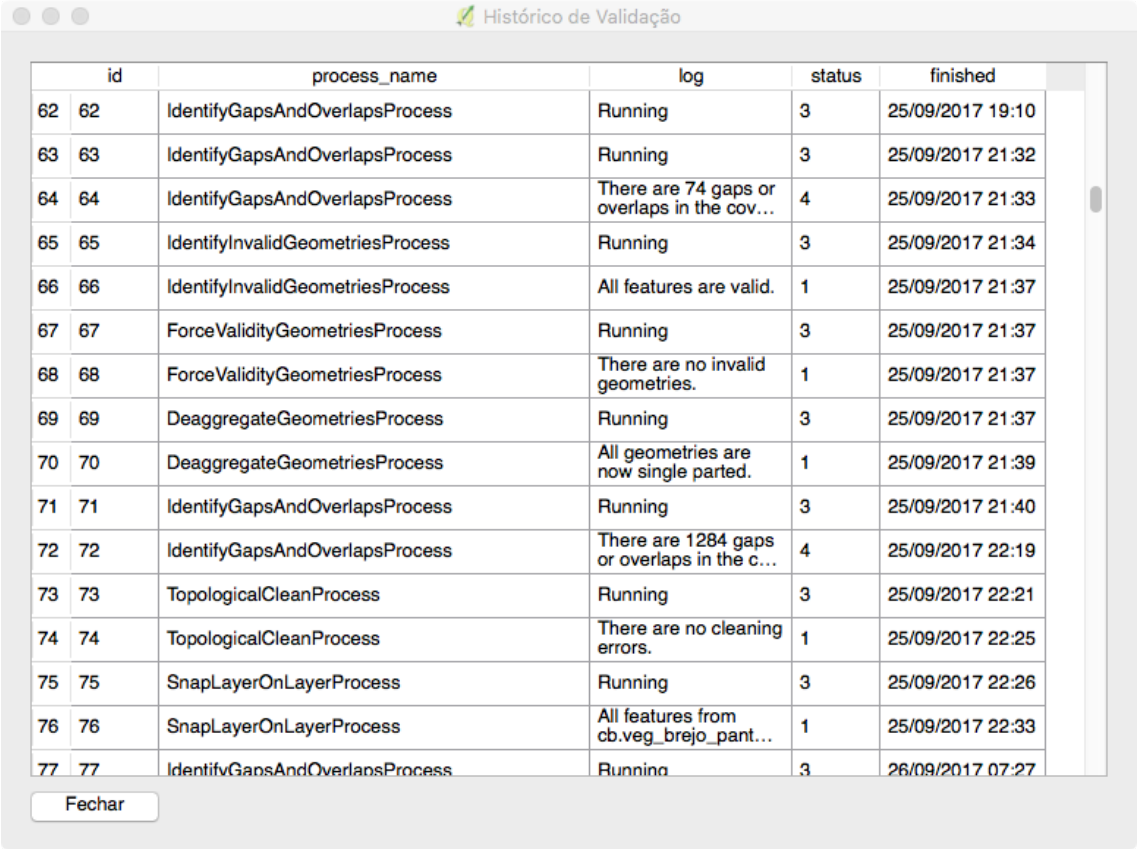
Figura 4.10 - Vizualização de uma *flag*



4.1.3 Histórico de Validação

Visando fornecer uma visão global do que já foi previamente executado durante a validação de um banco de dados específico é possível visualizar o completo histórico de execução de processos por meio do botão *Abrir histórico*. A figura 4.11 mostra um exemplo de histórico.

Figura 4.11 - Histórico de processos de validação



4.2 Caso de teste - Validação de Cobertura Terrestre

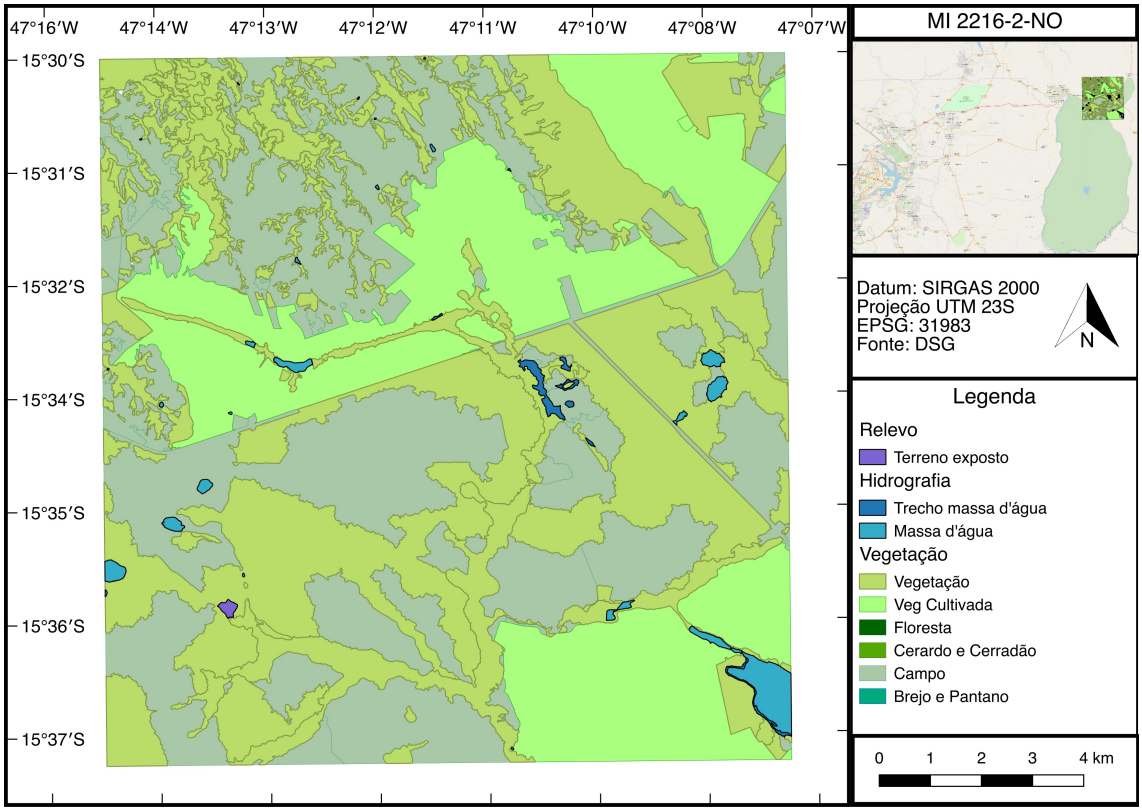
Nesta seção é abordado um caso de teste onde são apresentados os efeitos dos processos de validação quando aplicados por meio dos fluxogramas propostos nesta pesquisa. O caso em voga trata de uma carta obtida logo após a etapa de Aquisição Vetorial no 2º Centro de Geoinformação, que é uma Organização Militar Diretamente subordinada à DSG. Está carta é referente ao MI 2216-2-NO sendo um produto na escala 1:25.000.

Neste caso de teste são abordados os fluxogramas utilizados para validar as camadas que compõem a cobertura terrestre. A cobertura terrestre representa toda a superfície da área a ser mapeada e deve ser totalmente coberta pelas camadas que a compõem, não havendo vãos nem sobreposições. Quaisquer outras camadas mapeadas serão sobrepostas à cobertura terrestre (Exército Brasileiro, 2016).

A figura 4.2 mostra a área do caso de teste e, do produto referente à Carta Topográ-

fica de MI 2216-2-NO, somente as camadas do tipo polígono referentes à hidrografia, vegetação e terreno exposto foram consideradas durante a validação.

Figura 4.12 - Área do caso de teste (MI 2216-2-NO)



É com base nas camadas que compõem a cobertura terrestre que os fluxogramas a seguir foram propostos. Para este caso de teste, ao se seguir os fluxogramas propostos nesta pesquisa, se obteve bons resultados.

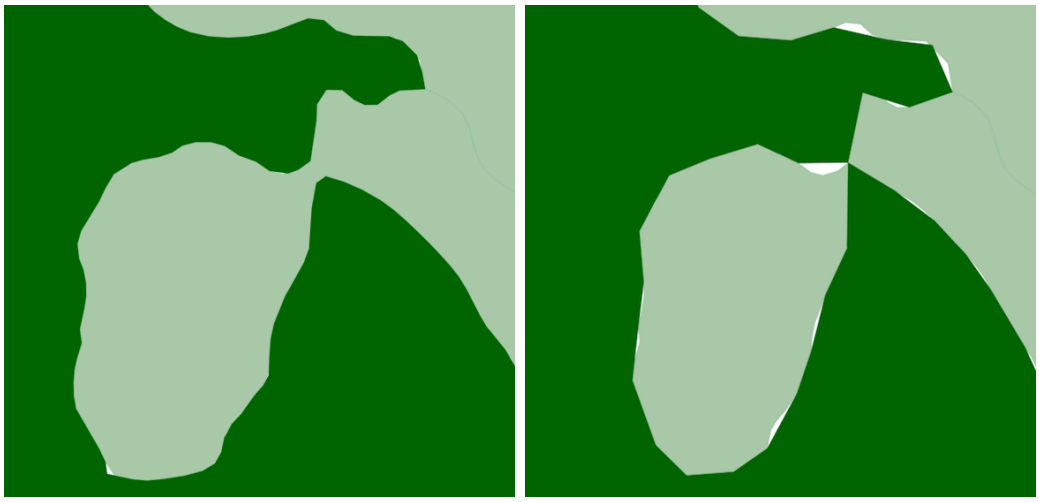
4.2.1 Considerações sobre os parâmetros dos processos

Durante a execução dos processos, foi possível notar que alterações nos valores dos parâmetros utilizados geravam mudanças na qualidade do processamento. Desta forma, foi observado que o uso dos processos deve ser efetuado com valores moderados para os parâmetros, principalmente no que se refere aos parâmetros de;

a) *snap* (ex. processo Limpar Geometrias). Constatou-se que um valor elevado pode vir a corromper a representação territorial e até mesmo gerar

problemas de validade nas geometrias. Sendo assim, verificou-se que iniciar o processo com valores na casa dos centímetros (ex. 0,01m) e ir aumentando a cada iteração, geraram melhores resultados, pois as mudanças nas geometrias ocorrem de forma gradual.

Figura 4.13 - Efeito do snap com parâmetro muito elevado

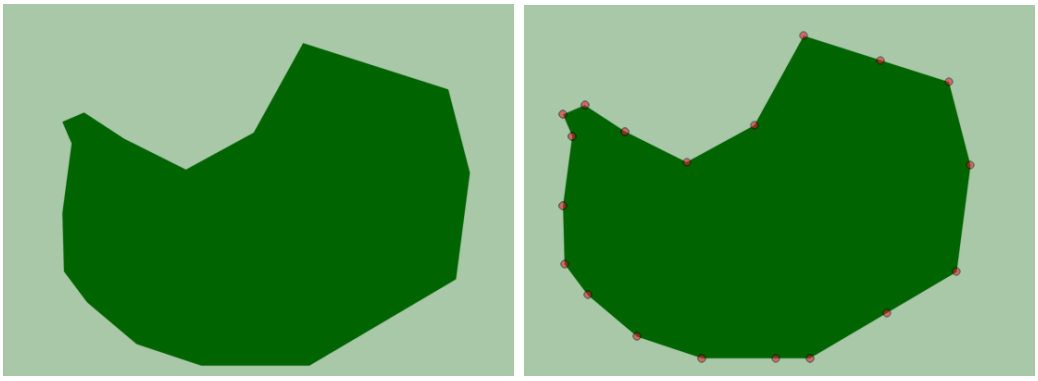


(a) Geometria original

(b) Geometria com sua representação territorial corrompida devido ao uso de um valor evelado para o parâmetro de *snap*

b) tolerância (ex. processo Identificar Vértices próximo a Arestas). Um valor elevado pode acabar identificando todos os vértices da geometria como errôneos, gerando *flags* excessivas que não necessariamente indicam um problema de qualidade do dado.

Figura 4.14 - Identificar vértices com tolerância elevada



(a) Geometria original

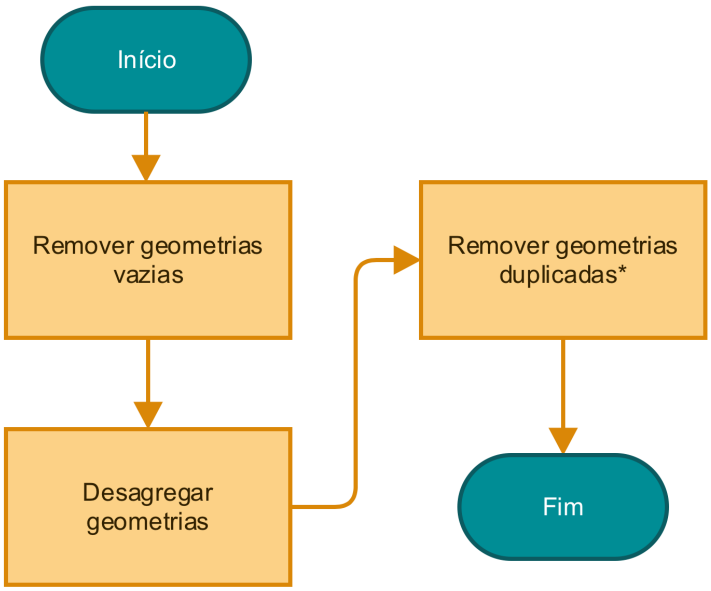
(b) Geometria com todos os seus vértices identificados errôneamente como problemas devido ao uso de um valor evelado para a tolerância

c) ângulo (ex. processo Identificar Ângulos Fora dos Limites). Assim como o caso da tolerância, um valor elevado pode acabar identificando todos os vértices da geometria como errôneos, gerando *flags* excessivas que não necessariamente indicam um problema de qualidade do dado, gerando o mesmo efeito da figura 4.14.

4.2.2 Fluxograma de preparação

Durante os testes de aplicação dos processos no caso de teste, foi verificado que o tratamento interno de camadas deve ser iniciado com a remoção de geometrias vazias para evitar problemas na execução dos processos. Em seguida, as geometrias devem ser desagregadas e as duplicatas deve ser removidas. Desta forma, a figura 4.15 mostra o fluxograma proposto para o processamento inicial das camadas, o qual não gera alterações visuais nas geometrias armazenadas no banco, somente mudanças estruturais.

Figura 4.15 - Fluxograma inicial de validação



Com a execução deste fluxo, não haverá feições com geometrias vazias, todas as geometrias serão individuais e as geometrias duplicadas terão sido removidas do banco. Cabe ressaltar que o processo de remoção de duplicatas possui como pré e pós-processo o processo Identificar Geometrias Duplicadas para que, após o processo de remoção, seja possível verificar se ainda há *flags* de geometrias duplicadas.

Para finalizar o uso deste fluxograma deve-se conferir se há *flags* existentes. A ocorrência de *flags* após esse fluxograma pode comprometer a identificação e harmonização de sistemas de referência espacial, sendo compatível com o apresentado por Laggner e Orthen (2012). Não havendo *flags*, o processo de validação pode ser continuado.

Figura 4.16 - Estado das *flags* após o fluxo inicial

4.2.3 Fluxograma Intracamada

4.2.3.1 Etapa 1

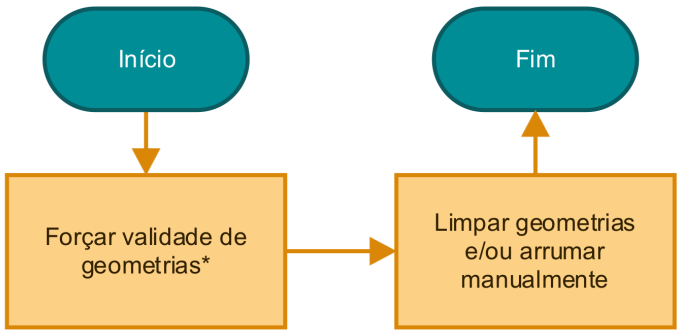
Posteriormente, de forma compatível com o apresentado por Oosterom et al. (2005), foi verificada a importância de se realizar o processamento interno das camadas, ou seja, o relacionamento de geometrias com outras geometrias da mesma camada.

Este processamento interno é efetuado em duas etapas. Sendo o primeiro, o fluxograma da figura 4.17, responsável por tratar problemas referentes a:

a) Validade de geometrias; e

b) Simplicidade de geometrias.

Figura 4.17 - Fluxograma Intracamada (etapa 1)



Quanto ao processo Forçar Validade de Geometrias, vale ressaltar que o mesmo possui o seguinte fluxo de trabalho:

a) pré-processo: Identificar Geometrias Inválidas; e

b) pós-processo: Desagregar Geometrias e Identificar Geometrias Inválidas respectivamente.

O pré-processo é executado para identificar as geometrias inválidas. Seu efeito pode ser visto na figura 4.18.

Figura 4.18 - Efeito do Identificar Geometrias Inválidas

(a) Geometria inválida (em verde escuro)

(b) Motivo da invalidade (auto-interseção)

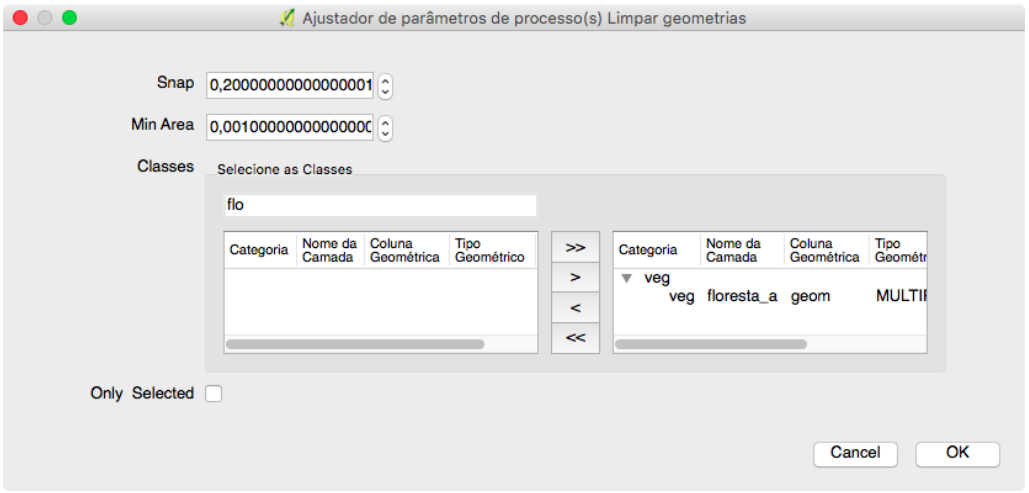
A identificação é feita pois o processo Forçar Validade de Geometrias funciona com base nas *flags* previamente levantadas. Como pós-processo, é executado o Desagregar

Geometrias pois, é possível que as geometrias válidas oriundas do Forçar Validade de Geometrias sejam multiparte.

Em sequência, ao final do processo, é executado novamente o Identificar Geometrias Inválidas para seja possível verificar a existência de *flags* de validade sem a necessidade de intervenção manual.

Finalizando a etapa 1 do fluxo interno, o processo Limpar Geometrias deve ser executado. Os efeitos da aplicação deste processo podem ser vistos na figura 4.19 ao se usar 0,2m como parâmetro para o *snap*.

Figura 4.19 - Efeito do Limpar Geometrias



(a) Janela do processo

(b) Geometria original

(c) Geometria após uso do processo limpar geometrias

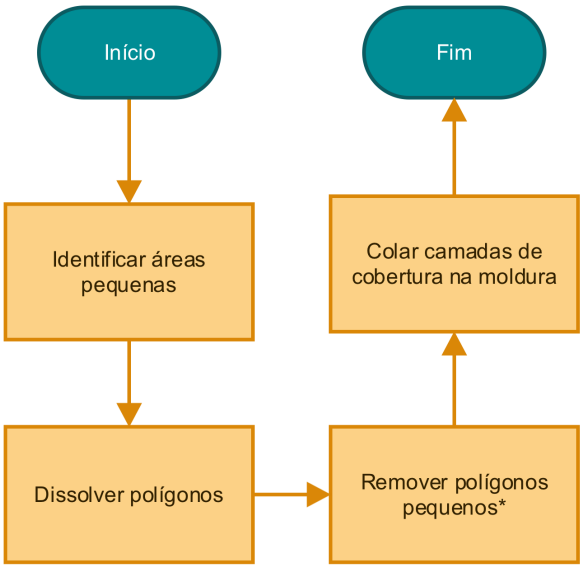
4.2.3.2 Etapa 2

Em seguida, no fluxograma da figura 4.20, são tratados problemas referentes a:

a) Existência de áreas pequenas; e

b) Existência de áreas que deveriam tocar mas não tocam a moldura (opcional para o caso de haver moldura).

Figura 4.20 - Fluxograma Intracamada (etapa 2)



O fluxograma se inicia pela identificação de áreas pequenas para que seja possível verificar quais delas devem ser mantidas, removidas ou dissolvidas. A figura 4.22 mostra o efeito deste processo.

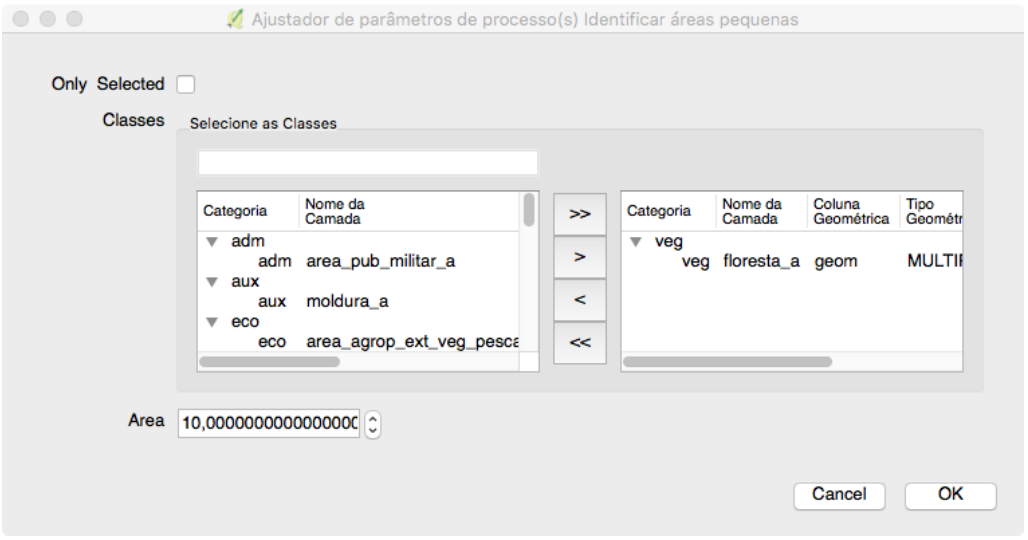
Cabe ressaltar que de acordo com as normas de produção, pequeno pode significar diversos valores de acordo com o tipo de camada e com a escala de representação. No caso da DSG, a produção é pautada pela ET-ADGV (Exército Brasileiro, 2016). Como exemplo, para o caso de feições da hidrografia, pode-se constatar na figura 4.21 os limites que definem o que é pequeno para a escala de 1:25.000 para a camada de hidrografia.

Figura 4.21 - Trecho da Tabela da ET-ADGV Defesa F Ter

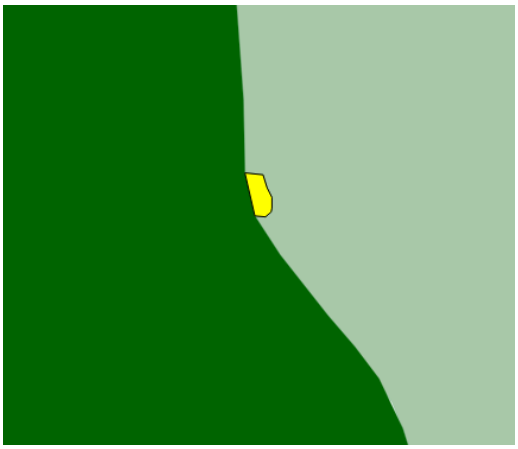
Classe Código na RCO	1:1.000					1:10.000					1:25.000				
	A		L		P	A		L		P	A		L		P
	Larg ≥	Comp ≥	Larg <	Comp ≥		Larg ≥	Comp ≥	Larg <	Comp ≥		Larg >	Comp >	Larg <	Comp >	
1.3.16															
Quebramar_Molhe 1.3.17	2,5mm (2,5m)	5mm (5m)	2,5mm (2,5m)	5mm (5m)	-	2,5mm (25m)	5mm (50m)	2,5mm (25m)	5mm (50m)	-	2,5mm (62,5m)	5mm (125m)	2,5mm (62,5m)	5mm (125m)	-
Queda_Dagua 1.3.18	2,5mm (2,5m)	5mm (5m)	2,5mm (2,5m)	5mm (5m)	X	2,5mm (25m)	5mm (50m)	2,5mm (25m)	5mm (50m)	X	2,5mm (62,5m)	5mm (125m)	2,5mm (62,5m)	5mm (125m)	X
Recife 1.3.19	2,5mm (2,5m)	5mm (5m)	2,5mm (2,5m)	5mm (5m)	X	2,5mm (25m)	5mm (50m)	2,5mm (25m)	5mm (50m)	X	2,5mm (62,5m)	5mm (125m)	2,5mm (62,5m)	5mm (125m)	X
Rocha_Em_Agua (2) 1.5.20	5mm (5m)	5mm (5m)	-		-	20mm (20m)	20mm (20m)	-		X	-		-		X
Sumidouro_Vertedouro 1.3.21	-		-		X	-		-		X	-		-		X
Terreno_Sujeito_ Inundacao 1.3.22	2,5mm (2,5m)	20mm (20m)	-		-	2,5mm (25m)	20mm (200m)	-		-	2,5mm (62,5m)	20mm (500m)	-		-
Trecho_Drenagem 1.3.23	-		-	50mm (50m)	-	-		-	20mm (200m)	-	-		-	20mm (500m)	-
	-		-	< 50mm **(3)**	-	-		-	< 20mm **(3)**	-	-		-	< 20mm **(3)**	-
Trecho_Massa_Dagua 1.3.24	2,5mm (2,5m)	5mm (5m)	-		-	0,8mm (8m)	5mm (50m)	-		-	0,8mm (20m)	5mm (125m)	-		-
Vala 1.3.25	2,5mm (2,5m)	50mm (50m)	2,5mm (2,5m)	50mm (50m)	-	0,8mm (8m)	20mm (200m)	0,8mm (8m)	20mm (200m)	-	0,8mm (20m)	20mm (500m)	0,8mm (20m)	20mm (500m)	-

Fonte: (Exército Brasileiro, 2016)

Figura 4.22 - Efeito do identificar áreas pequenas



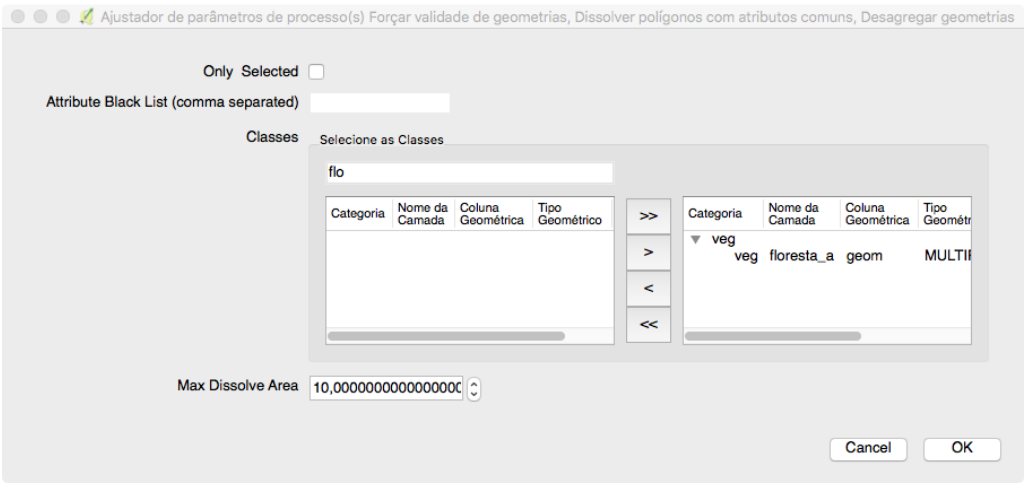
(a) Janela do processo



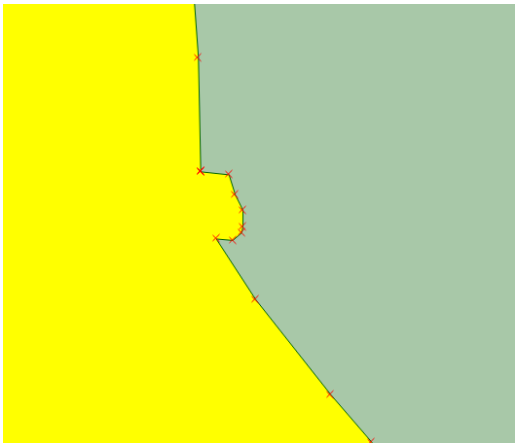
(b) Efeito do processo, selecionando a área em amarelo como pequena

Posteriormente, é executado o processo Dissolver Polígonos Com Atributos Comuns. Com isso as áreas pequenas que sejam vizinhas a áreas grandes que possuem o mesmo conjunto de atributos serão dissolvidas. A A figura 4.23 mostra o efeito deste processo. Vale ressaltar que o processo de dissolução desenvolvido neste trabalho possui o diferencial de poder trabalhar com um parâmetro de área máxima, fazendo com que somente polígonos que possuam área menor que a estipulada sejam dissolvidos.

Figura 4.23 - Efeito do dissolver polígonos



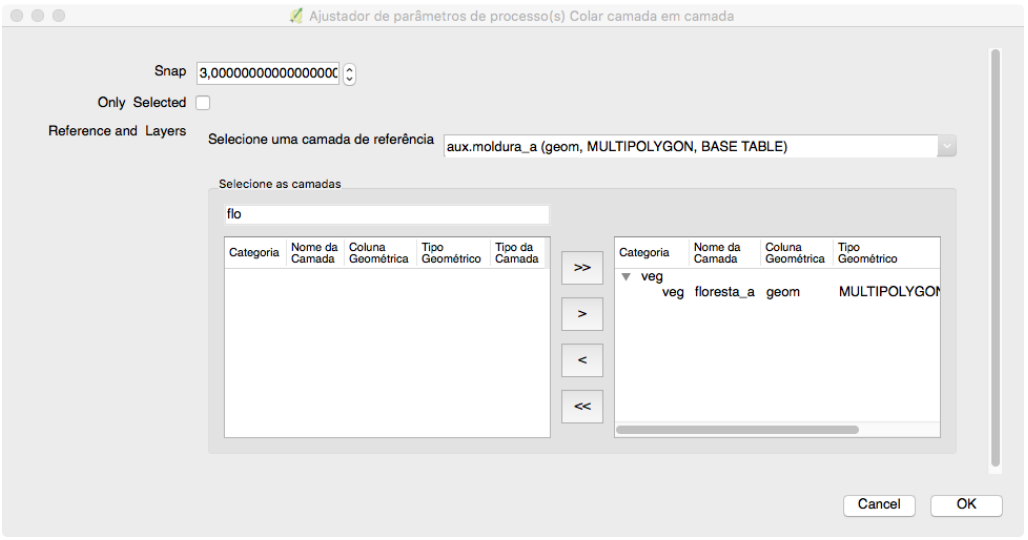
(a) Janela do processo



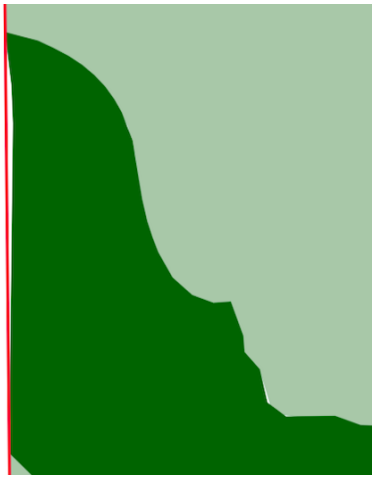
(b) Efeito do processo, a área antes identificada como pequena faz parte do polígono vizinho

Em seguida é necessário remover todas as áreas pequenas que ainda restam por não terem sido dissolvidas, sendo consideradas problemas. Finalmente, caso seja necessário, as geometrias podem ser coladas na moldura com o uso do Colar Camada em Camada. Na figura 4.24 é apresentado o efeito da aplicação deste processo para sanar a falta de conectividade de um polígono de vegetação com a moldura (camada que limita área do caso de teste).

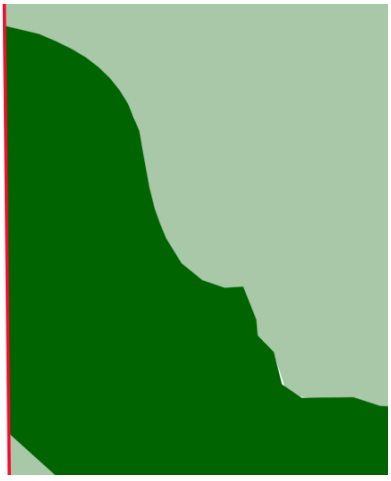
Figura 4.24 - Efeito do colar camada em camada



(a) Janela do processo



(b) Polígono antes



(c) Polígono depois

4.2.4 Fluxograma Intercamadas

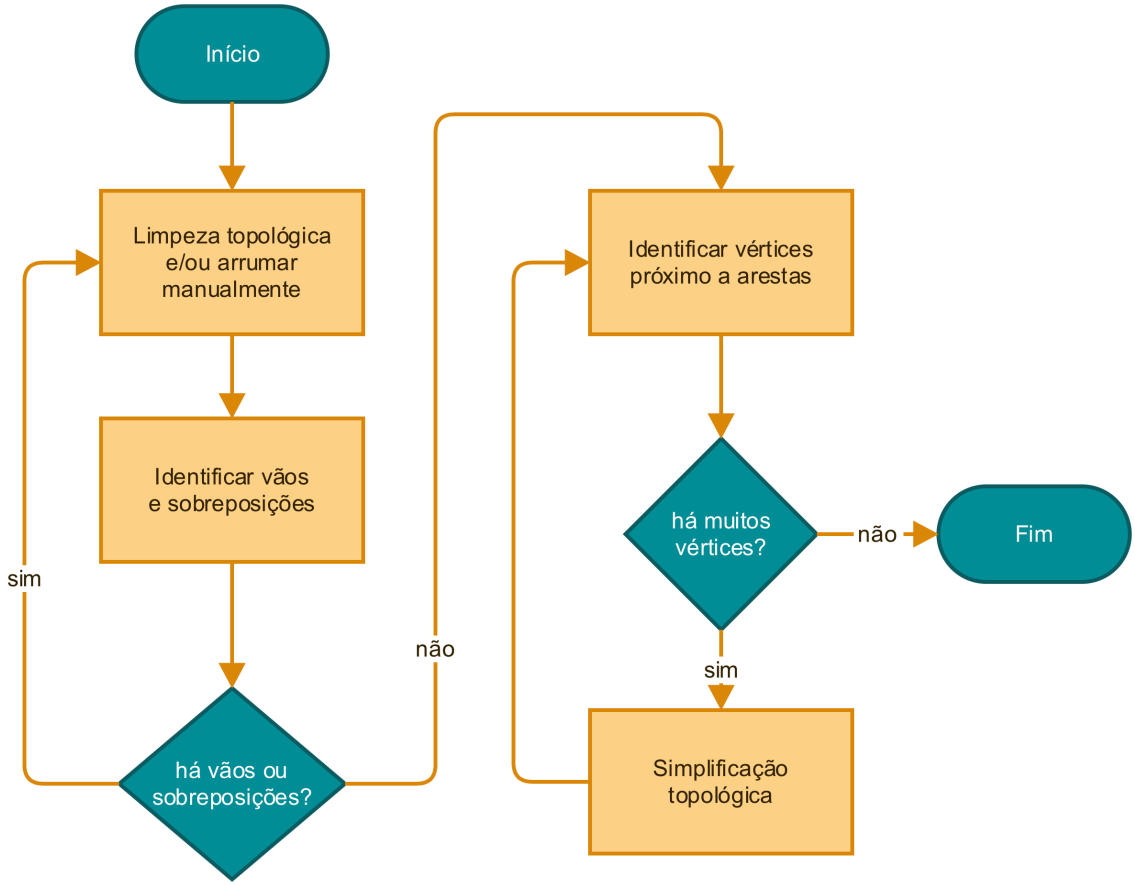
Até o presente momento, os fluxogramas propostos foram usados na garantia de consistência topológica intracamada. Porém, conforme mencionado por Servigne et al. (2000), existem os erros que ocorrem entre camadas. Desta forma, o fluxograma apresentado na figura 4.25 é proposto para tratar questões de:

a) Existência de vãos e sobreposições entre camadas que compõem uma cobertura; e

b) Excesso de vértices que devem ser tratados sem gerar vãos e sobreposições.

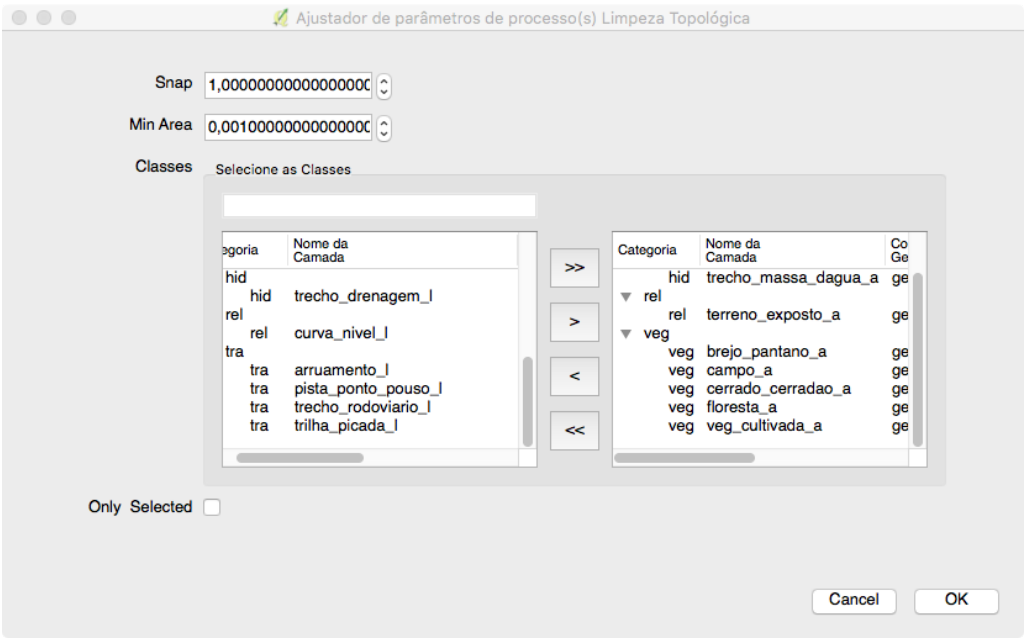
Estes problemas são costumeiramente ligados ao processo de digitalização, conforme apontado por Dangermond (apud MARAS et al., 2010), devendo ser sanados para que seja alcançada a consistência topológica intercamadas.

Figura 4.25 - Fluxograma Intercamadas de validação

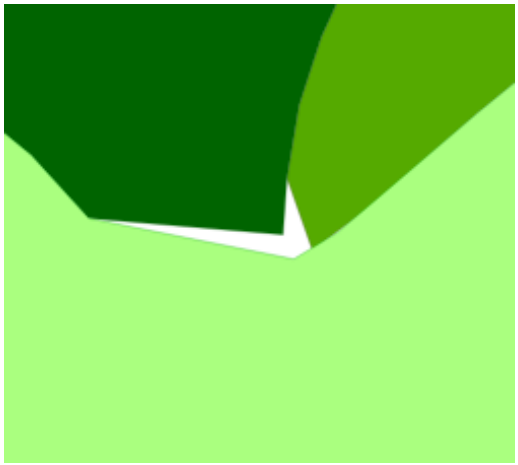


O fluxograma se inicia com o processo Limpeza Topológica, responsável por remover vãos e sobreposições entre as camadas selecionadas. A figura 4.26 mostra o efeito deste processo.

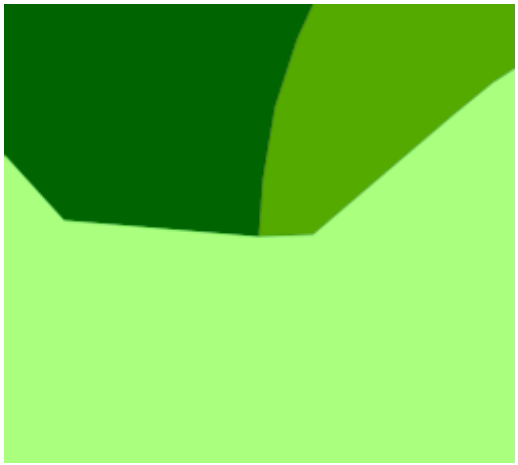
Figura 4.26 - Efeito do limpeza topológica



(a) Janela do processo



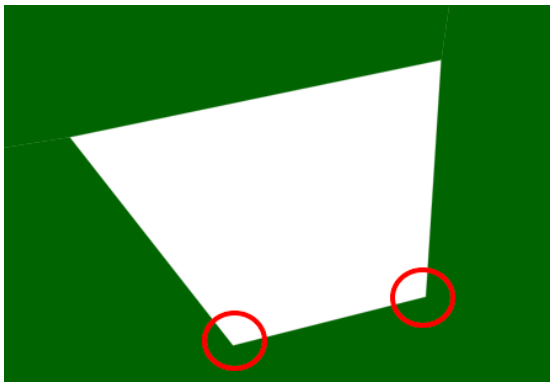
(b) Camadas antes



(c) Camadas depois

Na sequência, o processo Identificar Vãos e Sobreposições deve ser executado para verificar se ainda restam problemas de vão e sobreposições nas camadas analisadas. A figura 4.27 mostra um exemplo de vão que foi identificado pelo processo, que pode, neste caso, ser facilmente reparado por meio da remoção dos dois vértices em destaque.

Figura 4.27 - Vão encontrado na cobertura terrestre.



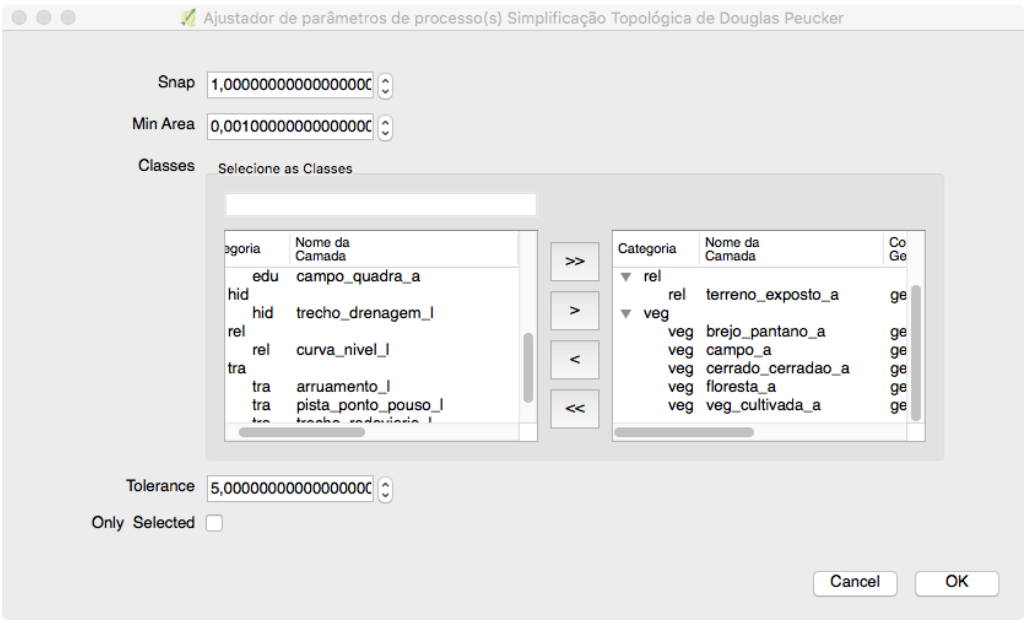
Em destaque, os vértices que podem ser removidos para sanar o problema

O mesmo procedimento pode ser adotado para sanar as sobreposições. O processo somente deve ser interrompido quando não houver mais *flags* após a execução deste fluxo.

Seguindo o fluxograma, o processo Identificar Vértices próximo a Arestas deve ser executado para verificar se há muitos vértices nas camadas. Como já abordado no item Considerações sobre os parâmetros dos processos, o valor para o parâmetro de tolerância do processo deve ser escolhido de forma compatível com a escala do produto para evitar o levantamento excessivo de *flags*. Esta determinação de valores de parâmetros pode ser apoiada em normas técnicas ou pode ser feita de forma empírica, caso não haja norma específica. No caso do Exército Brasileiro, existe uma metodologia de validação na qual, para cada escala, há a determinação de valores de parâmetros que devem ser usados.

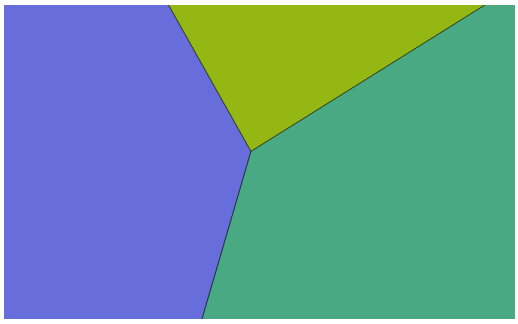
Caso haja muitas *flags* de vértices próximo a arestas, é possível simplificar as geometrias sem que sejam criados vão e sobreposições. Isto é atingido por meio do processo Simplificação Topológica de Douglas Peucker, como se pode ver por meio da figura 4.28.

Figura 4.28 - Efeito do limpeza topológica



(a) Janela do processo

(b) Camadas antes



(c) Camadas depois

Nesta etapa final do fluxograma deve haver uma verificação para se determinar se o resultado do processamento é o esperado para o produto. Esta verificação deve ser apoiada em normas técnicas de qualidade. Caso não haja normas, o controle de qualidade deve ser puramente visual, de acordo com o fim a que se destina o produto. Caso a qualidade seja a esperada, o fluxograma é encerrado, terminando a validação da cobertura terrestre.

4.2.5 Finalização da validação

Para finalizar a validação, como forma de verificação de qualidade, foi constatada a importância de se executar todos os processos de identificação para garantir a qualidade do banco de dados vetorial pós-validação. É possível que, após a validação,

ainda existam problemas registrados por meio de *flags* levantadas pelos processos de identificação. A figura 4.29 mostra um possível erro que pode ser identificado ao final da validação.

Figura 4.29 - *Flag* levantada na verificação final

(a) Geometria inválida (em verde escuro)

(b) Motivo da invalidade (auto-interseção)

Sendo assim, caso o número de problemas seja pequeno, recomenda-se o reparo manual, caso contrário, o banco deve ser processado novamente com uma nova seleção de parâmetros mais adequada, chegando-se, finalmente, ao mostrado na figura 4.30.

Figura 4.30 - Estado das *flags* no final da validação

5 CONCLUSÃO

Neste trabalho foram estudados problemas topológicos que são, comumente, decorrentes dos processos de aquisição vetorial. Os estudos realizados, foram baseados na definição teórica dos problemas abordados para que fosse possível, no decorrer do trabalho, ao entender a origem e o efeito deles, propor soluções tecnológicas para tratar estes problemas.

As soluções tecnológicas foram abordadas no capítudo MATERIAL E MÉTODOS, onde estão presentes os processos de validação que foram implementados no DSGTools (DSG, 2015) para tratar os problemas topológicos apresentados no capítulo REVISÃO BIBLIOGRÁFICA. Cada um dos processos teve seu cerne explicado por meio do uso de pseudocódigos gerados nesta pesquisa (para os processos baseados puramente em código Python) ou por meio de consultas SQL (para os processos que funcionam diretamente no SGBD) para facilitar a replicação do conhecimento em outras tecnologias.

Como resultado do que foi desenvolvido, é apresentada a Caixa de Ferramentas de Validação do DSGTools, que contempla, entre outros, os processos apresentados neste trabalho. É mostrado que o que foi desenvolvido já está disponível para uso da comunidade científica e produtora de geoinformação desde a versão 3.0 do DSGTools. Em sequência, é apresentado um caso de teste onde são apresentadas propostas de fluxogramas de processos de validação para direcionar o processo de Validação de cobertura terrestre para um banco de dados geoespacial vetorial armazenado em PostgreSQL/PostGIS.

Sendo assim, como conclusão deste trabalho, pode-se afirmar que o DSGTools pode ser usado como solução tecnológica para garantia de integridade topológica de dados geoespaciais vetoriais armazenados em PostgreSQL/PostGIS. Isso significa que qualquer aplicação que necessite de ferramentas para validar topologicamente um dado vetorial pode fazer uso do DSGTools. Neste contexto, pode-se incluir aplicações voltadas para geologia, geomorfologia, mapeamento temático em geral, mapeamento cadastral entre outros.

Ademais, o fato do DSGTools ser um *software* livre permite a contribuição da comunidade científica. Uma maior interação pode originar mais ferramentas que possam vir a atender mais amplamente a comunidade como um todo, sendo um passo rumo a um maior conhecimento técnico/científico no campo da qualidade de dados.

Finalmente, cabe ressaltar que o DSGTools é um trabalho em constante evolução e, como linha de pesquisa para trabalhos futuros, sugere-se quatro linhas de ação, a saber:

a) Desenvolvimento do conceito de *workspace*, onde o usuário poderá criar fluxos automatizados dos processos de validação. As ferramentas de *workspace* seriam úteis para, de acordo com a necessidade de cada usuário, criar um fluxo de trabalho específico. Como exemplo inicial, os fluxogramas apresentados neste trabalho poderiam ser disponibilizados como modelos iniciais para consulta e análise;

b) Evolução dos processos existentes para aumento de robustez e melhoria dos algoritmos para aumentar a eficiência temporal, reduzindo o tempo de execução de cada processo;

c) Criação de mais processos de validação de feições lineares, principalmente focadas no conceito de validação de redes, como malha viária e linhas de drenagem; e

d) Criação de um contexto de validação delimitado espacialmente, que permita a um usuário trabalhar somente em um determinado local.

REFERÊNCIAS BIBLIOGRÁFICAS

BANSAL, V. K. Use of GIS and Topology in the Identification and Resolution of Space Conflicts. **Journal of Computing in Civil Engineering**, v. 25, n. 2, p. 159–171, 2011. ISSN 0887-3801. Disponível em: <http://ascelibrary.org/doi/10.1061/{%}28ASCE{%}29CP.1943-5487.0000075>. 15

BARCELOS, T.; SOUZA, A.; SILVA, L.; MUÑOZ, R.; ACEVEDO, R. V. Mensurando o desenvolvimento do Pensamento Computacional por meio de mapas auto-organizáveis: um estudo preliminar em uma Oficina de Jogos Digitais. n. Cbie, p. 932, 2017. Disponível em: <http://www.br-ie.org/pub/index.php/wcbie/article/view/7479>. 2

BRASIL. **Decreto nº 6.666**. 2008. Disponível em: <http://www.inde.gov.br/images/inde/20@Decreto6666{_}27112008.pdf>. 1

CÂMARA, G.; MONTEIRO, A. M. V. Conceitos básicos em ciência da geoinformação. In: **Introdução à Ciência da Geoinformação**. [s.n.], 2001. cap. 2, p. 1–35. Disponível em: <http://www.dpi.inpe.br/gilberto/livro/introd/index.html>. 5

CLEMENTINI, E.; DI FELICE, P. A comparison of methods for representing topological relationships. **Information Sciences-Applications**, v. 3, n. 3, p. 149–178, 1995. 7

CONCAR. **Especificação Técnica para a Estruturação de Dados Geoespaciais Vetoriais (ET-EDGV) versão 2.1.3**. 2010. 246 p. Disponível em: <http://www.geoportal.eb.mil.br/images/PDF/ET{_}EDGV{_}Vs{_}2{_}1{_}3.pdf>. 1

_____. **Plano de Ação para Implantação da Infraestrutura Nacional de Dados Espaciais**. 2010. 205 p. Disponível em: <http://www.concar.gov.br/pdf/PlanoDeAcaoINDE.pdf>. 1

_____. **Especificação Técnica para a Estruturação de Dados Geoespaciais Vetoriais (ET-EDGV) versão 3.0**. 2018. Disponível em: <https://www.concar.gov.br/temp/365@ET-EDGV{_}versao{_}3.0{_}2018{_}05{_}20.pdf>. 2

CORREIA, A. H. Metodologias e Resultados Preliminares do Projeto Radiografia da Amazônia. **Anais XV Simpósio Brasileiro de Sensoriamento Remoto - SBSR**, p. 8083–8090, 2011. ISSN 00223077. Disponível em: <http://www.dsr.inpe.br/sbsr2011/files/indextema.htm{#}tema22>. 1

DANGERMOND, J. A Classification of Software Components Commonly Used in Geographic Information Systems. **Introductory Readings in Geographic Information System**, p. 30–51, 1990. 15, 80

DOUGLAS, D. H.; PEUCKER, T. K. Algorithms for the Reduction of the Number of Points Required to Represent a Digitized Line or its Caricature. **Classics in Cartography: Reflections on Influential Articles from Cartographica**, v. 10, n. 2, p. 15–28, 1973. ISSN 0317-7173. 16, 59

DSG. **DSGTools**. 2015. Disponível em: <https://plugins.qgis.org/plugins/DsgTools/>. 3, 19, 60, 86

EGENHOFER, M. J.; HERRING, J. R. Categorizing Binary Topological Relations Between Regions , Lines , and Points in Geographic Databases. **The**, v. 9, n. 94-1, p. 76, 1990. 7, 8

EGENHOFER, M. J.; SHARMA, J.; HALL, B.; MARK, D. M. A Critical Comparison of the 4-Intersection and 9-Intersection Models for Spatial Relations : Formal Analysis * Models for Topological Relations. **AUTOCARTO-CONFERENCE. ASPRS AMERICAN SOCIETY FOR PHOTOGRAMMETRY AND REMOTE SENSING**, n. 92, p. 1–1, 1993. 6, 7

Exército Brasileiro. Especificação Técnica para a Aquisição de Dados Geoespaciais Vetoriais. 2011. Disponível em: <http://www.geoportal.eb.mil.br/portal/images/PDF/ET{_}ADGV{_}Vs{_}2{_}1{_}3.pdf>. 1

_____. **Norma da Especificação Técnica para Aquisição de Dados Geoespaciais Vetoriais de Defesa da Força Terrestre**. 2ª. ed. [S.l.: s.n.], 2016. 15 p. 67, 75, 76

GRASS Development Team. **GRASS GIS manual: v.clean**. 2018. Disponível em: <https://grass.osgeo.org/grass72/manuals/v.clean.html>. 40, 41

HARDY, P. Active Objects and Dynamic Topology for Spatial Data Re-Engineering and Rich Data Modelling. **Dagstuhl Seminar**, v. 011910, n. May, 2001. 2, 6

HASHEMI, P.; ABBASPOUR, R. A. OpenStreetMap in GIScience. **OpenStreetMap in GIScience**, n. March, p. 19–36., 2015. ISSN 18632351. Disponível em: <http://link.springer.com/10.1007/978-3-319-14280-7>. 9

HOOP, S. de; OOSTEROM, P. van; MOLENAAR, M. Topological Querying of Multiple Map Layers. **European Conference on Spatial Information Theory**, p. 139–157, 1993. ISSN 16113349. 17

ISO. **Geographic Information - Metadata. ISO 19115:2003**. [S.l.: s.n.], 2003. 152 p. 8

JARDIM, J. M. A construção do e-gov no Brasil : configurações político-informacionais. **Cinform**, v. 5, p. 1–25, 2005. Disponível em: <http://www.cinform.ufba.br/v{_}anais/artigos/josemariajardim.html>. 2

KARAS, I. R.; BATUK, F.; ABDUL-RAHMAN, A. AUTOMATED CORRECTION OF TOPOLOGICAL PROBLEMS OF VECTORIZED DATA FOR GIS. **World Engineering Congress**, n. August, p. 3–6, 2010. 11, 13, 14

LAGGNER, B.; ORTHEN, N. A new GIS toolbox for integrating massive heterogeneous GIS data for land use change analysis. **OSGeo Journal**, v. 13, n. 1, p. 78–89, 2012. 11, 13, 15, 71

LAURINI, R. Spatial multi-database topological continuity and indexing: A step towards seamless gis data interoperability. **International Journal of Geographical Information Science**, v. 12, n. 4, p. 373–402, 1998. ISSN 13623087. 17, 18

LEDOUX, H.; ARROYO OHORI, K.; MEIJERS, M. Automatically repairing invalid polygons with a constrained triangulation. **Proceedings of the AGILE 2012 International Conference**, p. 13–18, 2012. 12, 13

MARAS, S. S.; MARAS, H. H.; AKTUG, B.; MARAG, E. E.; YILDIZ, F. Topological error correction of GIS vector data. **International Journal of the Physical Sciences**, v. 5, n. 5, p. 476–483, 2010. ISSN 19921950. Disponível em: <http://www.academicjournals.org/IJPS>. 1, 5, 15, 80

MARTINEZ-LLARIO, J.; WEBER-JAHNKE, J. H.; COLL, E. Improving dissolve spatial operations in a simple feature model. **Advances in Engineering Software**, Elsevier Ltd, v. 40, n. 3, p. 170–175, 2009. ISSN 09659978. Disponível em: <http://dx.doi.org/10.1016/j.advengsoft.2008.03.014>. 43

NGUYEN, T. **Indexing PostGIS databases and spatial Query performance evaluations**. 2009. 1–9 p. 14

OGC. **OpenGIS® Implementation Standard for Geographic information - Simple feature access - Part 1: Common architecture**. [s.n.], 2010. 93 p. Disponível em: <http://portal.opengeospatial.org/files/?artifact{_}id=25355E+Implementation+Standard+for+Geographic+information+-+Simple+feature+access{#}1>. 5, 11, 12

OOSTEROM, P. V.; QUAK, W.; TIJSSEN, T. About invalid, valid and clean polygons. **Developments in Spatial Data Handling**, p. 1–17, 2005. Disponível em: <http://link.springer.com/chapter/10.1007/3-540-26772-7{_}1>. 15, 16, 72

RACINE, P. **Launching the PostGIS Add-ons! A new PostGIS extension for pure PL/pgSQL contributions**. 2013. Disponível em: <http://geospatialelucubrations.blogspot.com.br/2013/11/launching-postgis-add-ons-new-postgis.html>. 27

SERVIGNE, S.; UBEDA, T.; PURICELLI, A.; LAURINI, R. A methodology for spatial consistency improvement of geographic databases. **GeoInformatica**, v. 4, n. 1, p. 7–34, 2000. ISSN 13846175. 6, 13, 79

STOLZE, K. SQL / MM Spatial : The Standard to Manage Spatial Data in Relational Database Systems. **BTW**, 2003. 51

WADEMBERE, L.; OGAO, P. Validation of GIS vector data during geo-spatial alignment. **International Journal of Geoinformatics**, v. 10, n. 4, p. 17–25, 2014. ISSN 16866576. 14