



Universidade do Estado do Rio de Janeiro

Centro de Tecnologia e Ciências

Faculdade de Engenharia

Lais Ribeiro Baroni

**Experimento de Confecção Automatizada de Mapas Temáticos em
Sistemas de Informação Geográfica de Código Aberto e Aplicação ao
Estudo de Despejo de Sal para Degelo em Estradas**

Rio de Janeiro

2019

Lais Ribeiro Baroni

**Experimento de Confecção Automatizada de Mapas Temáticos em Sistemas de
Informação Geográfica de Código Aberto e Aplicação ao Estudo de Despejo de Sal para
Degelo em Estradas**

Projeto Final apresentado, como requisito parcial para obtenção do grau de Engenheira-Cartógrafa, ao Departamento de Engenharia Cartográfica da Universidade do Estado do Rio de Janeiro.

Orientador: Prof. M.Sc. Irving da Silva Badolato
Coorientador: D.Sc. Alvaro Bueno Buoro

Rio de Janeiro
2019

Ficha elaborada pelo autor através do
Sistema para Geração Automática de Ficha Catalográfica da Rede Sirius - UERJ

B266

Baroni, Lais Ribeiro

Experimento de Confecção Automatizada de Mapas
Temáticos em Sistemas de Informação Geográfica de
Código Aberto e Aplicação ao Estudo de Despejo de Sal
para Degelo em Estradas / Lais Ribeiro Baroni. -
2019.

88 f.

Orientador: Irving da Silva Badolato
Projeto Final apresentado à Universidade do Estado
do Rio de Janeiro, Faculdade de Engenharia, para
obtenção do grau de bacharel em Engenharia
Cartográfica.

1. Mapa Temático - Monografias. 2. Sistema de
Informação Geográfica - Monografias. 3. API de
programação Python - Monografias. 4. Degelo de
Estradas - Monografias. I. Badolato, Irving da
Silva. II. Universidade do Estado do Rio de Janeiro.
Faculdade de Engenharia. III. Título.

Lais Ribeiro Baroni

**Experimento de Confecção Automatizada de Mapas Temáticos em Sistemas de
Informação Geográfica de Código Aberto e Aplicação ao Estudo de Despejo de Sal para
Degelo em Estradas**

Projeto Final apresentado, como requisito parcial para obtenção do grau de Engenheira-Cartógrafa, ao Departamento de Engenharia Cartográfica da Universidade do Estado do Rio de Janeiro.

Aprovada em 27 de 06 de 2019.

Banca Examinadora:

Irving S. Badolato

Prof. M.Sc. Irving da Silva Badolato (Orientador)
Faculdade de Engenharia – UERJ

Julia Celia M Strauch

Profª. D.Sc. Julia Celia-Mercedes Strauch
Escola Nacional de Ciências Estatísticas - ENCE

Luiz Henrique Guimarães Castiglione

Faculdade de Engenharia - UERJ

Rio de Janeiro

2019

DEDICATÓRIA

Àqueles que preencheram meu caminho de luz em momentos onde eu parecia não enxergar. Tenho a sorte de ter vários em minha vida. Especialmente mãe Márcia e pai Marcus, minhas lanternas de bolso.

RESUMO

BARONI, L. B. *Experimento de Confecção Automatizada de Mapas Temáticos em Sistemas de Informação Geográfica de Código Aberto e Aplicação ao Estudo de Despejo de Sal para Degelo em Estradas*. 2019. 86 f. Projeto Final (Graduação em Engenharia Cartográfica) – Faculdade de Engenharia, Universidade do Estado do Rio de Janeiro, Rio de Janeiro, 2019.

Esta monografia apresenta o processo de desenvolvimento de um componente de *software* como recurso cartográfico para elaboração de mapas temáticos que dão suporte à análise de impacto ambiental em corpos hídricos pela aplicação de sal nas estradas para o derretimento do gelo, prática utilizada com frequência durante o inverno em países frios. A solução consiste na utilização de Sistema de Informações Geográficas e sua Interface de Programação de Aplicações, adotando a linguagem Python, para concretização de funcionalidades e uma interface gráfica de usuário que atendam as necessidades de criação de mapas temáticos de forma automatizada. A região de estudo considerada para a avaliação experimental da ferramenta trata-se do Lago George, localizado no nordeste de Nova Iorque. O resultado deste trabalho esta concretizado no *plugin* Road Salt Map Builder compatível com o QGIS 3.6. Verificamos as funcionalidades por testes unitários e de integração , bem como validamos a capacidade em atender as necessidades do usuário, ao executar a confecção automatizada de mapas com *plugin* e os dados da região de estudo. Concluímos que a ferramenta opera de modo correto e que cumpre seus propósitos pré-estabelecidos. Os resultados deste trabalho serão divulgados no endereço eletrônico: <<https://github.com/laisbaroni/works/monografia>>.

Palavras-chave: Mapa Temático. Sistema de Informação Geográfica. API de Programação Python. Degelo de Estradas.

ABSTRACT

BARONI, L. B. *Experiment of Automated Confection of Thematic Maps in Open Source Geographic Information Systems and Application to Deicing Salt Spread on Roads.* 2019. 86 f. Projeto Final (Graduação em Engenharia Cartográfica) – Faculdade de Engenharia, Universidade do Estado do Rio de Janeiro, Rio de Janeiro, 2019.

This monograph presents Plugin Road Salt Map Builder as a cartographic resource for the creation of thematic maps that support the analysis of environmental impact in water bodies by the application of salt on roads for the melting of ice, a technique frequently used during winter in cold countries. The solution consists in the use of the Geographic Information System and its Application Programming Interface, adopting the Python language, for the realization of functionalities and a graphical user interface that meet the needs of thematic map creation in an automated way. The region of study considered for the experimental evaluation of the tool is Lake George, located in the northeast of New York. The result of this work is realized in the *plugin* Road Salt Map Builder compatible with QGIS 3.6. We verify functionality by unit testing and integration, as well as validate the ability to meet user needs by performing automated mapping with plugin and study region data. We conclude that the tool operates correctly and that it fulfills its pre-established purposes. The results of this work will be published in the electronic address: <<https://github.com/laisbaroni/works/monografia>>.

Keywords: Thematic map. Geographic Information System. Python API. Road Deicing.

LISTA DE FIGURAS

Figura 1 - Mapa da área de estudo.	20
Figura 2 - Dado de Bacias de Teste	22
Figura 3 - Dado de Estradas de Teste	23
Figura 4 - Esquema de modelo em cascata incremental	25
Figura 5 - Relações das camadas arquiteturais do <i>plugin</i> RSMB no QGIS	26
Figura 6 - Mapa mental de processos do mapa de indicador de risco.	28
Figura 7 - Mapa mental de processos da distribuição de sal.	29
Figura 8 - Mapa mental de dados do mapa IR.	30
Figura 9 - Mapa mental de dados do mapa DS.	31
Figura 10 - Diagrama de atividades para a confecção do mapa IR	35
Figura 11 - Diagrama de atividades para a confecção do mapa DS	38
Figura 12 - Protótipo de interface do <i>plugin</i>	40
Figura 13 - Mapa de Indicador de Risco.	45
Figura 14 - Validação dos resultados para o mapa de Indicador de Risco.	46
Figura 15 - Validação da área do buffer.	47
Figura 16 - Validação das coordenadas do mapa IR.	48
Figura 17 - Mapa Distribuição de Sal.	49
Figura 18 - Validação dos resultados para o mapa Distribuição de Sal.	50
Figura 19 - Validação do cálculo da quantidade de sal despejada no período.	51
Figura 20 - Validação das coordenadas do mapa DS.	52

LISTA DE TABELAS

Tabela 1 - Configuração dos dados de teste	24
Tabela 2 - Rotinas do Mapa IR	37
Tabela 3 - Rotinas do Mapa DS	39
Tabela 4 - Tempos de execução dos processos do mapa IR	53
Tabela 5 - Tempos de execução dos processos do mapa DS	53
Tabela 6 - Tempos de processamento	55

LISTA DE ABREVIATURAS E SIGLAS

API:	Application Programming Interface
GUI:	Graphic User Interface
CGI:	Common Gateway Interface
SIG:	Sistemas de Informações Geográficas
MVC:	Model-View-Controller
RSMP:	Road Salt Map Builder (relativo ao <i>plugin</i>)
PR:	Proxy de Risco (relativo ao mapa)
DS:	Distribuição de Sal (relativo ao mapa)
NWS:	National Weather Service

SUMÁRIO

	INTRODUÇÃO	10
1	FUNDAMENTAÇÃO TEÓRICA	12
1.1	Sal para Degelo em Estradas	12
1.2	Mapas Temáticos	13
1.3	Automatização de Processos em SIG	15
2	CONFIGURAÇÃO DE AMBIENTE EXPERIMENTAL	17
2.1	Requisitos e Proposta de Solução	17
2.2	Ferramentas de Desenvolvimento	19
2.2.1	<u>Opções de software adotadas</u>	19
2.2.2	<u>Configuração de Hardware para os Testes</u>	20
2.3	Região de Estudo	20
2.4	Dados para Testes	21
2.4.1	<u>Características dos Dados</u>	24
3	MÉTODOS	25
3.1	Estudo da solução	27
3.2	Criação de Rotinas	32
3.2.1	<u>Rotinas Comuns aos Dois Mapas</u>	33
3.2.2	<u>Rotinas do Mapa de Indicador de Risco</u>	34
3.2.3	<u>Rotinas do Mapa Distribuição de Sal</u>	37
3.3	Construir e Operacionalizar o <i>Plugin</i> do QGIS	39
3.3.1	<u>Prototipação da Interface Gráfica</u>	39
3.3.2	<u>Dar Funcionalidade à Interface (Ligar Funções)</u>	41
3.3.3	<u>Programação das Restrições</u>	42
3.3.4	<u>Definição de Apresentação dos Mapas</u>	43
4	APRESENTAÇÃO DOS RESULTADOS	44
4.1	Mapa de Indicador de Risco	44
4.2	Mapa de Distribuição de Sal	47
4.3	Análise de Desempenho	53
4.3.1	<u>Teste de Hipótese de Desempenho</u>	54
5	DISCUSSÃO	56
5.1	Aplicações Análogas para Utilização do <i>Plugin</i> RSMB	56
5.2	Questões em Aberto	57
5.3	Proposta de Trabalhos Futuros	58
5.3.1	<u>Utilidades a Serem Implementadas</u>	58
5.3.2	<u>Criação de Mapa Animado</u>	59
6	CONCLUSÕES	61

REFERÊNCIAS	62
APÊNDICE A – Arquivos do python	64

INTRODUÇÃO

A aplicação de sal para degelo de estradas é um procedimento que pode causar impacto ao meio ambiente. O sal se dissolve nas águas superficiais e é arrastado para as águas subterrâneas, prejudicando o ecossistema e o suprimento de água potável. É importante que estudos sejam feitos para dar suporte ao processo de mitigação dessa poluição e ajudar na tomada de decisão no que diz respeito a ação de políticas públicas.

A partir da criação de mapas temáticos com informações relacionadas ao fenômeno a cartografia pode ser uma ferramenta rica para dar suporte a estudos ambientais. Os mapas apresentam o fenômeno espacialmente e permitem uma representação didática do mesmo.

O objetivo deste trabalho é criar uma ferramenta capaz de confeccionar automaticamente mapas que deem suporte ao estudo de impacto ambiental causado pelo sal de degelo em estradas. Os produtos dessa ferramenta tratam-se de dois mapas, sendo o primeiro (aqui denominado mapa de indicador de risco) um mapa com percentuais das áreas em torno de estradas pelas áreas dos polígonos das regiões em estudo (idealmente, sub-bacias de uma bacia hidrográfica) e o segundo (aqui denominado Mapa de Distribuição de Sal) um mapa com a quantidade de sal despejada em cada polígono ao longo do tempo. A análise reproduzida nesses mapas é importante para determinar áreas que estão mais vulneráveis e assim perceber onde devem ser concentrados os esforços para mitigação da poluição.

Uma ferramenta de elaboração automatizada de mapas temáticos pode ser tratada como um componente (*plugin*) em distribuições de *software* de Sistemas de Informação Geográfica (SIG), o que aumenta a disponibilidade da solução e implica em maior integração dos resultados com demais informações a respeito da região em estudo. Isto envolve ainda modelagem do fenômeno em estudo, algo que não será detalhado no presente trabalho, que precede a implementação computacional.

Para a criação dessa ferramenta e a possibilidade de interação do usuário com a mesma, foi utilizada programação na linguagem Python como recurso da Interface de Programação de Aplicativos (API) PyQGIS de forma a automatizar processos no QGIS 3.6. Utilizamos recursos disponíveis para a confecção do *plugin* Road Salt Map Builder (RMSB) e optamos por uma abordagem de desenvolvimento orientado a testes e o padrão arquitetural Modelo-Visão-Controlador (MVC). O principal motivo dessas escolhas de metodologia se baseiam no princípio da possibilidade de desenvolvimento por incrementos e reutilização de código.

A possibilidade de existência de uma ferramenta que permite a automatização do processo de criação de mapas temáticos voltados para o estudo de sal de degelo em estradas é considerada como sendo de grande ajuda como suporte ao estudo de impacto ambiental. Além disso, essa ferramenta pode ser estendida e se tornar útil em outras aplicações. Tendo em vista a grande desmaterialização que ocorre nas áreas de conhecimento como a cartografia em função do avanço tecnológico, faz-se relevante ainda a elaboração de trabalhos que demonstrem a integração entre

as engenharias de computação e cartográfica.

Uma região que sabidamente promove estudos de impacto ambiental causados por sal de degelo em estradas é a região do Lago George, em Nova Iorque. Isto é ponto motivador para o desenvolvimento do *plugin*. Por esse motivo, a prova de conceito com o intuito de testar a ferramenta utizou-se de dados do Lago George para estudar o despejo de sal durante quatro meses do inverno 2017-2018. Os dados vetoriais necessários foram investigados junto à pesquisadores da área e o dado de despejo de sal foi criado de forma sintética. Verificamos e validamos o *plugin* com os dados de teste, provando a funcionalidade da ferramenta e gerando resultados para análise do cumprimento dos seus propósitos pré-estabelecidos. Apontamos também alguns pontos em aberto a serem melhorados e ideias para aprimoramento do *plugin*.

Este trabalho foi dividido em seis capítulos. O primeiro contempla a Fundamentação Teórica para os conceitos usados na confecção deste trabalho; o segundo é dedicado à explicar a Configuração do Ambiente Experimental; o terceiro apresenta os Métodos usados para atendimento ao objetivo, o quarto faz a Apresentação de Resultados; o quinto se dedica a fazer uma Discussão a cerca do potencial para uso da solução e possibilidades de melhorias futuras; e o sexto expõe as Conclusões deste Projeto Final.

1 FUNDAMENTAÇÃO TEÓRICA

A fundamentação teórica traz conceitos relacionados à temática, aos meios (ferramentas e métodos) e objeto do trabalho. Na seção 1.1 é apresentada uma síntese sobre o processo de despejo de material de degelo em estradas e as implicações para o meio ambiente, focando em poluição de corpos hídricos por derivados de sal. Na seção 1.2 apresenta-se a cartografia como ferramenta de suporte a estudos relacionados à poluição causada por sal de degelo. São apresentados conceitos básicos da cartografia, principalmente a cartografia temática. Por último, na seção 1.3 será introduzido o conceito de extensão de ferramentas de SIG para prover novos componentes (*plugins*) e a consequente automatização de processos pelo uso da linguagem de programação Python, objeto do estudo.

1.1 Sal para Degelo em Estradas

Estradas congeladas podem levar a desastres nos países que vivenciam invernos mais rigorosos. As superfícies de estradas congeladas tornam-se lisas e podem causar acidentes graves e acarretar em danos materiais e físicos. Como exemplo, apenas nos Estados Unidos, há uma média de 1.836 mortes e 136.309 feridos por ano devido a estradas com neve e com gelo, segundo dados da Administração Rodoviária Federal do Departamento de Transporte dos Estados Unidos (USDOT) (ICY ROAD SAFETY, 2019). Isso corresponde acerca de 6% das mortes e 7% dos feridos se comparado à média de mortos e feridos em acidentes de trânsito. Se comparado aos outros perigos climáticos, a contagem média de mortes por consequência de gelo nas estradas é de 3,6 vezes todos os outros combinados, entre eles alagamento, relâmpago, tornado, calor etc.

Para evitar danos à propriedade, ferimentos ou morte, é importante que as superfícies das estradas sejam descongeladas. Com o objetivo de manter as estradas pavimentadas limpas e garantir a segurança do trânsito, é um procedimento recorrente em vários países do hemisfério norte a aplicação de sal de degelo (cloreto de sódio) nos meses de inverno.

O sal é aplicado às superfícies da estrada no inverno para interromper o ciclo de congelamento, permitindo a remoção do gelo e neve antes que ele se solidifique nas estradas. No entanto, o sal pode causar impacto ambiental no ecossistema da bacia hidrográfica, pois se dissolve nas águas superficiais e subterrâneas, incorporando-se nos ecossistemas aquáticos e no suprimento de água potável.

Os corpos hídricos são os mais afetados pela poluição pelo sal. O aumento do uso de sal para atenuar o congelamento do inverno causou a salinização generalizada de lagos na América do Norte e na Europa, de forma que estima-se que mais de 7000 lagos dos Estados Unidos podem estar em risco de concentrações de sal elevadas (NOVOTNY; MURPHY; STEFAN, 2008; DUGAN *et al.*, 2017).

Existem campanhas de conscientização sobre o uso do sal e também investimento em pesquisa para procurar substitutos ao sal que causariam menores danos ao meio ambiente. Como exemplo, a Organização de Pesquisa Ambiental *Cary Institute of Ecosystem Studies* lidera uma linha de pesquisa voltada para estudo do impacto ecológico do sal aplicado em estradas. Uma das iniciativas é a publicação de relatórios que trazem informações a cerca do tema. O último relatório, publicado neste ano (2019) destaca a ciência mais recente sobre os impactos ambientais e de saúde da poluição de sal da estrada e fornece uma visão geral das melhores práticas de gerenciamento para minimizar o uso de sal em estradas, mantendo as estradas seguras (KELLY V.R., 2019). Produtos alternativos atualmente disponíveis apresentam um custo muito elevado e são, portanto, raramente utilizados.

1.2 Mapas Temáticos

A cartografia, entre suas diversas áreas e aplicação, visa comunicação cartográfica, ou seja, transmissão de informações por meio de dados cartográficos. Independentemente de todo o rigor necessário para a confecção de um dado cartográfico, a comunicação deve ser feita de forma eficaz e eficiente por meio de mapas com referência aos diferentes grupos de usuários (GUELKE, 1977).

A cartografia temática é o campo da cartografia que tem foco no processo de comunicação, diferindo das outras áreas da cartografia que possuem foco na qualidade geométrica e/ou posicional dos dados geoespaciais (SAMPAIO, 2019). Dentro deste campo estuda-se todas as etapas de produção do mapa levando-se em consideração o perfil do usuário e a natureza e finalidade da informação a ser transmitida.

A cartografia temática pode ser de três tipos, conforme a abordagem e a finalidade do mapeamento: cartografia de inventário, cartografia analítica e cartografia de síntese (MENEZES; FERNANDES, 2016). A cartografia de inventário é definida através de um mapeamento qualitativo e possui uma característica discreta, realizando a representação posicional da informação no mapa. Já a cartografia analítica é definida através de um mapeamento qualitativo e tem seus elementos primários originados de técnicas estatísticas. A cartografia de síntese, por sua vez, tem a finalidade explicativa, em que a representação de fenômenos é realizada mediante as suas relações externas, exigindo um profundo conhecimento técnico dos assuntos a serem mapeados.

Os mapas temáticos podem ser classificados também de acordo com a forma de emprego das variáveis visuais. Alguns tipos são: mapa de símbolos pontuais, mapas de isolinhas e mapas de fluxos, mapas coropléticos, mapas de símbolos proporcionais ou círculos proporcionais, mapas de pontos ou de nuvem de pontos, entre outros (ARCHELA; THÉRY, 2008).

Os mapas coropléticos comportam a representação de valores absolutos e valores relativos a partir de variações visuais de cor (valor, matiz ou saturação) (MARTINELLI, 2003). Neste tipo de mapa, em geral, os dados são “envelopados” por unidade espacial e apresentados por

classes de valores que podem ou não apresentar intervalos simétricos de valores ou de unidades espaciais agrupadas (TOBLER, 1973). Esse tipo de mapa é considerado como um dos mais significativos para representar fenômenos qualitativos ordenados (ARCHELA; THÉRY, 2008).

A cartografia ambiental é considerada por alguns autores uma vertente da cartografia temática pelo fato de reproduzirem uma síntese da situação ambiental de uma determinada área geográfica (MARTINELLI, 1994). Os mapas temáticos são largamente empregados em estudos ambientais por exercerem um importante papel no processo de investigação ambiental (ARCHELA *et al.*, 2002).

A visualização da espacialização geográfica das informações a partir do dado cartográfico permite a detecção dos problemas e potencialidades existentes na área sendo estudada, desde que a modelagem conceitual seja adequada. Logo, o apoio de mapas temáticos proporciona uma base para a otimização do processo de tomada de decisão (SALGADO *et al.*, 2002). Atualmente no Brasil existem propostas metodológicas para a cartografia ambiental com o objetivo contribuir para as discussões sobre as representações cartográficas.

Para a cartografia interessa como os aspectos espaciais mudam com o tempo. Isso porque algumas variáveis ambientais são impermanentes e variam no tempo. Como exemplo, uma carta de representação do tipo de solo ou topográfica, não vai variar significativamente ao longo do tempo, já uma carta que apresente índices pluviométricos ou temperatura vão ser consideravelmente diferentes caso se considere dados de diferentes momentos no tempo.

A maneira de lidar com essas variáveis dinâmicas é a partir da cartografia dinâmica. A cartografia dinâmica tem o objetivo de incorporar o tempo na representação dos fenômenos e é possível atualmente graças aos avanços da geomática, principalmente em ambiente multimídia (MARTINELLI, 2005). Isso porque essas tecnologias possibilitam a utilização de técnicas de representação dessa outra dimensão (o tempo) num documento 2D (já que a representação do espaço já faz uso dessas duas dimensões).

Dentre as técnicas de representação da dinamicidade de certo fenômeno ambiental apoiadas por tecnologias existe, principalmente, a interação e a animação. Mapas animados são caracterizados pela mudança contínua ou dinâmica de fenômenos espaciais enquanto mapas interativos são caracterizados por uma interface intuitiva ao usuário consistindo de ícones gráficos, um dispositivo de apontamento e visualizações instantâneas de mapas (SLOCUM *et al.*, 2008; PETERSON, 1995).

Os mapas animados podem ainda, ser categorizados em: animação por séries temporais, animação em área, animação temática e animação de processo (LOBBEN, 2003). Na animação temática a ênfase é dada à variável, de forma que as localizações permanecem constantes, enquanto o tempo e as variáveis se alteram em seus atributos ou valores (MARTINELLI, 2005). Exemplos comuns de aplicação são sequências temporais de mapas coropléticos para observação de fenômenos numa mesma área de investigação variando no tempo, onde cada mapa representa um intervalo temporal ou um momento estático no tempo.

Nesse contexto, a personalização da visualização por parte do usuário é de extrema

importância. Dependendo da aplicação, resolução e especificidade do estudo o usuário pode determinar os intervalos de tempo que melhor atendem aos seus objetivos. É importante ressaltar que a possibilidade da observação de um fenômeno ao longo do tempo, neste conceito de animação, não está obrigatoriamente relacionado à ideia de “vídeo” ou sobreposição de *frames* (nesse caso, de mapas). A simples observação dos mapas lado-a-lado ou sequencialmente já permite a ideia de dinâmica e percepção da variação espaço-temporal de um fenômeno. Veremos que essa representação é utilizada para a confecção de um dos mapas neste trabalho.

1.3 Automatização de Processos em SIG

Com o desenvolvimento da tecnologia, aumento da disponibilidade de dados, aumento da demanda por produtos cartográficos e ainda a característica de projetos em SIG onde alguns trabalhos são bastante repetitivos, a automatização de processos pode ser de grande ajuda para aumento de produtividade.

As ferramentas SIG permitem condições de maior produtividade para o trabalho se comparadas ao processo analógico de análise espacial e consequentemente da produção cartográfica. Em particular, nas implementações mais abrangentes do mercado como o QGIS e o ArcGIS, existem dois meios principais para automatização de processos. São eles:

- A execução de *scripts* direto na interface da plataforma, a partir do terminal Python.
- O desenvolvimento e uso de *plugins* para aplicações específicas.

Plugins são pedaços de *software* que estendem as principais funcionalidades de ferramentas de SIG pela incorporação de novos algoritmos. Eles criam interfaceamento com técnicas específicas para determinados grupo de usuários, permitindo a aquisição de seus recursos sob demanda, sem necessariamente aumentar o tamanho para distribuição da ferramenta de SIG padrão.

O QGIS é um Sistema de Informação Geográfica livre (*free software*) a aberto (*open source*) disponibilizado sob a licença GNU GPL, portanto pode ser livremente modificado para executar tarefas diferentes de sua configuração padrão e/ou mais especializadas (QGIS PROJECT, 2019; FREE SOFTWARE FOUNDATION, 2019). Por esse motivo também, o *software* possui uma das mais ativas comunidades de usuários da área de Geotecnologias criando uma rede de suporte e compartilhamento de informações relacionadas ao QGIS de forma gratuita. Este software, bem como outras ferramentas de código aberto ou licenciamento livre da atualidade, tem como características marcantes:

- Extensibilidade através de *plugins* em linguagens como o Python e o C++;
- Disseminação de conhecimento pela acessibilidade aos seus algoritmos por seus usuários.

Além do QGIS, outro Sistema de Informação Geográfica amplamente utilizado é o ArcGIS. O ArcGIS reúne um conjunto de ferramentas para criar e usar mapas, compilar dados geográficos, analisar informações mapeadas, gerenciar informações geográficas em um banco de dados, entre outras. Muito se discute na comunidade científica sobre as vantagens do ArcGIS com relação ao QGIS ou vice-versa, mas o fato é que ambos são excelentes ferramentas que compartilham os mesmos propósitos e apresentam muitas semelhanças. O grande diferencial entre elas consiste no fato de que o ArcGIS, ao contrário do QGIS, é um *software* comercial disponível em versões com diferentes níveis de complexidade que podem ser obtidas a partir da compra de licenças. Com relação à extensibilidade, ambos os programas podem ser estendidos a partir de scripts/complementos/plugins/extensões.

O Python é recomendado como recurso no desenvolvimento de *plugins* no QGIS. Para isto é necessário uma API chamada PyQGIS (QGIS PYTHON API, 2019), que é implementada na linguagem Python, embora também sejam suportados *plugins* desenvolvidos em linguagem C++ nesta ferramenta de SIG. A recomendação da linguagem se dá pela maior portabilidade da solução e pela menor curva de aprendizado requerida para que novos colaboradores possam divulgar seus *plugins*.

Python é uma linguagem de programação de alto nível, de desenvolvimento comunitário, aberto e gerenciado pela *Python Software Foundation*. A principal vantagem do Python em relação a outras linguagens, como C e C++, é possuir maior legibilidade e facilidade de aprendizado devido a sua simplificação de sintaxe. Esta é também uma linguagem interpretada e admite programação interativa sendo ideal para prototipação de soluções. Além disto, seu código é eminentemente lógico e as regras da linguagem se alinham com a linguagem humana, trazendo motivação e segurança ao usuário. Como uma linguagem interpretada, o Python não compete por desempenho com linguagens da família C/C++, contudo, quaisquer recursos desenvolvidos nestas linguagens podem ser embutidos no interpretador Python que conta com inúmeras bibliotecas, como a Numpy ou a GDAL/OGR, que roteirizam tarefas de programação de alto uso, reduzindo significativamente o comprimento do código a ser escrito para uma nova aplicação.

2 CONFIGURAÇÃO DE AMBIENTE EXPERIMENTAL

Nesse capítulo são apresentadas características do experimento conduzido neste trabalho. Entre elas estão os requisitos dos mapas que serão produzidos pelo *plugin*, bem como importância no contexto do estudo de impactos causados por sal para degelo em estradas e proposta de solução. São apresentadas também os principais recursos em termos de ferramental de *hardware* e *software* utilizado durante o desenvolvimento do trabalho. Além disso, apresentamos a região de estudo de onde foram compilados dados para confecção de uma prova de conceito.

2.1 Requisitos e Proposta de Solução

As ideias dos mapas propostos neste trabalho surgiram da necessidade de pesquisadores da área em obter esse tipo de informação. No contexto de um pesquisador visando estudar o sal como poluente em uma região de interesse, dentre as primeiras questões a serem levantadas, uma delas é se o sal constitui realmente um problema ambiental nessa região e, se sim, em que proporção. Mais que isso, para o pesquisador interessa saber quais são as regiões mais afetadas e quando o impacto é mais expressivo. Diante destas necessidades essenciais a serem atendidas pela solução, definimos os seguintes requisitos:

- Determinar a proporção de sal numa bacia ou parte desta. O que pode ser derivado da quantidade de estradas e um indicador do raio de potencial impacto do poluente;
- Determinar a expressividade em termos de acúmulo de poluente despejado num dado trecho ao longo de um intervalo.

Um pesquisador comumente trabalha com experimentos, análises, hipóteses e até mesmo com diferentes regiões e condições de pesquisa. Sendo assim, as informações relevantes a serem alcançadas são diversas e variáveis de acordo com a configuração experimental. Levando isso em consideração, a disponibilidade de uma ferramenta que automaticamente gere produtos de análise pode dar assistência ao pesquisador, apoiando e otimizando seus estudos.

Além disso, a proposta aqui apresentada de gerar mapas para representação dessa informação é mais um diferencial. Os mapas têm a capacidade de sintetizar a informação e mostrá-la de forma espacializada. Isso permite ao pesquisador uma leitura mais clara e direta para obtenção do conhecimento útil pretendido.

Ao longo do trabalho, os objetos construídos por meio de automatização de processos dentro da ferramenta de SIG são chamados de mapas. Segundo Casti em 2015, toda representação gráfica que se destina a localizar, planejar ou visualizar dados sobre o espaço ou, o próprio espaço, é entendida como sendo um mapa e pode ser denominada desta forma. Ainda que outros

subprodutos atuais possam não ser entendidos como mapas, a exemplo das geovisualizações de fenômenos em realidade virtual ou aumentada, adotamos tal nomenclatura para os representações gráficas exibidas em tela nas ferramentas de SIG. Para fins de publicação é fundamental ainda que outros elementos sejam incluídos para a caracterização de um mapa, tais quais, projeção, *datum*, escala gráfica, orientação etc.

A ferramenta de SIG adotada neste trabalho é o QGIS, em sua versão 3.6.0, que possui interpretador da versão 3.6.5 do Python. O nome dado a proposta de solução (e ao *plugin*) é “**Road Salt Map Builder**”, que traduzido para o nosso idioma pode ser compreendido como “Construtor de Mapa de Sal em Estrada” e será reduzido neste trabalho pela sigla RSMP nas seções seguintes.

Os mapas produzidos pelo RSMP são dois, sendo assim chamados: **mapa de indicador de risco** e **mapa de distribuição de sal**, que atendem respectivamente aos dois requisitos listados anteriormente. Para simplificar, ao citar esses dois mapas no trabalho usaremos as siglas **mapa IR** para o mapa de indicador de risco e **mapa DS** para o mapa distribuição de sal.

O **mapa IR** trata-se da representação da proporção entre áreas, áreas de estrada por áreas das bacias, e é dado em intervalos de porcentagens. Portanto, esse pode ser considerado um mapa temático estatístico, já que seus resultados derivam de análises estatísticas.

É chamado indicador de risco pois, nesse contexto, as áreas que apresentam maior quantidade de estradas estão mais expostas à poluição pelo sal. Essa análise é importante para determinar áreas que estão mais vulneráveis e assim ter ideia preliminar de onde devem ser concentrados os esforços para mitigação da poluição. Além disso, essa informação espacializada pode ser utilizada em conjunto com outras informações, como a inclinação do terreno, o tipo de solo e/o a proximidade com corpos hídricos para ajudar a prever para onde o poluente está sendo carregado, as possíveis áreas onde ele fica acumulado e os corpos hídricos mais afetados, por exemplo.

O **mapa DS** trata-se de quantidades de sal despejada em cada bacia e constitui, por sua vez, uma representação de distribuição e, portanto, um mapa temático de notação. Esse mapa se diferencia do anterior pois há uma análise temporal envolvida, ou seja, o valor de quantidade de sal representado no mapa é referente a um período no tempo determinado pelo usuário (intervalo).

Sendo assim, a mesma região pode ter diversos mapas derivados dependentes do tempo a ser considerado. Por exemplo, ao se gerar um mapa com o acumulado de sal das semanas do inverno, ao compará-los é possível observar qual semana houve maior acúmulo de sal. Mapas como este podem ajudar no processo de tomada de decisão no que diz respeito a ação de políticas públicas para mitigação da poluição como por exemplo, a aplicação de produtos alternativos, que causem menos danos ao meio ambiente, para o descongelamento de estradas em áreas mais afetadas nos períodos críticos.

2.2 Ferramentas de Desenvolvimento

2.2.1 Opções de software adotadas

Principalmente três classes de ferramentas foram utilizadas durante todo o processo de confecção do *plugin*. A primeira e a principal classe de ferramenta foi o SIG, onde optou-se pelo o QGIS, tanto para interação direta em sua interface gráfica com usuário (GUI), quanto para uso do terminal interpretador de linguagem Python embarcado no software de SIG. A segunda classe de ferramenta foi de editores de texto com realce de código, utilizado para elaboração de estudos e construção de arquivos associados ao componente em desenvolvimento. A terceira classe de ferramentas foi a de *plugins* para desenvolvedores. Foram adotados o *Plugin Builder* e o *Plugin Reloader* que encontram-se entre os recursos disponíveis no repositório de *plugins* do QGIS.

O *Plugin Builder* fornece uma configuração inicial a partir da qual é possível construir novos *plugins* para o QGIS. Para isso, faz-se necessário o preenchimento de algumas informações básicas sobre o *plugin* a ser desenvolvido como nome, descrição e autores. O *Plugin Builder* então cria uma pasta com todos os arquivos necessários para o funcionamento do *plugin* em desenvolvimento e configura sua integração com o QGIS. Ele forma toda a base necessária do *plugin*. Depois disso, basta editar os arquivos para adequar o *plugin* para as funcionalidades desejadas. Também é necessário configurar sua interface gráfica com usuário. Detalhes dos processos codificados e da prototipação de interface gráfica são dados no Capítulo 3 nas seções 3.2 e 3.3.1, respectivamente.

O *Plugin Reloader* é outra ferramenta útil para desenvolvedores que funciona fazendo a atualização do *plugin* em desenvolvimento através de um único e simples clique. Ele possibilita maior eficiência nos testes da ferramenta ao dispensar a necessidade de atualização manual do *plugin* no QGIS.

O editor de texto foi utilizado para a confecção e edição do código. Alternativas ao editor de código embarcado no QGIS foram utilizadas, sendo o GEdit, o notepad++ e também o Geany considerados em diferentes momentos deste trabalho. A utilização de editores de texto com recursos para desenvolvedores é de grande utilidade, uma vez que estes identificam a linguagem, realçam a sintaxe e permitem endentação automática e identificação de caracteres não visíveis, o que é especialmente importante para desenvolvedores em Python, haja visto que endentação é usada para iniciar e concluir blocos de código nesta linguagem.

O código, conforme seu desenvolvimento, foi testado diversas vezes pelo terminal embarcado no QGIS. As rotinas escritas no editor foram verificadas no terminal de forma que se comprovasse o correto funcionamento das mesmas. No terminal, caso não seja possível executar uma instrução, são retornadas mensagens indicando as linhas de código e motivos do erro, o que ajuda o desenvolvedor a identificar as falhas existentes. Como o terminal Python funciona diretamente dentro do QGIS, a visualização das edições é imediata permitindo que haja também validação para determinar se o resultado ficou como o esperado.

2.2.2 Configuração de Hardware para os Testes

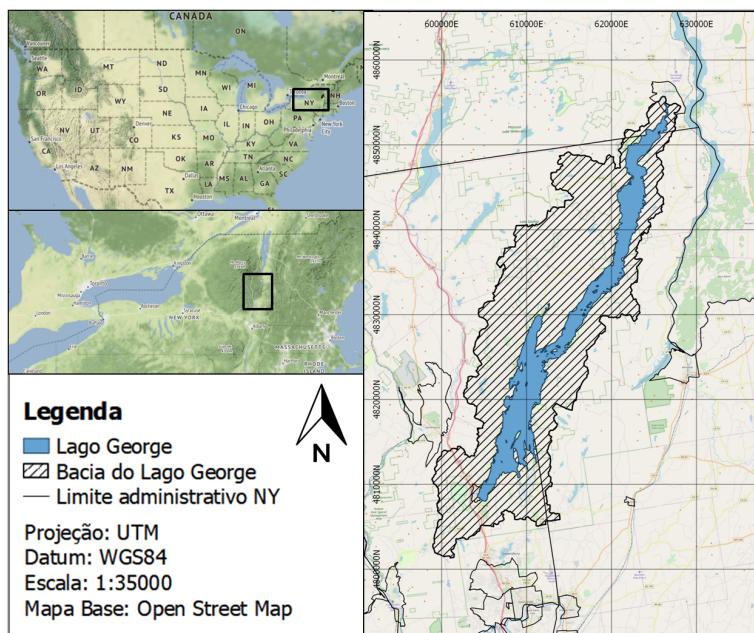
Os testes apresentados na avaliação dos resultados (4) foram executados em uma máquina Windows edição 10 com processador Intel(R) Core(TM) i5, 8,00 GB de memória RAM e sistema operacional 64 bits.

Para constar, o *plugin* foi executado com êxito em máquinas de diferentes configurações e, inclusive, em diferente sistema operacional. As configurações apresentadas não se tratam de requisitos, mas apenas a especificação dos recursos disponíveis que produziram como consequência os tempos de processamento apresentados.

2.3 Região de Estudo

Os dados utilizados para a avaliação dos resultados pertencem à região do Lago George. O Lago George está situado na cidade de mesmo nome no estado de Nova Iorque, Estados Unidos. O lago foi formado por terremotos e geleiras e é estreito e profundo, tendo largura e profundidade médias de aproximadamente 1,33 quilômetros e 21 metros, respectivamente. A área da bacia hidrográfica em torno do lago é cerca de cinco vezes maior que a área do mesmo. A Figura 1 apresenta a localização e dimensões do Lago George e sua Bacia Hidrográfica.

Figura 1 - Mapa da área de estudo.



Fonte: A Autora

Pelo fato do Lago George ser um lago de drenagem e possuir tempo de retenção de água muito longo, a qualidade de suas águas é altamente influenciada pela introdução de elementos

poluentes introduzidos em sua bacia. Um dos poluentes introduzidos na bacia do Lago George é o sal utilizado para a manutenção das estradas em torno do Lago.

Devido ao monitoramento constante, pesquisas sucessivas e cuidado minucioso aplicados às águas do Lago, existem dados capazes de comprovar os efeitos do sal no corpo hídrico. Porém, não foi feita ainda nenhuma análise qualitativa e temporal desses efeitos.

Além dos dados utilizados para testes, que serão apresentados na seção 2.4, alguns outros materiais foram utilizados para compor as camadas de base ou ressaltar a área do lago nas visualizações exibidas neste estudo. Estes materiais serão apresentados nas Figuras 16 e 20 adiante e são listados a seguir:

- arquivo vetorial com o contorno do lago George obtido a partir do site do governo de Nova Iorque (NYS ADIRONDACK PARK AGENCY, 2019);
- arquivo vetorial com os limites dos municípios de Nova Iorque obtido a partir do site da cidade de Nova Iorque (NYC'S WEBSITE, 2019);
- imagem raster de base oriunda da integração do QGIS com o Open Street Map (OSM FOUNDATION, 2019);

2.4 Dados para Testes

Arquivos vetoriais contendo os polígonos das sub-bacias da hidrografia para a região de estudo são parte fundamental para a confecção do resultado. Tendo em vista que empresas atuando no serviço de manutenção de estradas estão reguladas por leis locais que normalmente obrigam o registro de informações sobre despejo de sal, é possível determinar que no mínimo dois outros conjuntos de informações serão usadas como entradas no processo de confecção dos mapas temáticos para salinização. São eles, os arquivos contendo as feições lineares das estradas existentes dentro desta área que receberam sal e informações de despejo de sal ao longo do tempo por unidade de estrada.

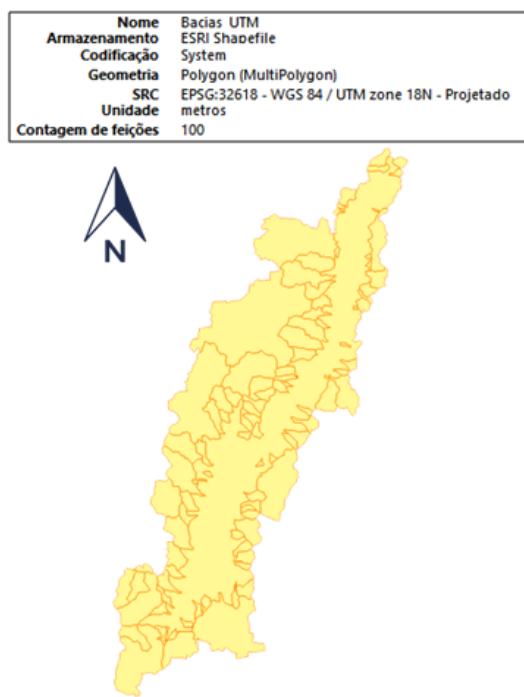
Usamos como entrada para nossos testes o arquivo vetorial de estradas obtido no site <http://gis.ny.gov>. O arquivo, de formato *shapefile*, originalmente encontra-se em WGS84 (coordenadas geográficas) e é disponibilizado sem projeção. Desse arquivo, onde constam todas as estradas do estado de Nova Iorque, foi feito um recorte para a área da bacia do Lago George. Depois disso, foi feita a seleção das estradas estaduais, onde passam os caminhões do governo fazendo o despejo sal e, por último, o arquivo final foi salvo projetado em UTM, Fuso 18N (referente à região onde se encontra o lago).

O arquivo vetorial de bacias adotado foi cedido por pesquisadores da área. Segundo a fonte, anônima por questões de privacidade, o arquivo é derivado de vetorização de uma imagem da região a partir de utilização de recursos do ArcGIS®. A manipulação necessária para preparar

esses dados foi a correção de alguns erros de geometria (como nós duplicados) e ajuste para a projeção adequada.

As informações gerais das camadas vetoriais e a visualização dos arquivos de bacias e de estradas estão apresentadas nas Figuras 2 e 3, respectivamente. Os dados de entrada devem estar projetados em UTM no fuso correto, neste caso, FUSO 18N. A necessidade de trabalhar com camadas vetoriais projetadas em UTM é a de possibilitar que as distâncias possam ser medidas em metros, já que a projeção consiste na mudança da percepção esférica da Terra em um plano com coordenadas métricas.

Figura 2 - Dado de Bacias de Teste



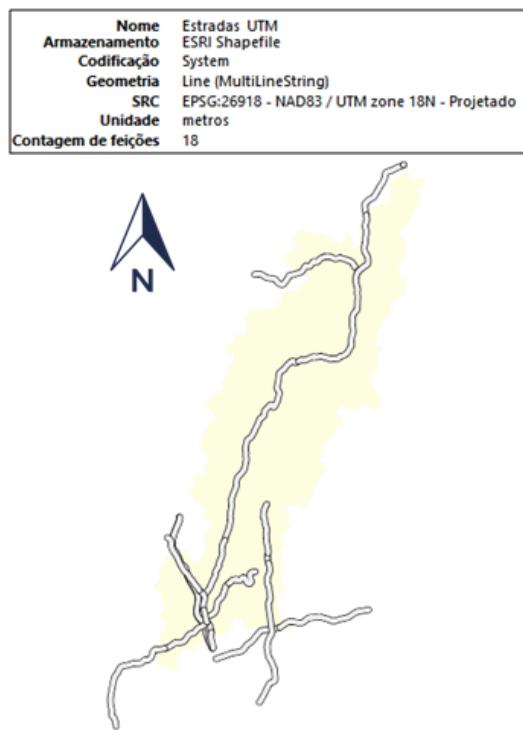
Fonte: A Autora

Por questões de sigilo, a informação de quantidade de sal despejada na bacia não se encontra disponível. Então foi necessário criar um dado sintético para compor o arquivo com as quantidades de sal despejadas a cada dia por unidade de estrada. Para tanto, duas fontes de dados foram usadas para derivação da informação sintética, a saber:

- Registro de precipitação nos meses de inverno para a região;
- Média de quantidade de sal despejado no estado de Nova Iorque.

Os valores das medidas diárias de precipitação (em polegadas) de todos os dias do inverno em estudo foram obtidos a partir de dados de estação meteorológica, disponíveis no site do *National Weather Service* (NWS) (NATIONAL WEATHER SERVICE, 2019). A estação

Figura 3 - Dado de Estradas de Teste



Fonte: A Autora

utilizada, chamada “Rutland - Southern Vermont Rgnl”, está a uma distância de aproximadamente 57 quilômetros do centro do Lago George e, portanto, os valores de precipitação oferecidos são representativos das condições climáticas dentro da região de estudo.

As informações de média de quantidade de sal despejada por unidade de estrada é proveniente de uma das iniciativas do consórcio nacional de pesquisa ClearRoads que tem o objetivo de estudar melhorias e solução para manutenção de estradas no inverno (CLEAR ROADS, 2019). O projeto de pesquisa anual de dados estaduais de manutenção de inverno, promovido pelo ClearRoads, reúne um conjunto de dados e estatísticas sobre mais de 30 estados dos Estados Unidos. Estes dados estão disponíveis abertamente para *download* a partir de uma planilha com as informações do inverno em questão. O projeto propõe que a cada inverno um novo conjunto de dados seja gerado e disponibilizado. Cada estado é representado por um especialista que é responsável pelas informações fornecidas.

Dentre as informações presentes neste projeto existe a informação de quantidade média de sal despejado por milha de estrada durante o inverno. Essa informação está dividida em duas unidades diferentes, um em quantidade de galões por milha (referente ao despejo de sal líquido) e outro em toneladas por milha (referente ao despejo de sal sólido). Para compatibilização dessas unidades foi considerada a concentração de sal na solução líquida (23.3%) e assim, o valor em toneladas correspondente foi somado ao valor dado para o despejo de sal sólido. Devido à indisponibilidade dos dados para o inverno 2017-2018, foram utilizadas as informações do

inverno anterior.

O critério adotado para determinar a quantidade de sal a ser aplicada nas estradas leva em consideração a previsão de precipitação para o dia seguinte. Sabendo disso, para criação do dado sintético a ser utilizado neste trabalho procuramos adotar o mesmo critério de forma que possamos gerar um dado condizente com a realidade. Para tanto, considerou-se a quantidade de precipitação dia a dia durante os meses do inverno no estado de Nova Iorque obtidos do projeto ClearRoads e os valores de precipitação obtidos no site do NWS. Então, distribui-se os valores de quantidade de sal com taxas proporcionais às taxas de precipitação do dia seguinte de forma que a soma de todos os dias coincidisse com o valor da média para o inverno em Nova Iorque.

2.4.1 Características dos Dados

As informações dos dados de entrada apresentadas na Tabela 1, servem de referência para estudos futuros quanto a escalabilidade da solução.

Tabela 1 - Configuração dos dados de teste

	Arquivo de Bacia	Arquivo de Estradas	Arquivo de Sal
Formato	shapefile	shapefile	csv
Tamanho	80,3 KB	58,4 KB	2,02 KB
Nº de Feições/Registros	100	18	121
Número de atributos	2	9	2

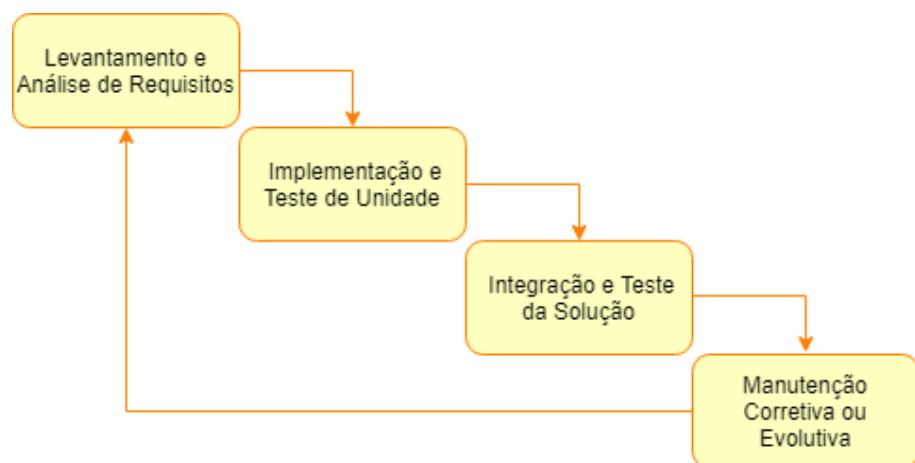
3 MÉTODOS

Neste capítulo são apresentados detalhes sobre a metodologia aplicada para obtenção dos objetivos propostos. De uma maneira prática, apresenta-se essencialmente os processos para o desenvolvimento do *plugin* RSMB. Os processos estão representados de maneira simplificada e são discutidas decisões arquiteturais e práticas de desenvolvimento de *software*.

O conjunto de práticas de desenvolvimento de sistemas adotadas neste trabalho incluem a definição do modelo de processo em cascata incremental a ser utilizado e o reconhecimento da importância da integração contínua. O padrão de arquitetura admitido para o produto final é o MVC (sigla em língua inglesa para *Model-View-Controller*) que divide a solução em três camadas interconectadas.

Deste modo, camadas de modelagem da solução e interface com usuário, além de um controlador para a solução, foram propostas e construídas em diversos incrementos de análise, construção, integração e manutenção para adicionar cada parte necessária ao funcionamento das soluções propostas, conforme indica a Figura 4. Ao refletir sobre cada incremento necessário para atender os requisitos, descritos na seção 2.1, dividimos o produto final em partes menores e projetamos componentes que atuem nas soluções de cada uma destas partes.

Figura 4 - Esquema de modelo em cascata incremental



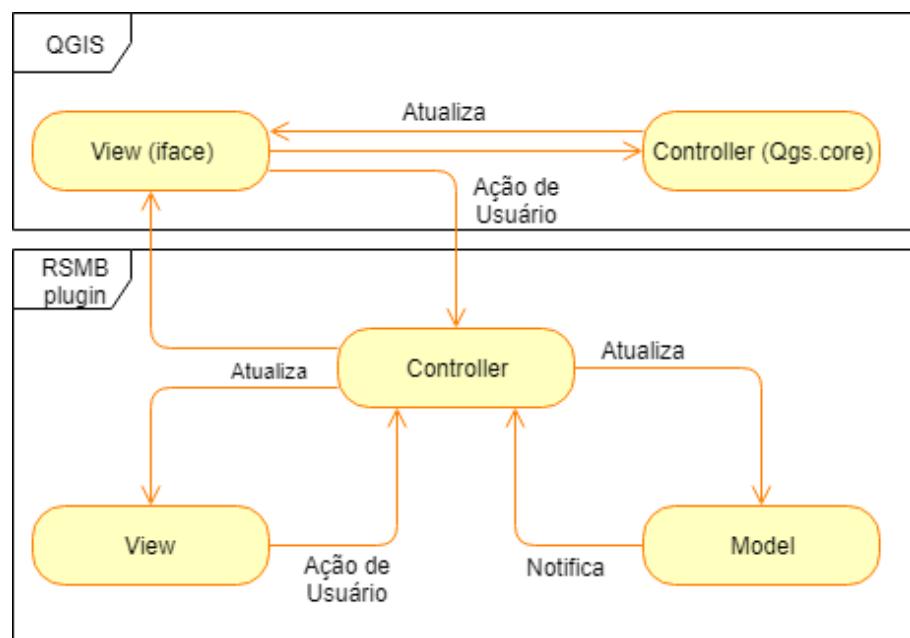
Fonte: Adaptado de (ROYCE, 1987)

A camada de modelagem da solução, ou simplesmente modelo (*Model*), é onde se desenvolvem todas as funcionalidades da ferramenta e, por isso, é considerada a principal estrutura computacional da solução na arquitetura MVC. Em seu código é representado o problema que está se tentando resolver e são aplicadas as regras a serem consideradas para resultar na solução das necessidades do usuário. A seção 3.2 detalha esta camada, sendo voltada para a discussão das técnicas de análise espacial adotadas neste trabalho.

A camada de interface com usuário, ou simplesmente visualização (*View*), é utilizada para receber a entrada de dados e apresentar visualmente o resultado. Uma boa interface deve ser auto-explicativa, induzindo ao usuário sua correta utilização. Na seção 3.3.1 a interface do *plugin* RSMB é apresentada e na seção 3.3.4 é apontado o critério de escolha para a apresentação dos resultados após a execução do *plugin*.

A camada controladora (*Controller*) provê ligação entre as outras duas camadas, ou seja, permite que as ações sejam enviadas para serem executadas ou atualizadas entre o modelo e a visualização. Alterar a camada controladora pode dar novas funcionalidades a uma mesma interface ou acesso por diferentes interfaces para um mesmo modelo. No caso específico do *plugin* em QGIS é a controladora quem liga o *plugin* não somente a própria interface gráfica, como também é que detém acesso a interface gráfica do QGIS. A Figura 5 apresenta a relação das camadas internas ao *plugin* RSMS e demonstra como o *plugin* está relacionado à interface do QGIS (representado pelo objeto *iface*) e por consequência pode provocar ações suportadas pelo núcleo do QGIS (objetos do pacote *core*).

Figura 5 - Relações das camadas arquiteturais do *plugin* RSMB no QGIS



Fonte: A Autora

Cada camada pode ainda ser subdividida em vários pacotes de componentes. Neste trabalho, consideramos apenas um pacote, sendo este todo o *plugin*, com diferentes módulos. Foi construído um módulo para cada camada e diferentes componentes são oferecidos por cada módulo. A saber:

- O módulo de modelo (concretizado no anexo A.1.1), provê principalmente funções como componentes para cada parte necessária à solução;

- Os módulos relacionados as camadas de visualização e controladora (anexos A.1.3 e A.1.2) fornecem classes para instanciação da interface gráfica e tratamento das ações associadas ao *plugin* no QGIS.

Os componentes individuais de nossa solução foram testados independentemente, isolados por unidade recebendo entrada e tendo suas saídas analisadas para conferir o correto atendimento de suas funcionalidades. Posteriormente cada componente é integrado e testado em conjunto com demais componentes. Testes de integração são necessários para observar se a comunicação entre as partes da solução é efetiva. Tal orientação do desenvolvimento aos testes garante que cada componente está operando corretamente, recebendo a entrada adequada e gerando o resultado esperado.

Os testes foram realizados no terminal Python do QGIS durante toda a criação dos componentes da camada de modelo. Estes itens estão relacionados diretamente a confecção dos mapas que atendem aos requisitos. Para testar as operações da interface gráfica com usuário, no entanto, os testes foram feitos na execução do *plugin* de forma que, a cada incremento da solução, o *plugin* era atualizado e testado.

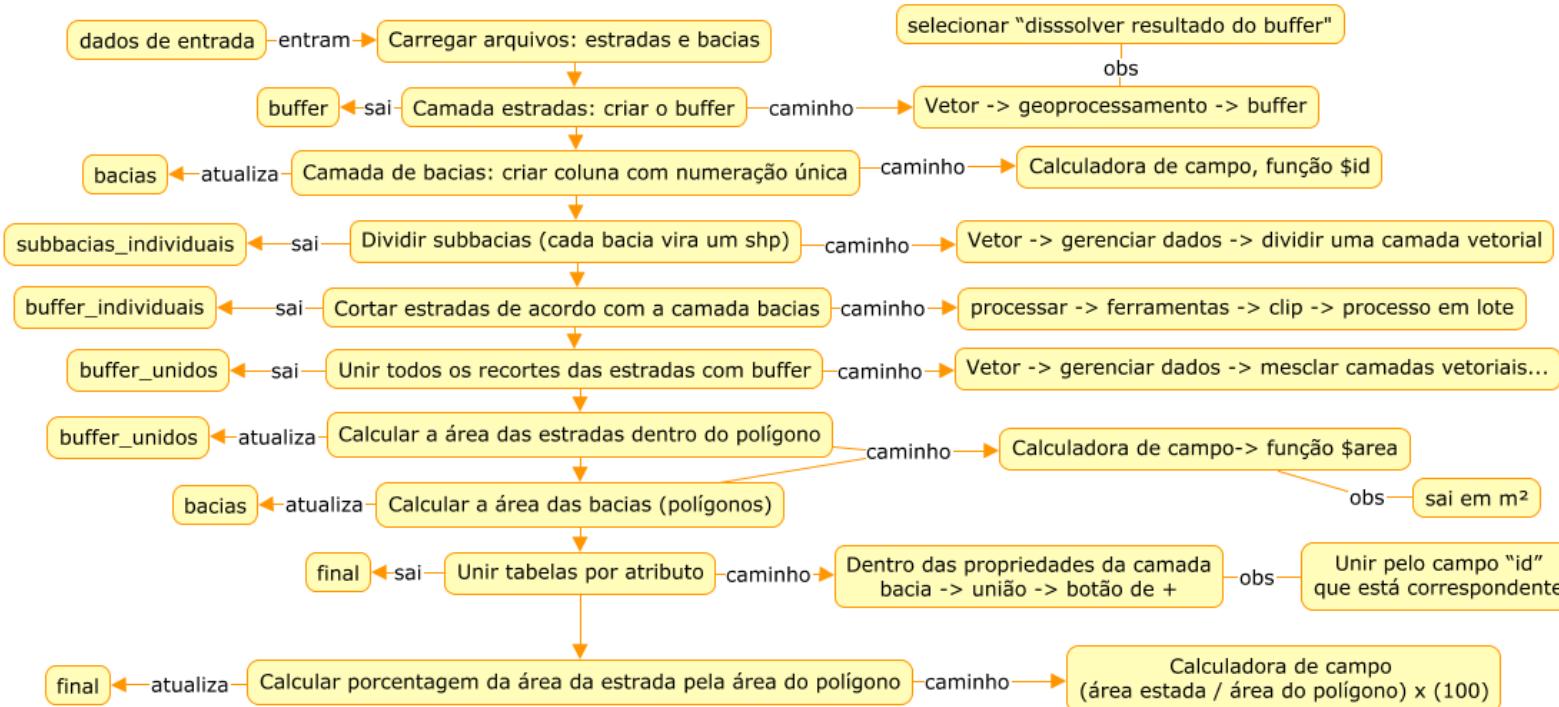
A seção 3.1 apresenta a primeira etapa da metodologia, onde a análise dos requisitos para o processo de confecção dos mapas foi feito de forma manual no QGIS. A seção 3.2 traz a explicação a respeito da implementação de rotinas envolvidas na criação dos mapas. Esta seção aborda sobre a parte de programação e testes de unidades de código voltada para a camada de modelagem da solução. Já a seção 3.3 apresenta as atividades necessárias à confecção das camadas de visualização e do controlador. Nesta seção apresenta-se a prototipação da interface gráfica, e os passos necessários para torná-la funcional, bem como a programação das restrições e a definição do estilo de renderização dos mapas finais carregados em tela.

3.1 Estudo da solução

Visando a confecção da solução no QGIS, adotamos o procedimento de análise espacial para identificar os processos necessários à criação de mapas que atendessem aos requisitos apontados na seção 2.1. Portanto, de maneira manual, através da interface gráfica de usuário, vários testes foram realizados até que se determinasse um conjunto de funções de análise espacial, de seus parâmetros ideias e da sequencia mais adequada a partir dos dados de entrada disponíveis pudesse ser transformados para produzir as saídas temáticas requisitadas.

Foi criado um roteiro para guiar a confecção de dois mapas, sendo um para cada requisito. Nele foram descritos todos os processos utilizados e também os parâmetros empregados. Nesse ponto, foram determinados parâmetros fixos e todos os arquivos foram salvos em uma pasta local, servindo de base para testes futuros.

Figura 6 - Mapa mental de processos do mapa de indicador de risco.



Legenda: Roteiro para confecção do mapa de indicador de risco (mapa IR). Relações:

entra → Evidencia o carregamento de camadas no QGIS

sai → Evidencia a criação de uma nova camadas no QGIS

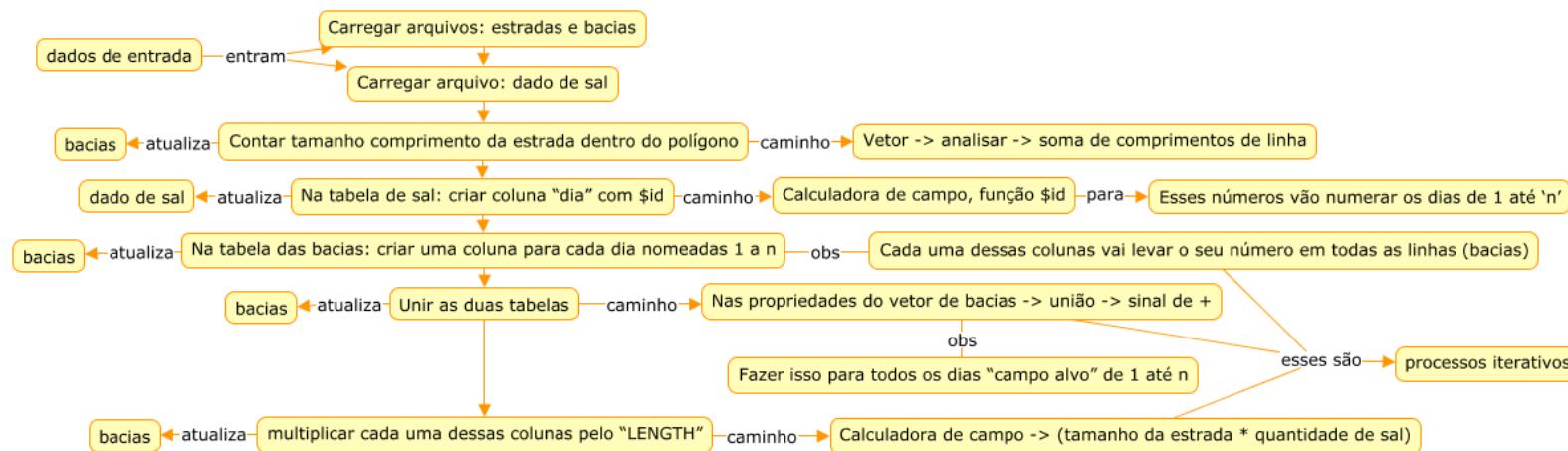
caminho → Evidencia o caminho para um processo acessível no menu do QGIS

atualiza → Evidencia a atualização de uma camada existente

obs → Evidencia observações importantes para o processo executado.

Fonte: A Autora

Figura 7 - Mapa mental de processos da distribuição de sal.



Legenda: Roteiro para confecção do mapa de distribuição de sal (mapa DS). Relações:

entra → Evidencia o carregamento de camadas no QGIS

sai → Evidencia a criação de uma nova camadas no QGIS

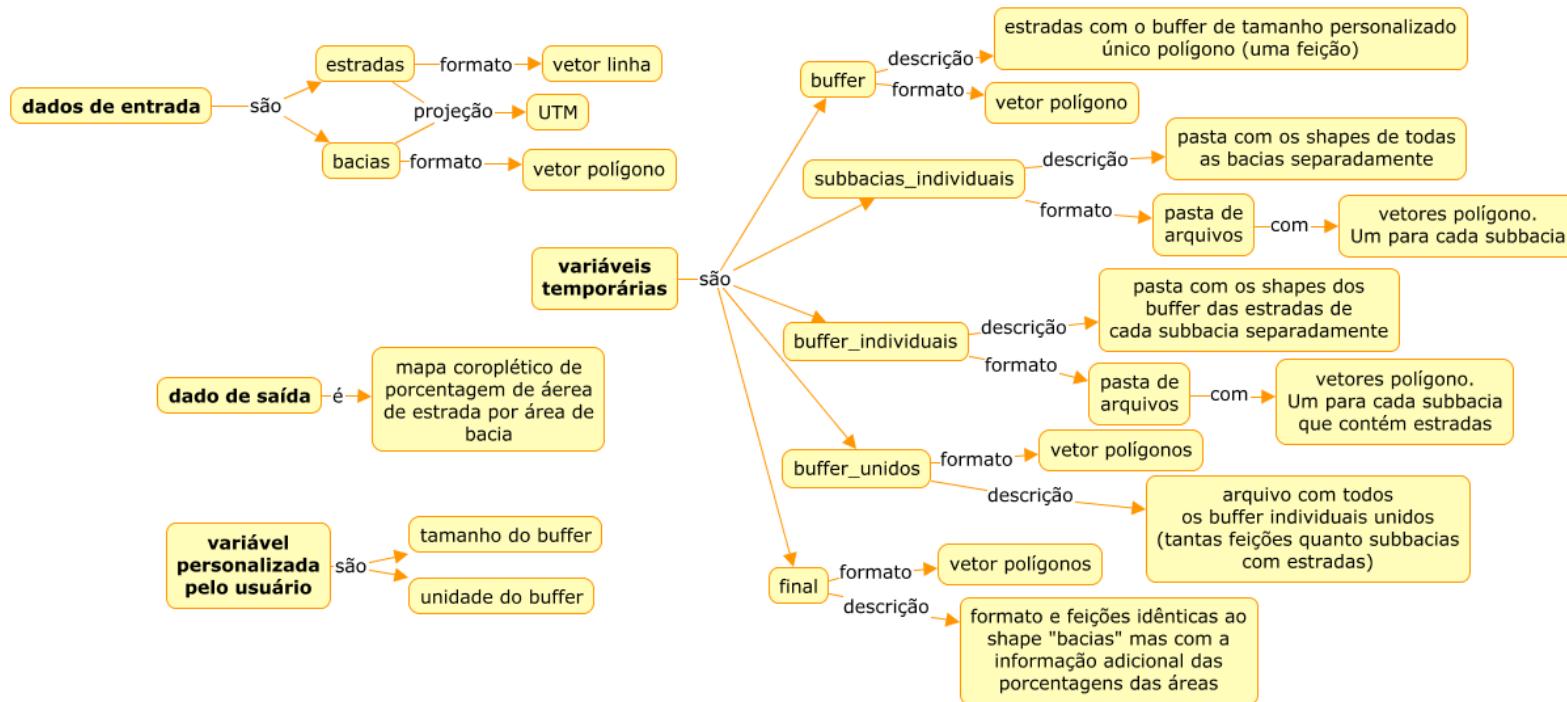
caminho → Evidencia o caminho para um processo acessível pela interface do QGIS

atualiza → Evidencia a atualização de uma camada existente

obs → Evidencia observações importantes para o processo executado.

Fonte: A Autora

Figura 8 - Mapa mental de dados do mapa IR.



Legenda: Dados para confecção do mapa de indicador de risco (mapa IR). Relações:

são/é → Evidencia relação/correspondência

projeção → Evidencia a projeção adequada para carregamento no QGIS

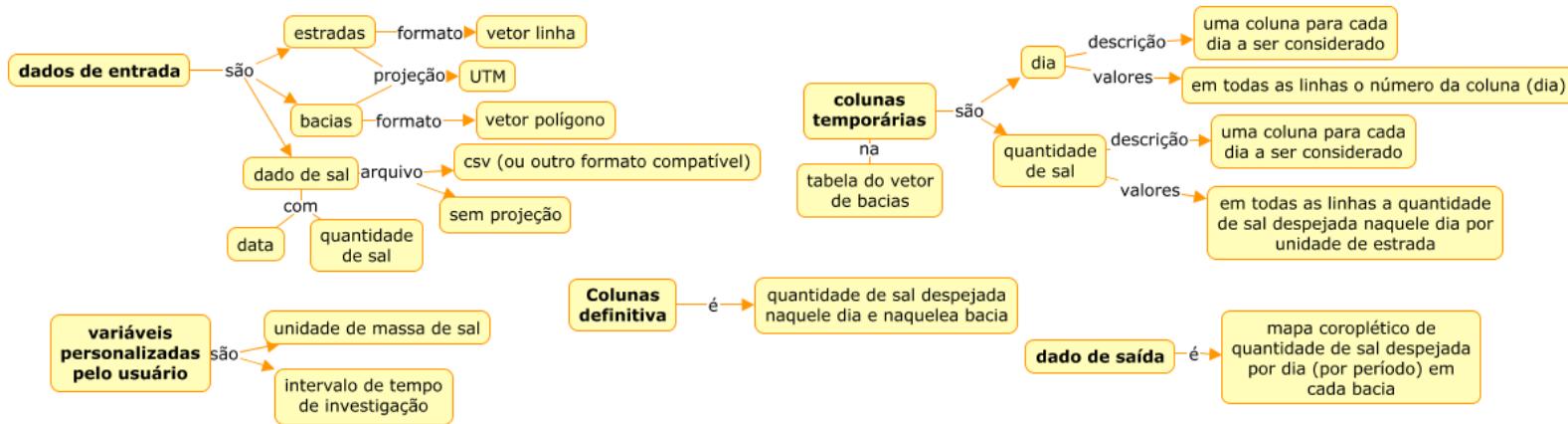
formato → Evidencia o formato do arquivo

descrição → Evidencia uma descrição sucinta do dado

com → Evidencia características do dado.

Fonte: A Autora

Figura 9 - Mapa mental de dados do mapa DS.



Legenda: Dados para confecção do mapa de distribuição de sal (mapa DS). Relações:

são/é → Evidencia relação/correspondência

projeto → Evidencia a projeção adequada para carregamento no QGIS

formato → Evidencia o formato do arquivo

descrição → Evidencia uma descrição sucinta do dado

valores → Evidencia uma descrição dos valores presentes no dado

com → Evidencia características do dado.

Fonte: A Autora

Adicionalmente, para organizar o processo foram criados mapas mentais que compilam informações úteis sobre os processos e encadeiam os mesmos para rápida identificação das partes no roteiro. Os mapas mentais dos processos para a criação do **mapa IR** e do **mapa DS** são apresentados nas Figuras 6 e 7, respectivamente.

Note que para o **mapa DS** a solução encontrada, embora envolva menos processos que o **mapa IR**, é mais trabalhosa para ser executada de forma manual. Para integrar os dados da tabela de sal com os dados do arquivo de bacias foram criadas diversas colunas em um processo exaustivo. Mais tarde no processo de desenvolvimento, como vamos ver na seção 3.2.3, a solução final foi ajustada diferindo desta, pensada inicialmente, já que computacionalmente temos alternativas mais eficientes de executar esse mesmo processo.

Considerando a necessidade de uma abordagem sistemática de descrição foram ainda esboçadas as características dos processos para a criação dos mapas, o próximo passo foi definir então quais seriam os dados de entrada e intermediários necessários, seus formatos, e os parâmetros que seriam posteriormente definidos pelos usuários. As Figuras 8 e 9 mostram essa organização para o **mapa IR** e para o **mapa DS**, respectivamente. Observe que na Figura 9 as colunas intermediárias são necessárias para a abordagem que se pensou inicialmente com os recursos do QGIS. No modelo final a criação destas colunas intermediárias foi descartada.

A etapa de análise, elaboração manual de resultados parciais e caracterização dos processos foi importante para, primeiro, garantir que seria possível criar os mapas desejados com os recursos do QGIS e, segundo, organizar todo o procedimento, mapeando recursos necessários e possibilitando orientar as etapas seguintes da criação e testes do *plugin RSMB*. A seguir, as atividades de construção e testes das rotinas para automatizar desses processos foram feitas incrementalmente, isto é, cada um dos processos listados no roteiro houve um incremento.

3.2 Criação de Rotinas

Uma vez que todos os processos e parâmetros para a criação dos dois mapas foram definidos teve início a programação. O primeiro passo foi a criação de rotinas de modo isolado para entendimento de como cada processo é executado pelo QGIS. Dessa forma, os mesmos processos que foram executados de forma manual na etapa anterior agora foram transformados, de modo incremental, em rotinas na linguagem Python. Inicialmente a integração destas rotinas se deu de forma isolada para o dois produtos **mapa IR** e o **mapa DS**.

Por tratar-se da camada de modelo da arquitetura de *software* escolhida, e pela até então inexistência da camada de interface com o usuário, cada rotina foi executada diretamente a partir do terminal Python no QGIS e os arquivos intermediários foram salvos no disco para verificações posteriores. Como suporte, podemos destacar a utilidade da função *algorithmHelp()* disponível pelo interpretador com o PyQGIS e a documentação do QGIS que apresentam exemplos e o dicionário de parâmetros. Posteriormente as rotinas dos mapas foram integradas num único

módulo Python e anexadas ao pacote do *plugin*.

Para ambos os mapas utilizamos a mesma estratégia de codificação, isolando cada processo interno do QGIS para acionamento isolado em uma de nossas rotinas ou **estágios** (como são referidos em nosso código fonte no anexo A.1.1). Os resultados parciais da solução são então armazenadas em variáveis (**partes**), definindo o conjunto de dados de entradas e saídas usados em nossas rotinas. Todas as rotinas são encadeados em uma sequência por rotinas denominadas **métodos**. Os dois **métodos**, na camada de modelo, implementam efetivamente o todo que é esperado para a execução do processamento que será requisitado pelas demais camadas no *plugin*. Ressaltamos que os arquivos intermediários foram armazenados em memória e não mais em disco, com algumas exceções a serem justificadas ainda nessa seção.

Optamos pelo acionamento dos processos internos do QGIS, sempre que possível, em favorecimento da aceleração da solução, pois como será demonstrado na seção 4.3, os processos internos são potencialmente otimizados para entrega de um melhor desempenho.

Deste modo, salvo em testes e na coordenação de partes iterativas ou na seleção de transformações nossas rotinas se resumem a definição da lista de parâmetros necessários aos processos internos, execução e acesso aos dados para permitir a correta conectividade entre nossas rotinas. Como na maioria dos casos a entrada de uma execução é a saída de uma execução anterior (o conjunto de execuções anteriores), nossa estruturação ajuda a sistematizar os processos de confecção que desejamos automatizar.

Essa seção está dividida em três, uma para cada mapa e uma para demarcar os processos comuns aos dois mapas, onde cada processo é explicado. Pretende-se esclarecer todas as etapas executadas para que se chegue aos mapas finais. Ao fim dessa etapa da metodologia, temos os processos individualizados definidos, integrados e organizados para a execução. A necessidade posterior é associar essas rotinas à interface e aos dados carregados nela, o que será tratado na próxima seção.

3.2.1 Rotinas Comuns aos Dois Mapas

Dois processos são comuns às confecção dos dois mapas: o de carregamento dos dados; e o responsável por exibir o mapa através da interface do QGIS com o estilo aplicado.

O estágio `loadData` é o responsável por atribuir os arquivos de entrada a uma variável identificada, de forma que esses arquivos possam passar a ser chamados pelo seu nome nas funções seguintes. Ele armazena os arquivo de entrada em um dicionário, atribuindo a chaves nominais “bacias” para o arquivo de bacias, “estradas” para o arquivo de estradas e “sal” para o arquivo de sal. Este estágio é o primeiro a ser chamado em cada método. Quando executado para o mapa IR apenas os arquivos de bacias e estradas são recuperados, uma vez que o arquivo de sal não é utilizado.

O estágio `showData` define as configuração de apresentação do mapa em tela. Para

isso, utiliza-se a função *addVectorLayer()* para carregamento do arquivo na tela do QGIS. São definida informações de estilo como escala de cor, tipo de classificação e número de classes para que uma renderização conforme o tema possa ser observada de imediato pelo usuário. Dentro de nossos métodos, esse estágio é chamado por último, depois de ter o arquivo final salvo em disco. Ao chamar *showData()* define-se o arquivo do mapa e a coluna a ser representada através de argumentos de nossa rotina. O que vai ser diferente em cada um dos mapas é justamente o argumento relativo à coluna de dados a considerar para a classificação aplicada. Os critérios escolhidos para adoção do estilo empregado são justificados na seção 3.3.4.

3.2.2 Rotinas do Mapa de Indicador de Risco

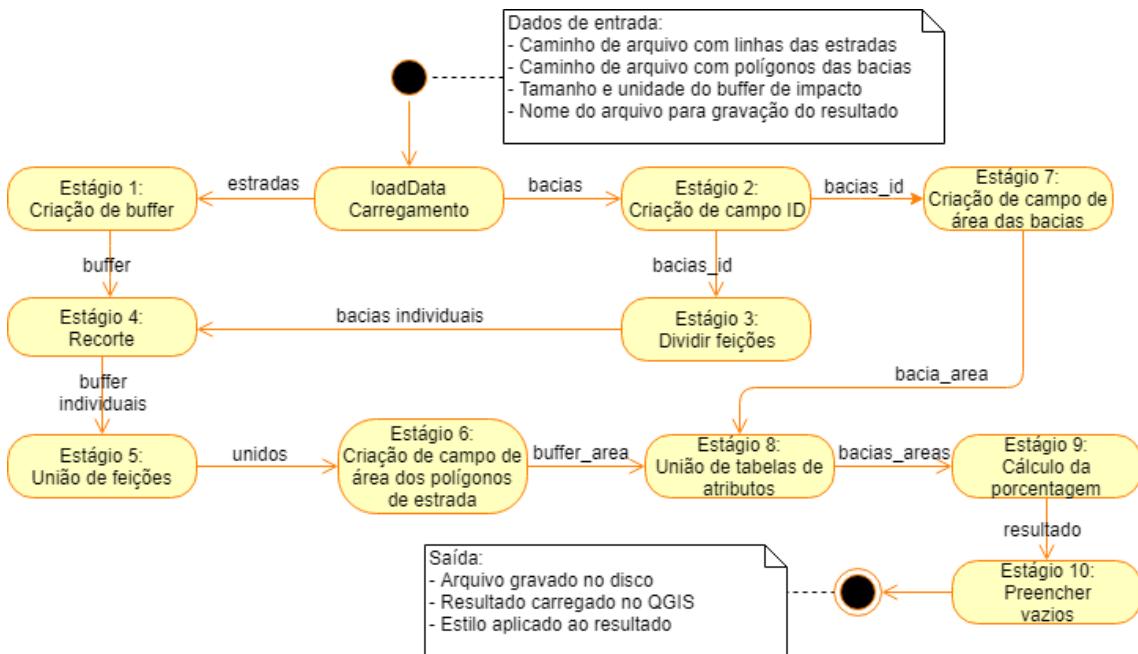
O diagrama de atividades da Figura 10 apresenta os processos executados para a criação do **mapa IR**. O diagrama também apresenta os dados de entrada e saída com algumas informações. Nos próximos parágrafos cada rotina será apresentada para caracterização do processo interno ao QGIS em uso e dos parâmetros relevantes. Para alguns parâmetros adotamos o valor *default* do QGIS. Vamos adotar os nomes indicados no diagrama para acelerar a assimilação do que cada estágio está realizando, contudo, vale observar que em nosso código-fonte eles são identificados por sua numeração sequencial também observável no diagrama. Tal abordagem nos ajuda a mapear que artefato de código está relacionado a qual atividade originária do diagrama de atividades e consequentemente do estudo apresentado por mapas mentais na seção 3.1.

O carregamento no diagrama trata-se da rotina comum já discutida anteriormente. Para o mapa IR este estágio recebe os arquivos de estradas e bacias para que, então, seja dado início aos processos para criação do mapa. Os dados são carregados em memória, mas não são apresentados de imediato ao usuário pela interface gráfica do QGIS.

Na Criação de buffer começam as manipulações no dado carregado em memória. Este estágio chama o processo interno **buffer** do QGIS para criação de um polígono envolvente com vértices que se encontram a uma distância constante, definido como parâmetro tamanho do *buffer*, a partir do vetor de estradas. Como parâmetros, é importante a escolha da opção para dissolver as feições, pois queremos que o arquivo de estradas seja transformado em um único polígono de estrada. O tamanho do *buffer* será oriundo da escolha do usuário na plataforma. Nossa rotina é parametrizada para receber a unidade de medida usada durante o processamento, caso não seja dado adotamos metros como padrão. Realizamos a conversão das unidades antes de acionar o processo interno de computação do *buffer* pelo QGIS.

Na criação do campo ID é manipulado o vetor de bacias. Neste estágio, cria-se uma nova coluna na tabela de atributos que tem como objetivo criar identificadores únicos para todas as feições existentes no arquivo. Para isso, é chamada a calculadora de campo (*field calculator*) que cria um novo campo de inteiros na tabela de atributos ('NEW FIELD': True) de nome "IDapp" a partir da função "\$id". A versão do vetor de bacias atualizada com a nova

Figura 10 - Diagrama de atividades para a confecção do mapa IR



Fonte: A Autora

coluna é salva em memória.

Na divisão de feições o vetor de bacias é dividido em arquivos únicos. O campo que determina a unicidade das funções e utilizado como parâmetro para a função de divisão de feições (*split vector layer*) é o campo “IDapp”. Neste caso, a saída não vai ser um arquivo único, mas um conjunto de arquivos. Não foi possível identificar como manter em memória de forma escalonável estes dados e, então, recorremos por salvar esses arquivos temporariamente em uma pasta de nome “bacias_individuais” no mesmo diretório de *output* definido pelo usuário. Em um processo posterior essa pasta é removida.

O recorte tem como objetivo fracionar o polígono da estrada pela intercessão com cada um dos polígonos no vetor de bacias. Para isso, utiliza-se o resultado criado em criação de buffer e os vetores das feições individuais criados na divisão de feições na função de recortar do QGIS (*clip*). Como queremos manter as informações do arquivo de bacias temos que recortar cada um dos arquivos de bacias individuais pelo arquivo de *buffer*, e não o contrário. É o campo “IDapp”, oriundos do vetor de bacias, que vão permitir a união das tabelas de atributos no união de tabelas de atributos. Assim como na divisão de feições, as saídas são vários arquivos e são salvas em uma pasta nomeada “buffer_individuais” no diretório do *output*. Essa pasta também será apagada posteriormente.

O próximo passo é unir as feições dos *buffer* recortados em recorte. Portanto, em união de feições utiliza-se a função de união de arquivos vetoriais (*merge vector layers*) que tem como entrada todos os arquivos da pasta “buffer_individuais”. Temos ao fim deste

estágio um arquivo de polígonos de *buffer* das estradas onde cada feição representa o recorte do *buffer* para uma única feição do arquivo de bacia. Esse vetor é salvo em memória e chamado de “unidos”. Para os polígonos de bacias que não continham estradas, nenhuma feição de *buffer* correspondente foi gerada nesse arquivo.

Em criação de campo de áreas dos polígonos de estradas e em criação de campo de áreas das bacias criam a informação de área dos polígonos de *buffer* e de bacia, respectivamente. Para isso, é chamada a calculadora de campo (*field calculator*) que cria um novo campo *float* na tabela de atributos ('NEW_FIELD': True) partir da função “\$area”. O nome dos campo criados são “areaBuf” e “areaBac”. Em criação de campo de áreas dos polígonos de estradas é atualizado o vetor “unidos” com a nova coluna em um vetor chamado “buffer_area” e em criação de campo de áreas das bacias é atualizado o vetor “bacias_id” com a nova coluna em um vetor chamado “bacia_area”.

Em união de tabela de atributos é utilizada a função de união de tabelas de atributos (*join attributes table*). O objetivo é trazer para o arquivo de bacias “bacia_area” as informações de área do *buffer* que constam na tabela de atributos do vetor “buffer_area”. A união é feita a partir do campo “IDapp” copiando apenas o campo “areaBuf” e mantendo as feições que não tiveram correspondência. O resultado é salvo em memória em um arquivo chamado “bacia_areas”.

Com o vetor “bacia_areas”, que possui a geometria dos polígonos do vetor de bacias e as informações de áreas dos polígonos e das áreas dos *buffer* das estradas dentro de cada polígono, podemos calcular as porcentagens de área de estrada por área de bacia. Para isso, em cálculo da porcentagem usa-se a calculadora de campo (*field calculator*) para criar um novo campo *float* na tabela de atributos ('NEW_FIELD': True), chamado “areaPor” partir da utilização da fórmula de divisão de áreas dada na equação abaixo:

$$\frac{\text{buffer_area}}{\text{bacia_area}} \times 100$$

Como saída temos o arquivo “bacia_areas” com o campo novo com a informação que deseja-se representar no mapa IR. Esse vetor recebe o nome de “resultado”. Ainda não se trata do arquivo final por um detalhe relevante que é solucionado no último estágio.

O problema do campo criado na união de tabela de atributos é que as feições que não tem nenhuma estrada ficam com valor vazio neste campo. O valor vazio corresponde a falta de informação, mas neste caso sabemos que esses campos, na verdade, devem ser preenchidos com o valor 0. Ou seja, tem 0% de área de estrada nessas áreas de bacia. Os campos sem valores prejudicam a apresentação do mapa ao fim, já que não entra na classificação, e as feições correspondentes acabam por não ser representadas na visualização do mapa.

Portanto, preencher vazios fica responsável por popular esses campos vazios com o valor 0 (zero). Para isso utiliza a função condicional: *if*(“areaPor”*isnull*, 0, “areaPor”). Nesse caso não é criado nenhum campo novo, apenas atualiza-se o campo “areaPor”. A saída

desse estágio é justamente o resultado final desejado. Ele é, então, salvo no formato *shapefile* no destino escolhido pelo usuário.

A Tabela 2 apresenta um resumo dos estágios do mapa IR, trazendo as funções do QGIS que utilizam e os vetores de entrada e saída.

Tabela 2 - Rotinas do Mapa IR

Estágio	função	entrada	saída	armazenamento
1	native:buffer	estradas	buffer	memória
2	qgis:fieldcalculator	bacias	bacias_id	memória
3	qgis:splitvectorlayer	bacias_id	bacias_individuais	disco
4	native:clip	bacias_individuais; buffer	buffer_individuais	disco
5	native:mergevectorlayers	buffer_individuais	unidos	memória
6	qgis:fieldcalculator	unidos	buffer_area	memória
7	qgis:fieldcalculator	bacias_id	bacia_area	memória
8	native:join	bacia_area; buffer_area	bacia_areas	memória
9	qgis:fieldcalculator	bacia_areas	resultado	memória
10	qgis:fieldcalculator	resultado	'definido pelo usuário'	disco

Uma observação é que para a explicação dos estágios, para facilitar entendimento, consideramos que essas definições **estágio** recebessem o(s) arquivo(s) de entrada e já executasse a função. Sabemos, no entanto, que as definições **parte** chamam as definições **estágio** e definem suas entradas e saídas. Outra questão é que os processos são totalmente dependentes da camada do controlador, que faz a ligação das entradas da interface com as variáveis utilizadas nas definições **parte**. Na prática isso não faz diferença, mas é importante na organização do código.

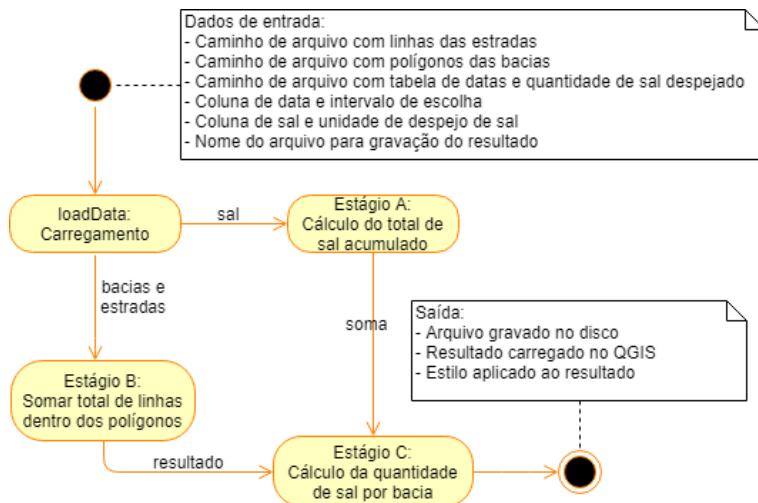
3.2.3 Rotinas do Mapa Distribuição de Sal

O diagrama da Figura 11 apresenta os processos executados para a criação do mapa DS. As definições **estágio** para este mapa são nomeadas de A a C. Nos próximos parágrafos cada estágio será apresentado. Além de determinar a função do estágio vamos apontar os parâmetros relevantes utilizados. Para os parâmetros que não forem citados foram atribuídos o valor *default* do QGIS.

O carregamento no diagrama trata-se do estágio “loadData”, comum entre os dois mapas. Para o mapa DS este estágio recebe os arquivos de estradas, bacias e sal para que, então, seja dado início aos processos para criação do mapa.

No cálculo do total de sal acumulado é calculado o total de sal despejado por unidade de estrada no período determinado pelo usuário. Esse estágio itera no arquivo de sal e é o único a não utilizar nenhuma função do QGIS. É considerada uma variável chamada “soma” que recebe inicialmente o valor 0. É feita uma investigação na coluna de data do arquivo de sal se a data na coluna de data está dentro do intervalo definido pelo usuário. Caso esteja, a variável “soma” vai ser somada ao valor de quantidade de sal correspondente. Isso é feito até

Figura 11 - Diagrama de atividades para a confecção do mapa DS



Fonte: A Autora

que todas as linhas da tabela de sal sejam verificadas.

Tendo definido o valor total de quantidade de sal por unidade de estradas, necessita-se agora obter o comprimento de estrada dentro de cada polígono. Em somar total de linhas dentro dos polígonos é chamada a função do QGIS que recebe de entrada os vetores de estradas e bacias e calcula, para cada feição de bacias, o comprimento total de estradas(*sum line lengths*). Duas colunas são criadas por essa função, uma com número de estradas (feições lineares) dentro do polígono, chamada “NEstr”, e outra com o comprimento total dessas estradas em metros, chamada “tamEstr”. O resultado é salvo em memória em um arquivo de polígonos de nome “resultado”.

Em cálculo da quantidade de sal por bacia cria-se a coluna com as informações a serem representadas no mapa, ou seja, a quantidade total de sal acumulado em cada bacia. Para isso usa-se a calculadora de campo (*field calculator*) para fazer a multiplicação do valor “soma” pela comprimento total de estradas de cada feição armazenado na coluna “tamEstr”. Nesse ponto é importante considerar a unidade definida pelo usuário na interface. Lá existe duas opções de unidade de comprimento: milha e quilômetro. Como “tamEstr” está em metros é preciso fazer a conversão. Como resultado, temos o arquivo vetorial final salvo no diretório definido pelo usuário no campo *output*. É importante frisar que a unidade de massa de sal representada no produto final é a mesma dada na entrada. Por exemplo, caso o usuário tenha definido como entrada a unidade toneladas por milha, a quantidade de sal em cada bacia representada no mapa DS será em toneladas.

Como podemos observar, a confecção do mapa DS envolve muito menos processos que a confecção do mapa IR. A Tabela 3 apresenta um resumo dos estágios do mapa DS, trazendo as funções do QGIS que utilizam e os vetores (ou variável, no caso do estágio B) de entrada e saída.

Tabela 3 - Rotinas do Mapa DS

Estágio	função	entrada	saída	armazenamento
A	-	sal	soma	memória
B	qgis:sumlinelengths	bacias e estradas	resultado	memória
C	qgis:fieldcalculator	resultado e soma	‘definido pelo usuário’	disco

Assim como explicado na seção anterior, as descrições dos estágios para a confecção desse mapa estão abstraídas das questões de implementação na prática, onde as definições de parte e a participação das rotinas do controlador estão envolvidas.

3.3 Construir e Operacionalizar o *Plugin* do QGIS

Nessa seção trataremos da camada de Visualização (Visão), ou seja, a parte do desenvolvimento voltado para dar funcionalidade ao *plugin*. Aqui são contemplados os métodos aplicados para possibilitar que as tarefas definidas na seção 3.2 sejam executadas a partir do acionamento dos campos na interface. Em outras palavras, essa camada irá permitir que os dados de entrada fornecidos pelo usuário na interface sejam recebidos, que as tarefas enviadas pela interface sejam executadas e que exista interação direta do *plugin* com o usuário.

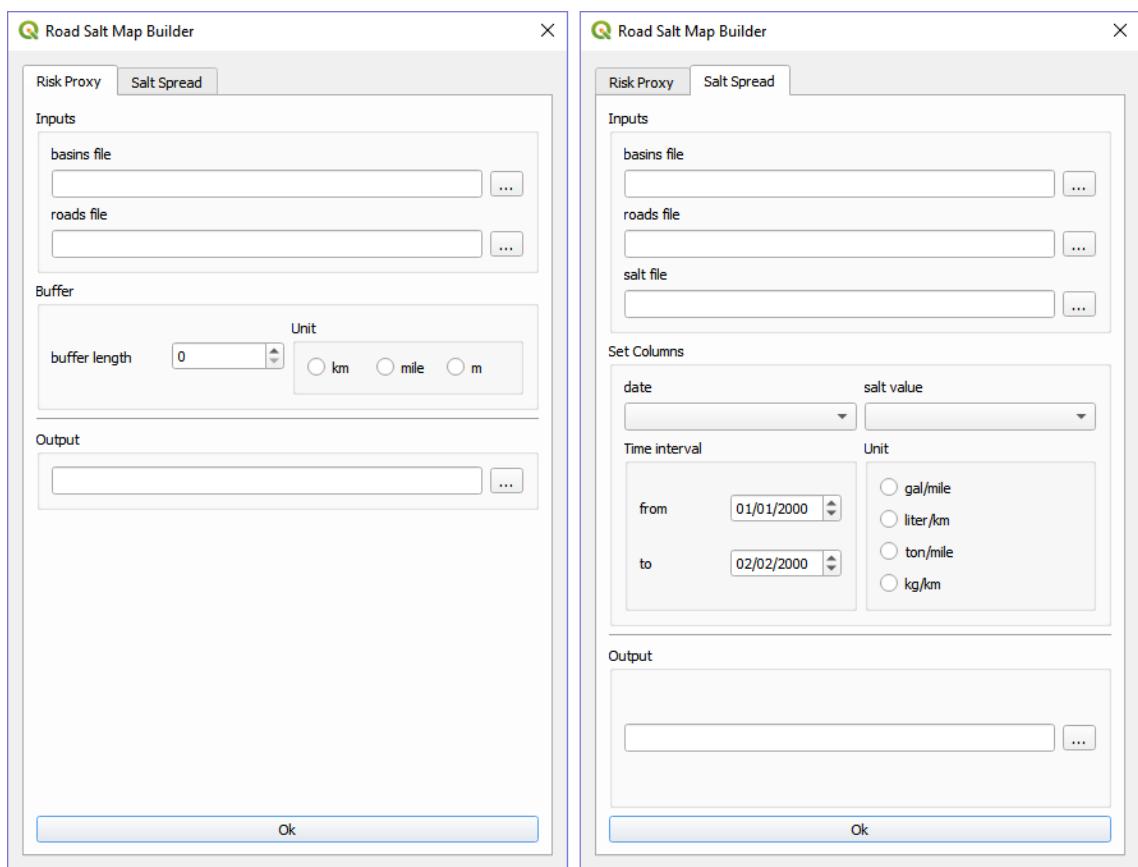
Essa seção é dividida em quatro subseções. A seção 3.3.1 trata-se da apresentação da interface para orientar o entendimento do que ela proporciona ao usuário. A seção 3.3.2 apresenta como é feita a conexão da interface com as funções do *plugin*. A seção 3.3.3 apresenta as medidas de proteção da interface para evitar o mal preenchimento dos campos de entrada e evitar possível *crash* da aplicação. Por fim, a seção 3.3.4 justifica a escolha da aparência definida para a apresentação dos mapas gerados.

3.3.1 Prototipação da Interface Gráfica

Como já comentado na seção 2.2, uma interface padrão foi criada pelo *Plugin Builder*. A partir do Qt Designer, programa multi-plataforma para desenvolvimento de interfaces gráficas do Qt, biblioteca gráfica adotada pelo QGIS, e já integrado na sua instalação, essa interface foi editada de forma que atendesse às necessidades do nosso Plugin. Optou-se por compartimentar a janela em duas abas, onde cada uma é responsável pela criação de um dos mapas. A Figura 12 apresenta a interface do Plugin RSMB.

Como pode ser observado na Figura 12 optamos por colocar os nomes dos campos em inglês. O motivo disto é que pretende-se publicar o *Plugin* Road Salt Map Builder e o inglês é a língua padrão a ser utilizada. É importante destacar que nesta etapa do projeto essa interface ainda não está funcional. É apenas uma interface “oca” que aparece no QGIS ao se carregar o

Figura 12 - Protótipo de interface do *plugin*



Fonte: A Autora

plugin mas não tem nenhuma função habilitada.

3.3.2 Dar Funcionalidade à Interface (Ligar Funções)

Em um arquivo python diferente de onde foram escritas as funções do modelo, as rotinas da camada de visualização foram definidas. É essa camada que permite a ligação dos comandos executados na interface com a execução dos códigos do modelo. O **Plugin Builder** já fornece um modelo padrão desse arquivo, assim como forneceu o modelo padrão da interface. Nesse modelo já vem a parte de programação básica, como as questões de instalação e desinstalação do *plugin* dentro do QGIS.

Os nomes definidos no QT Designer para cada campo presente na interface são utilizados nas definições junto com as funções a serem executadas. Sem riqueza de detalhes, serão apresentadas as rotinas da camada de visualização. Isso inclui o preenchimento, atualização e limpeza dos valores nos campos de linha e combobox, a conexão do clique dos botões com as definições e a execução dos métodos ao dar o comando “Ok” na interface.

Os campos para carregar os dados de entrada são chamados em definições nomeadas “select” onde o campo é atribuído à função *getOpenFileName*. Nessa definição é restringido o formato da entrada como sendo shapefile para o caso do arquivo de bacias e estradas e csv no caso do arquivo de sal. Como os arquivos de bacias e estradas são comuns aos dois mapas, ao escolher esses arquivos em qualquer uma das abas, a linha correspondente na interface é preenchida das duas abas (dois mapas). Isso foi feito para benefício do usuário, que, caso queira os dois mapas, não precisará buscar os arquivos duas vezes.

Outra definição do tipo “select” é aquela para preencher as linhas com os diretórios do *output*. Neste caso a função utilizada é a “*getSaveFileName*” e, ao contrário das seleções de entrada, quando selecionado na aba de um dos mapas, apenas o campo de *output* desse mapa é preenchido. Isso é razoável já que não faz sentido que as duas saídas tenham o mesmo caminho e nome pois uma sobreporia a outra.

Além das linhas de entrada e saída dos arquivos, para o mapa DS as combobox para a escolha dos campos de data e quantidade de sal da tabela de sal e o intervalo de datas também deve ser preenchido. As definições responsáveis por esses preenchimento receberam o nome de “set”. Implementamos essas definições de forma que os campos são automaticamente preenchidos ao se determinar o arquivo de sal. Para atribuir uma coluna no campo de data da interface, procura-se na tabela de sal qual a coluna com o formato ‘data’ . A primeira coluna encontrada é então introduzida no campo da interface. Se não for possível encontrar coluna com esse formato, uma mensagem de erro é apresentada ao usuário. De forma análoga é preenchida a coluna de quantidade de sal, dessa vez procurando pelo formato ‘float’ .

Uma vez que são preenchidos o campo com a determinação da coluna com as datas, os campos com intervalos de datas também têm definições para que sejam preenchidos automati-

camente. O padrão é pegar os intervalo máximo presente na tabela (os seja, o dia mais antigo e o mais recente). Caso o usuário deseje utilizar um intervalo diferente, ele poderá editar na interface.

Implementou-se também uma definição de atualização do campo com intervalo de datas para que, caso o usuário modifique a coluna da tabela de sal onde constam as datas (isso considerando os casos quando o arquivo de sal possuir mais de uma coluna com o formato ‘data’), os intervalos sejam atualizados instantaneamente.

O botão “Ok” de cada aba da interface vai executar as tarefas para a confecção dos mapas a partir das definições “run_method”. Basicamente o que é feito é fazer a chamada dos métodos para o mapa IR e para o mapa DS, que trazem todas as rotinas para a elaboração dos mapas da camada de modelo. Além de chamar as definições `método`, aqui são determinadas as exigências para o preenchimento dos campos (discutido com mais detalhe na seção 3.3.3) e também a verificação das unidades definidas pelos ‘radiobutton’.

Tendo as definições dos preenchimentos dos campos “select” e “set” todas definidas, as definições chamadas “run” são responsáveis por fazer a conexão delas com o clique do mouse. Para isso, utiliza-se a chamada “clicked.connect”. Por último, existe ainda o conjunto de funções responsáveis por limpar todos os campos da interface quando o *plugin* é reiniciado.

3.3.3 Programação das Restrições

Como forma de proteger a ferramenta, foram definidas restrições para as entradas enviadas pelo usuário. Caso as restrições não sejam atendidas, o processo não executa e, ao invés disso, uma caixa de mensagem aparece para o usuário informando quais alterações devem ser feitas.

Os campos protegidos são os de entrada de dados onde são permitidos arquivos não vazios e no formato adequado. O campo de bacias só admite arquivos shapefile e com geometria de polígono. O campo de estradas só admite arquivos shapefile e com geometria de linha e o campo de sal só admite arquivos csv. Para o mapa IR o campo de tamanho de *buffer* não pode estar vazio. Para o mapa DS as colunas são preenchidas automaticamente quando o arquivo de sal é enviado, mas se estiverem vazios também acusa um erro e o processo não executa. O diretório de *output* também deve ser definido nos dois mapas. As unidades, para o *buffer* (mapa IR) e para a taxa de sal (mapa DS), no entanto, não acusam erro ao não ser selecionado nenhuma opção. O algoritmo considera a unidade padrão de metros para o *buffer* de tonelada por milha para a taxa de sal.

Essa etapa é importante para proteger a interface de receber dados incorretos. Caso não houvesse essa verificação, o processamento dos códigos para execução dos mapas seria iniciado sem nenhum critério, comprometendo o desempenho do *plugin* que ficaria totalmente dependente da expectativa de que o usuário tenha inserido os dados corretamente.

3.3.4 Definição de Apresentação dos Mapas

Tendo todo o processo sendo executado para que os dois mapas sejam automaticamente criados diretamente pelo *plugin* RSMB a partir dos dados fornecidos pelo usuário, precisamos apresentar o mapa na janela de visualização do QGIS. Visando os objetivos dos mapas, pretendemos mostrar a informação de porcentagem de áreas (de estradas por bacias) no caso do mapa IR e de quantidade de sal (no período de tempo) no caso do mapa DS.

Optamos por uma aparência padrão pros mapas de forma que o usuário possa interpretar de forma clara e imediata as informações transmitidas. Para tanto, utilizou-se a simbologia de graduação de cores, do branco (valores mais baixos) ao vermelho escuro (valores mais altos) e a classificação em 5 classes definidas por quebras naturais (jenks). A escolha de classificação por quebras naturais foi feita pois nela a variância dentro de cada classe é mínima e a variação entre as classes é máxima, discernindo bem as áreas mais afetadas das menos afetadas. A escolha das cores foi feita pelo fato de que, como os mapas representam áreas vulneráveis, a cor branca, mais neutra, mostra as áreas que estão menos afetadas enquanto a cor vermelha ressalta as áreas mais críticas. Como os arquivos gerados pelo *plugin* RSMP são salvos na pasta de escolha do usuário e as informações geradas estão armazenadas na tabela de atributos dos arquivos, o usuário pode modificar posteriormente o estilo desses mapas como bem desejar.

4 APRESENTAÇÃO DOS RESULTADOS

Nesse capítulo são apresentados os resultados da execução do *plugin* a partir dos dados de teste. Para isso, nas seções 4.1 e 4.2, apresentamos uma feição como exemplo para cada mapa, verificando os resultados parciais e finais dos dados.

Considerando as configurações da máquina utilizada nos testes e as características dos dados de entrada, os tempos de processamento são indicativos da qualidade de escrita do código. Por isso, na seção sec:desempenho, as condições de teste são descritas e os tempos de processamentos para execução do plugin RSMB são mostrados.

Com os exemplos mostrados para cada mapa, provamos o correto funcionamento do *plugin* RSMB na confecção dos mapas IR e DS. Esses exemplos também ajudam no reconhecimento dos processos definidos na seção 3.2, uma vez que ilustra a teoria de forma prática.

4.1 Mapa de Indicador de Risco

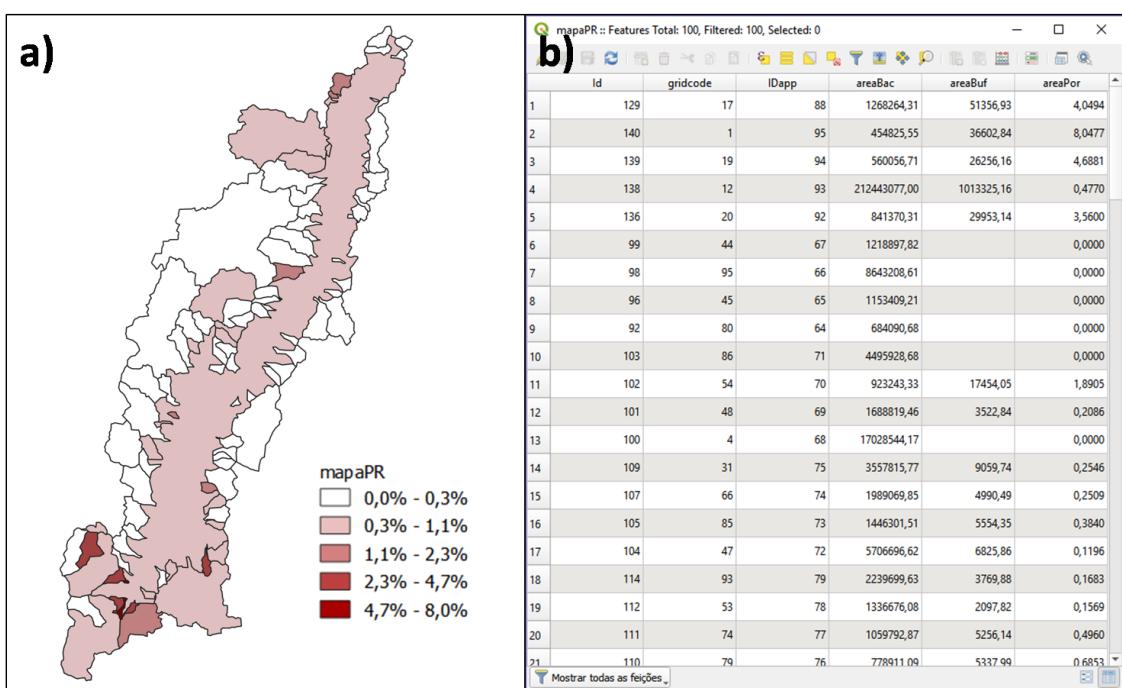
A Figura 13a) apresenta o mapa IR da maneira que é gerado pelo *plugin* e a Figura 13b) sua tabela de atributos. Para gerar esse mapa no *plugin* RSMB utilizamos os dados de teste apresentados na seção 2.4 e o tamanho de *buffer* 10 metros.

Temos seis colunas nesta tabela. As duas primeiras colunas são oriundas do dado de entrada que são mantidos para possível necessidade do usuário. A coluna de nome “IDapp” é a coluna criada pelo algoritmo para definir valores únicos para cada feição. As últimas três colunas se tratam das colunas criadas para armazenar as áreas calculadas. Enquanto a coluna “areaBac” armazena a área de cada feição do arquivo de bacias (ou polígonos), a coluna “areaBuf” armazena as áreas provenientes do arquivo de estradas, considerando o *buffer* definido pelo usuário. Por fim, a coluna “areaPor” mostra o resultado esperado que se trata da divisão das áreas dos *buffer* pelas áreas das bacias. Esta última coluna que é a utilizada para a visualização do mapa carregado na tela do QGIS.

Note que algumas feições não tem valores na coluna “areaBuf”. Isso se dá pelo fato de que por esses polígonos não passava nenhuma estrada e, logo, nenhuma área de *buffer* foi formada. Para a coluna “areaPor” esse resultado vai se propagar, mas ao invés de vazio, as feições vão tomar o valor 0, de acordo com o executado no estágio 10 do módulo de execução para criação do mapa IR. O valor 0 significa que 0% da área da bacia é ocupada por área de estradas.

Como forma de ilustrar e corroborar os resultados calculados, vamos apresentar algumas feições e verificar seus dados. Vamos utilizar o arquivo do *buffer* das estradas, que é gerado pelo *plugin* mas não fica disponível para o usuário, como forma de validar os valores apresentados na tabela de atributos do produto final mapa IR.

Figura 13 - Mapa de Indicador de Risco.

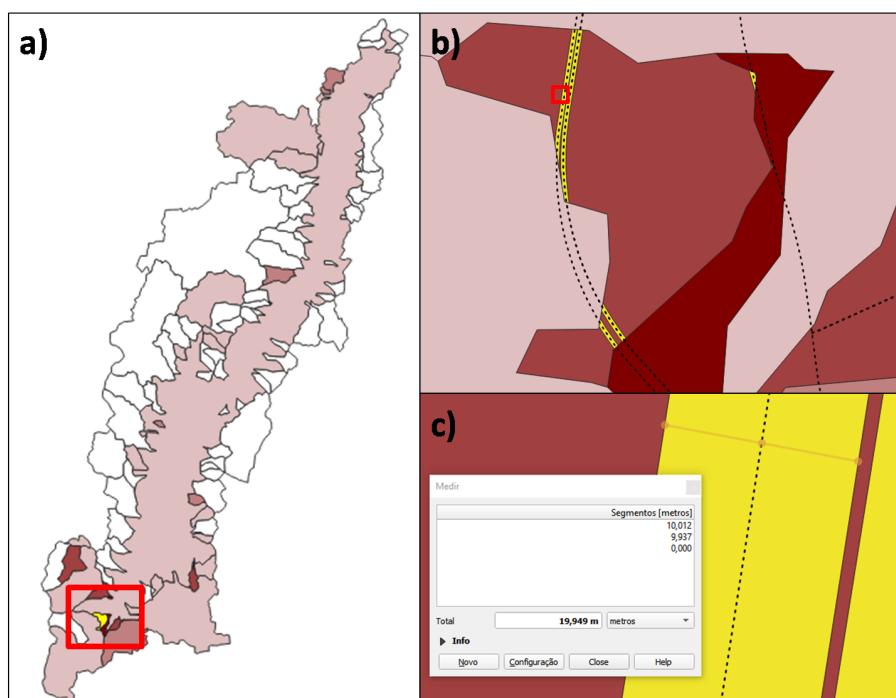


Legenda: Apresentação do mapa IR: (a) mapa renderizado e (b) sua tabela de atributos.

Fonte: A Autora

A Figura 14a) mostra a localização do polígono (bacia) de identificador “IDapp” número 92 que é usado como exemplo para verificar a correta confecção do *buffer* e os cálculos de áreas. A Figura 14b) mostra em amarelo o recorte das estradas já com *buffer* para esse polígono. A linha preta pontilhada é a representação do arquivo de estradas dado com entrada na interface do *plugin*. A Figura 14c) mostra um zoom para um trecho do *buffer* onde foi utilizada a ferramenta de medição de distâncias do QGIS para medir aproximadamente a distância da linha da estrada para as extremidades do *buffer*. Podemos conferir o valor aproximado de 10 metros para cada lado, de acordo com o que foi escolhido na interface.

Figura 14 - Validação dos resultados para o mapa de Indicador de Risco.



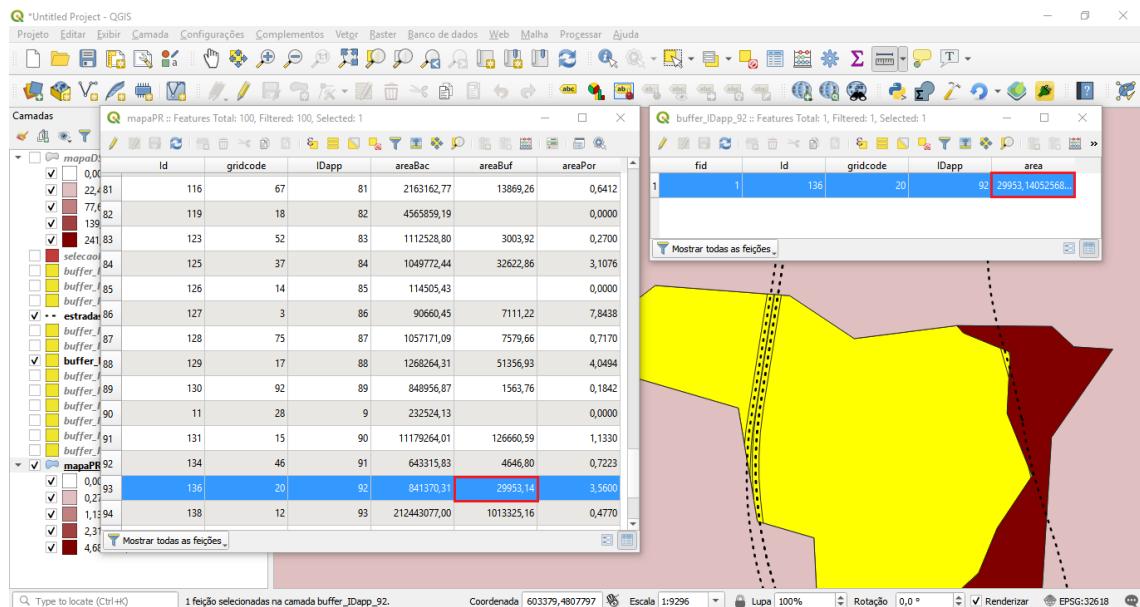
Legenda: Imagens para apoiar a validação da correta execução do *plugin* na confecção do mapa IR: (a) região total, (b) recorte para feição de bacia e (c) medição do tamanho do *buffer*

Fonte: A Autora

Agora, vamos conferir a área desse *buffer* e checar se coincide com o valor reproduzido na tabela do arquivo do mapa IR. Para calcular a área, usamos a calculadora de campo no recorte de *buffer* para o polígono de número 92. A Figura 15 mostra a janela do QGIS com as duas tabelas de atributos abertas. A Tabela de atributos da esquerda trata-se da tabela de atributos do mapa IR, enquanto a do lado direito (superior) é a tabela de atributos da feição de *buffer* do polígono de “IDapp” número 92. No destaque, vemos que os valores coincidem, comprovando que o cálculo de área e a correspondência na tabela do mapa IR foi bem feita.

Olhando ainda a Figura 15 podemos conferir que o campo “areaPor” foi corretamente

Figura 15 - Validação da área do buffer.



Fonte: A Autora

calculado, uma vez que 29953,14 dividido 841370,31 (área do polígono) é, de fato, 0,0356 ou 3,56%. Verificamos ainda que este valor pertence ao intervalo de cor que foi atribuído ao polígono na representação final do mapa IR (Figura 13a)).

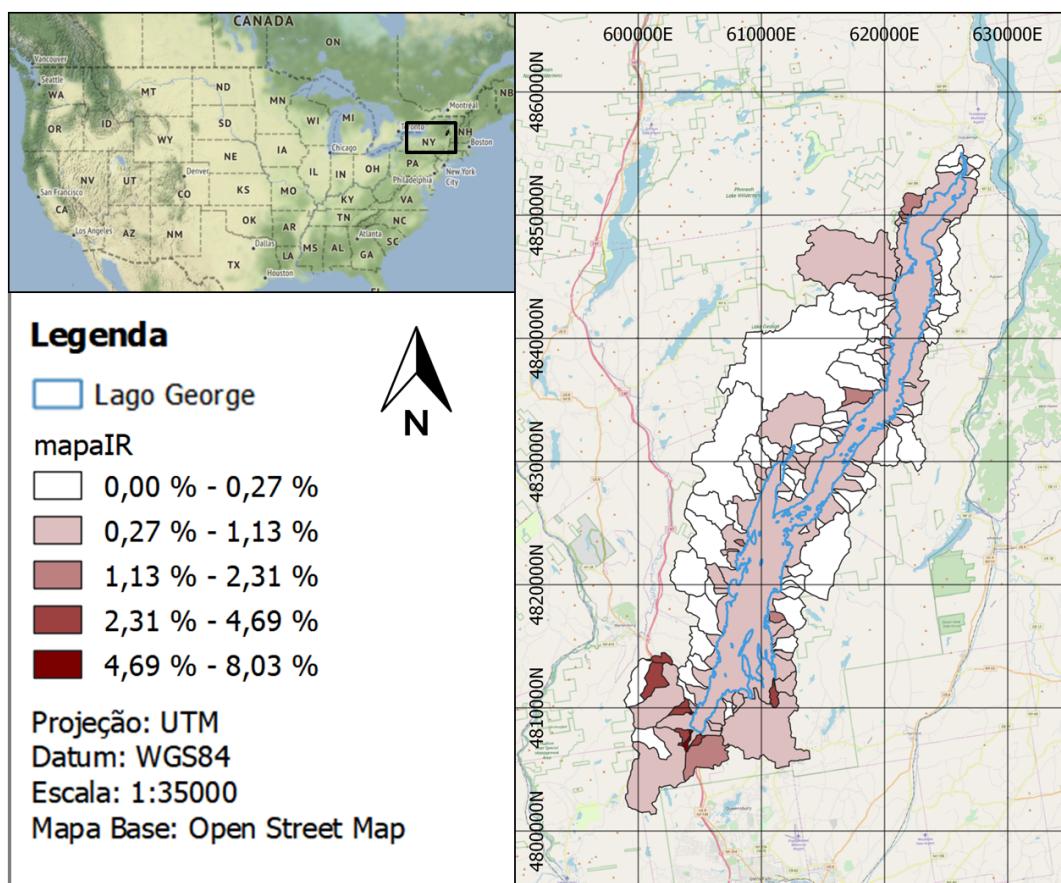
A última conferência trata-se da verificação do mapa IR com relação às suas coordenadas. A Figura 16 mostra o mapa IR em conformidade com a região do Lago George. Isso mostra que não houve distorção ou deslocamento do vetor durante a execução dos processos.

4.2 Mapa de Distribuição de Sal

De forma análoga ao que fizemos para o mapa IR, vamos agora apresentar os resultados para o mapa DS e buscar validar seus dados de saída. A Figura 17a) apresenta o mapa DS da maneira que é gerado pelo *plugin* e a Figura 17b) sua tabela de atributos. Para gerar esse mapa no *plugin* RSMB utilizamos os dados de teste apresentados na seção 2.4, o intervalo de data de 01/12/2017 a 31/03/2018 para contemplar todo o período de inverno, e unidade tonelada por milha (ton/mile) já que é nessa unidade que está o acúmulo de sal no dado sintético de sal setado como dado de entrada na interface (vide seção 2.4).

A tabela de atributos do mapa DS, apresentada na Figura 17b), é composta por 5 colunas. As duas primeiras colunas são oriundas do dado de entrada que são mantidos para possível necessidade do usuário. As colunas “tamEstr” e “NEstr” são oriundas da execução do estágio B do módulo de execução para criação do mapa DS. A coluna “qtd_sal” mostra a quantidade de

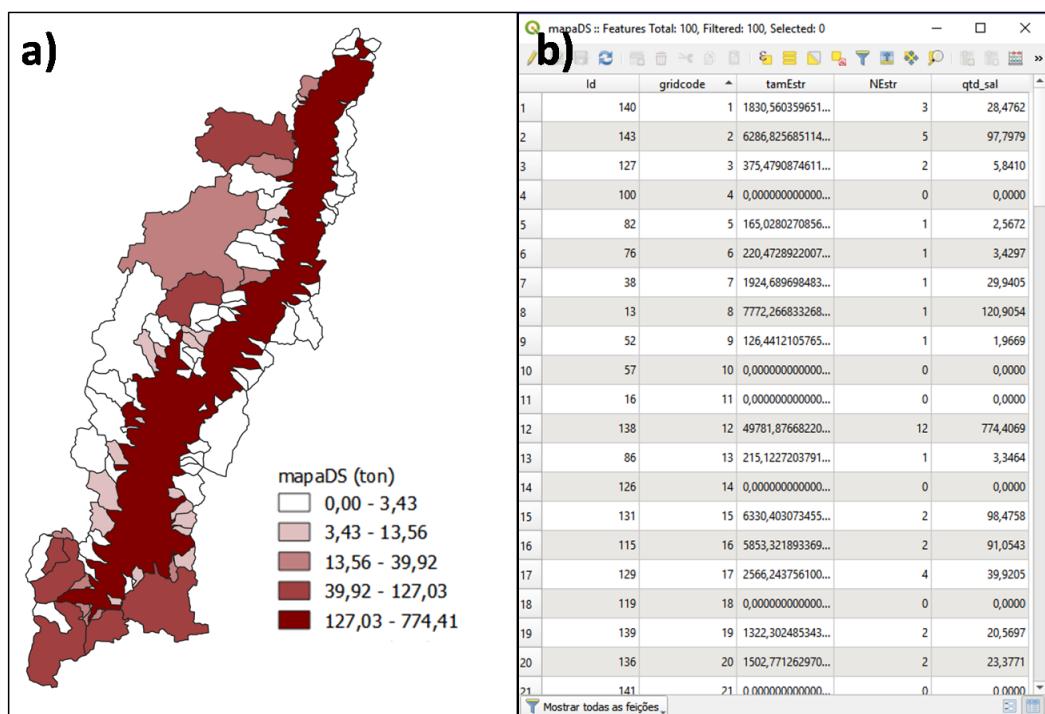
Figura 16 - Validação das coordenadas do mapa IR.



Legenda: Os valores na legenda correspondem aos intervalos de porcentagens de área do *buffer* das estradas por área de bacia.

Fonte: A Autora

Figura 17 - Mapa Distribuição de Sal.



Legenda: Apresentação do mapa DS: (a) mapa renderizado e (b) sua tabela de atributos.

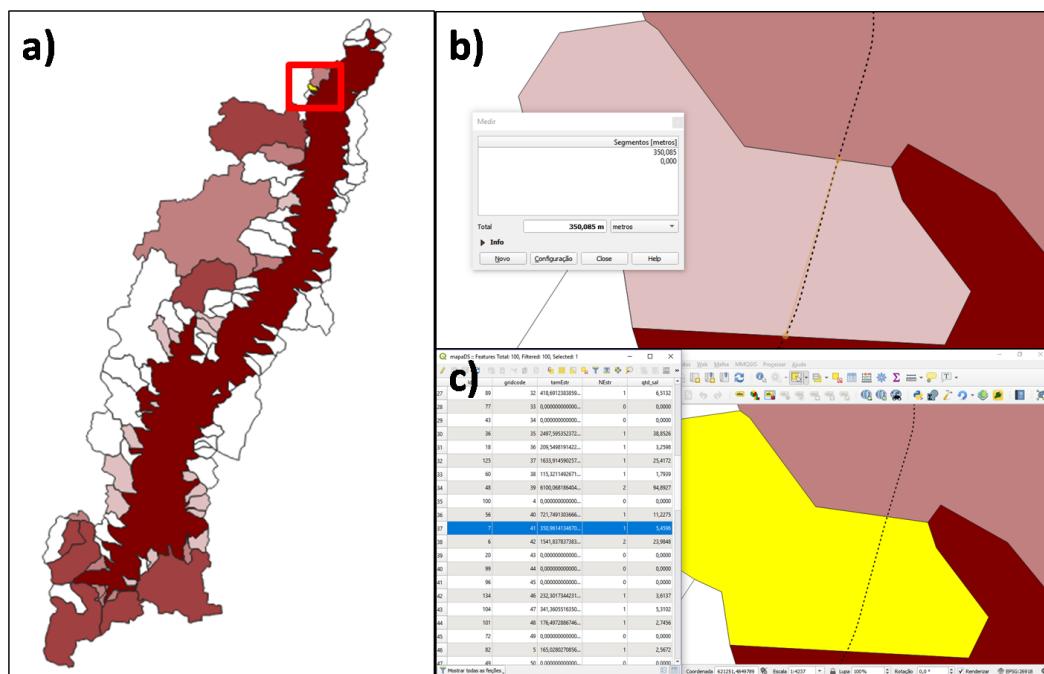
Fonte: A Autora

sal despejada no período determinado para cada bacia. Como a unidade escolhida na interface foi tonelada por milha, os valores correspondem a quantidade de sal em toneladas. Esta última coluna que é a utilizada para a visualização do mapa carregado na tela do QGIS.

Como os arquivos de bacias e estradas são os mesmos nos dois mapas, da mesma maneira que para o mapa IR, existem bacias que não possuem estradas e nestas os atributos “tamEstr”, “NEstr” e “qtd_sal” levam o valor 0. Isso significa que nenhum sal foi despejado diretamente nessa bacia no período observado, de acordo com os dados existentes.

A Figura 18a) mostra a localização do polígono (bacia) utilizada como exemplo para verificar o cálculo do tamanho de estrada feito na coluna “tamEstr”. A Figura 18b) mostra um zoom para o trecho da estrada no polígono onde foi utilizada a ferramenta de medição de distâncias do QGIS para medir aproximadamente o comprimento da linha da estrada. Podemos conferir o valor aproximado de 350 metros, que bate com aquele no campo da tabela de atributos da bacia correspondente, como pode ser visto na Figura 18c).

Figura 18 - Validação dos resultados para o mapa Distribuição de Sal.



Legenda: Imagens para apoiar a validação da correta execução do *plugin* na confecção do mapa DS: (a) região total, (b) recorte para feição de bacia e (c) tabela com destaque para feição

Fonte: A Autora

Para conferir se a quantidade de sal calculada na coluna “qtd_sal” está correta, vamos utilizar o mesmo polígono indicado na Figura 18a) e observar a tabela de dado sintético de sal. A Figura 19 mostra o dado sintético com os valores da quantidade de sal por milha de estrada

por dia. Em destaque (em amarelo) todos os valores que foram somados resultando em 25,0349 toneladas por milha. Todos os dias foram considerados na soma pois foi definido o intervalo total de dados na escolha da interface.

Figura 19 - Validação do cálculo da quantidade de sal despejada no período.

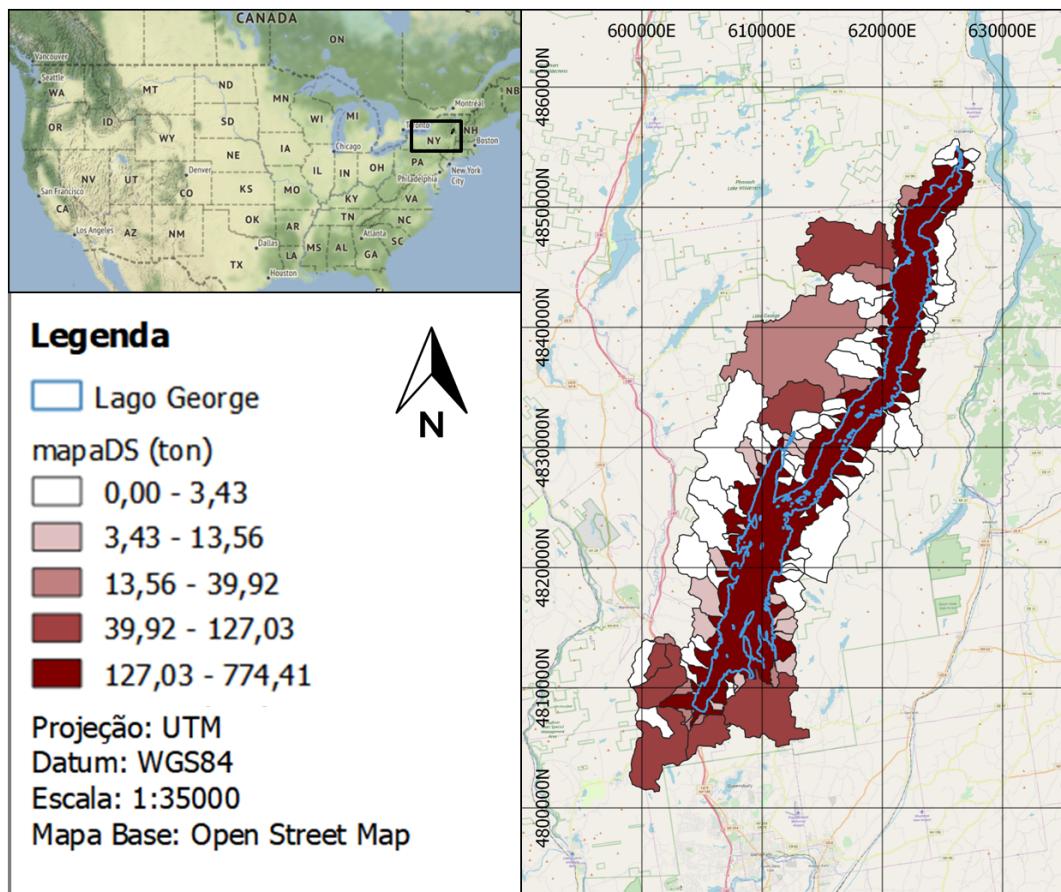
	K11		<i>f(x)</i>	=SOMA(C3:C33;E3:E33;G3:G30;I3:I33)							
	A	B	C	D	E	F	G	H	I	J	K
1											
2											
3	Data	SOMA	Data	SOMA	Data	SOMA	Data	SOMA			
4	01/12/2017	0	01/01/2018	0	01/02/2018	0,0633	01/03/2018	1,86733			
5	02/12/2017	0	02/01/2018	0	02/02/2018	0	02/03/2018	0			
6	03/12/2017	0	03/01/2018	0,53805	03/02/2018	2,34208	03/03/2018	0			
7	04/12/2017	1,74073	04/01/2018	0	04/02/2018	0	04/03/2018	0			
8	05/12/2017	0,1899	05/01/2018	0	05/02/2018	0	05/03/2018	0			
9	06/12/2017	0	06/01/2018	0	06/02/2018	2,21548	06/03/2018	0,75959			
10	07/12/2017	0	07/01/2018	0	07/02/2018	0	07/03/2018	0,94949			
11	08/12/2017	0	08/01/2018	0	08/02/2018	0,15825	08/03/2018	0,0633			
12	09/12/2017	0	09/01/2018	0	09/02/2018	0,0633	09/03/2018	0			
13	10/12/2017	0	10/01/2018	0	10/02/2018	0,0633	10/03/2018	0			
14	11/12/2017	1,26599	11/01/2018	2,50033	11/02/2018	0	11/03/2018	0			
15	12/12/2017	0	12/01/2018	0,85454	12/02/2018	0	12/03/2018	0,1899			
16	13/12/2017	0	13/01/2018	0	13/02/2018	0	13/03/2018	0,09495			
17	14/12/2017	0	14/01/2018	0	14/02/2018	0	14/03/2018	0			
18	15/12/2017	0	15/01/2018	0	15/02/2018	0,1266	15/03/2018	0			
19	16/12/2017	0	16/01/2018	0,03165	16/02/2018	0,03165	16/03/2018	0			
20	17/12/2017	0,0633	17/01/2018	0	17/02/2018	0,1266	17/03/2018	0			
21	18/12/2017	0,03165	18/01/2018	0	18/02/2018	0,22155	18/03/2018	0			
22	19/12/2017	0	19/01/2018	0	19/02/2018	0,1266	19/03/2018	0			
23	20/12/2017	0	20/01/2018	0	20/02/2018	0	20/03/2018	0			
24	21/12/2017	0,72794	21/01/2018	0,0633	21/02/2018	0,2532	21/03/2018	0			
25	22/12/2017	1,55084	22/01/2018	1,45589	22/02/2018	0,1899	22/03/2018	0			
26	23/12/2017	0	23/01/2018	0	23/02/2018	0	23/03/2018	0			
27	24/12/2017	1,29764	24/01/2018	0	24/02/2018	1,20269	24/03/2018	0			
28	25/12/2017	0	25/01/2018	0	25/02/2018	0	25/03/2018	0			
29	26/12/2017	0	26/01/2018	0,0633	26/02/2018	0	26/03/2018	0			
30	27/12/2017	0	27/01/2018	0	27/02/2018	0	27/03/2018	0,34815			
31	28/12/2017	0	28/01/2018	0	28/02/2018	0	28/03/2018	0,66464			
32	29/12/2017	0	29/01/2018	0			29/03/2018	0,53805			
33	30/12/2017	0	30/01/2018	0			30/03/2018	0			
34	31/12/2017	0	31/01/2018	0			31/03/2018	0			

Fonte: A Autora

Sabendo que 25,0349 toneladas de sal foram despejadas por milha de estrada no período definido, vamos calcular a quantidade despejada no polígono de exemplo (indicado na Figura 18a)). Transformando metros em milhas, temos que o trecho da estrada no polígono tem aproximadamente 0,21748 milha ($350/0,000621371$). Multiplicando esses valores temos o resultado final como sendo 5,45, exatamente igual àquele que consta na tabela de atributos do mapa DS, como pode ser visto na Figura 18c). Verificamos ainda que este valor pertence ao intervalo de cor que foi atribuído ao polígono na representação final do mapa DS (Figura 17a)).

Analogamente ao feito com o mapa IR, vamos verificar se o mapa DS manteve suas coordenadas após os processamentos a que foi submetido. A Figura 20 mostra o mapa DS em conformidade com a região do Lago George, mostrando que não houve distorção ou deslocamento do vetor durante a execução dos processos.

Figura 20 - Validação das coordenadas do mapa DS.



Legenda: Os valores na legenda correspondem aos intervalos de quantidade de sal despejado no inverno em toneladas.

Fonte: A Autora

4.3 Análise de Desempenho

Para a análise de desempenho do *plugin* RSMB será apresentado o tempo de processamento para mapa (Tabelas 4 e 5). Como visto na seção 3.2, para o mapa IR, existe um número muito maior de processos em comparação ao mapa DS e veremos que, portanto, este leva mais tempo para ser processado.

Tabela 4 - Tempos de execução dos processos do mapa IR

Estágio	Tempo (s)
criação de buffer	0.2187337
criação de campo ID	0.0468719
dividir feições	7.8744575
recorte	74.7992508
união de feições	59.7770984
criação de campo de área dos polígonos de estrada	0.0479876
criação de campo de área das bacias	0.0399727
união de tabelas de atributos	0.0239908
cálculo da porcentagem	0.0360157
preencher vazios	0.2210023

Tabela 5 - Tempos de execução dos processos do mapa DS

Estágio	Tempo (s)
cálculo do total de sal acumulado	0.0624573
somar total de linhas dentro dos polígonos	0.4374651
cálculo da quantidade de sal por bacia	0.1406137

O tempo total de execução (considerando o momento que o botão “Ok” é acionado até o aparecimento do mapa na tela de visualização do QGIS) é de cerca de 3 minutos e de 3 segundos para os mapas IR e DS, respectivamente. Esses tempos correspondem não só à execução das tarefas (apresentados nas tabelas) mas também ao carregamento dos dados, das definições, e da aplicação do estilo e carregamento do mapa na tela do QGIS.

O tempo de execução do mapa IR supera o excessivamente o mapa DS. Isso porque os estágios mais complexos 3, 4 e 5 (vide 3.2.2) vão realizar tantas iterações quantos forem o número de feições. Já o número de feições do arquivo de estradas não é tão relevante para o tempo de confecção do mapa IR, uma vez que as feições são dissolvidas logo na primeira manipulação desse vetor (na criação do buffer, estágio 1, vide 3.2.2).

Para o mapa DS, o número de registros do arquivo csv de dados de sal não interfere tanto no tempo de processamento já que o formato csv é um formato de arquivo simples e “leve”. Além disso, apenas o intervalo escolhido pelo usuário é submetido aos processos do *plugin*, independentemente do total de registros existentes no arquivo. Neste mapa o que provavelmente

afetará mais o tempo de execução é a densidade do arquivo de estradas, que vai influenciar diretamente o tempo de execução do estágio B de processamento.

Quando se realiza um teste de *software* é preciso considerar a configuração do ambiente de teste. Isto por que o *plugin* vai rodar suas funções de maneira mais rápida ou mais lenta dependendo da capacidade de processamento da máquina e também das características dos dados de entrada fornecidos. Por isso, apresentamos as configurações da máquina (configurações de hardware) utilizada nos testes apresentados nas seções 2.4 e 2.2 e as configurações dos dados utilizados como entrada na ferramenta.

Ainda é importante lembrar que executamos o *plugin* diretamente no QGIS versão 3.6.0. Para isso instalamos o plugin no QGIS ao copiar a pasta do *plugin* desenvolvida neste trabalho com o suporte do *Plugin Builder* na pasta responsável pelos *plugins* Python dentro da pasta de instalação do QGIS.

4.3.1 Teste de Hipótese de Desempenho

Ao longo do desenvolvimento das rotinas esbarramos em situações onde existiam mais de uma alternativa para chegar no mesmo resultado. Nesses casos, optamos por trabalhar da seguinte forma: testamos as duas hipóteses e medimos o tempo de processamento. Aquela hipótese que apresentou menor tempo de preprocessamento foi então utilizada na solução final e serve de base para justificar a tomada de decisão pelo uso de processos internos do QGIS sempre que disponíveis.

O tempo de processamento mais curto corresponde em maior desempenho para o sistema. Portanto, trabalhando dessa forma, buscamos otimizar o código para apresentar um *plugin* de maior qualidade. Um exemplo ilustrativo desse processo é apresentado nos próximos parágrafos.

Para criar o campo de acúmulo de sal resultante do período definido pelo usuário no mapa DS, deparamos com duas soluções. A primeira trata-se de executar a soma em linguagem Python pura e depois criar uma coluna na tabela de atributos da camada vetorial de bacia e populá-la com os dados calculados (solução 1). A segunda alternativa executa todo o processo na calculadora de campo do Qgis (solução 2).

Tendo o comprimento total de estradas dentro de cada polígono (guardado na coluna “tamEstr”) e a quantidade de sal despejada no tempo escolhido pelo usuário por unidade de estrada (guardado na variável “soma”) as soluções para a criação da variável com o total acumulado de sal em cada bacia (variável “qtd_sal”) seguem das seguintes formas:

- Solução 1: cria-se a coluna “qtd_sal” vazia na tabela de atributos que vai sendo preenchida com os valores calculados fora do QGIS. Ou seja, multiplica-se o tamanho da estrada de cada feição (oriunda da coluna “tamEstr”) pelo valor de quantidade de sal por unidade de estrada (“soma”) em uma variável e o campo da coluna “qtd_sal” recebe o valor dessa variável na feição correspondente. Isto é feito até preencher todas as feições da coluna

“qtd_sal”.

- Solução 2: utiliza-se a calculadora de campo do QGIS para criar uma nova coluna com o resultado da multiplicação dos tamanhos de estradas (oriunda da coluna “tamEstr”) pela variável “soma”. Ou seja, a coluna já é criada com os valores correspondentes.

Observamos a solução empregada utilizando as chamadas para a criação das colunas no QGIS (solução 2) apresentou maior desempenho, como pode ser comparado a partir da média de 8 medidas de tempo de processamento para cada solução apresentada na Tabela 6.

Tabela 6 - Tempos de processamento

	Solução 1	Solução 2
Tempo médio de Processamento (s)	0,8050	0,2444

A explicação disso pode ser devido ao fato que as rotinas do QGIS são programadas em C++ que, de uma forma geral, apresenta maior velocidade de processamento que Python. Dessa maneira, mesmo utilizando Python para chamar a função em C++ temos um resultado mais eficiente que fazer a utilização da linguagem Python pura para executar todo o procedimento. A partir de testes como esses definimos priorizar no desenvolvimento do *plugin* RSMB essa forma de programação, procurando sempre que possível trabalhar com funções internas do QGIS.

5 DISCUSSÃO

Neste Capítulo são discutidos alguns tópicos a cerca do trabalho desenvolvido. Essa discussão é importante para dar desfecho ao trabalho, fazendo algumas considerações que não se encaixam nos outros capítulos.

Como sinalizado na introdução, o *plugin* RSMB pode ser útil em aplicações diferentes daquela a qual teve como finalidade. Entendemos ainda que diferentes fenômenos podem implicar em modelagens distintas e necessidades de alterações no código fonte do produto aqui apresentado. De todo modo, a seção 5.1 visa apontar possibilidades de aplicações análogas, reforçando a prerrogativa de maior interação entre trabalhos de engenharia cartográfica com automatização computacional. A seção 5.2 traz as questões desafiadoras do desenvolvimento que por algum motivo não foram implementadas nesta versão do *plugin*. Essas questões, no entanto, não prejudicam o funcionamento da ferramenta. Por fim, a seção 5.3 aponta para as possibilidades de trabalhos futuros.

5.1 Aplicações Análogas para Utilização do *Plugin* RSMB

Muito embora todo o desenvolvimento e a avaliação experimental tenham sido pautadas na problemática da poluição causada pelo sal para degelo de estradas, nesta seção pretendemos mostrar outras aplicações para as quais os recursos desenvolvidos no *plugin* RSMP podem ser úteis.

O mapa IR apresenta porcentagens relativas de uma área de *buffer* de feições lineares por áreas de polígonos. Um exemplo onde esse mapa pode ser útil é no estudo de impacto causado pela construção de uma estrada ou linha de transmissão. Tendo como polígonos as áreas de floresta, ou áreas protegidas ou habitadas e como linhas os possíveis traçados da estrada ou linha de transmissão que pretende-se construir, a partir desse mapa é possível estabelecer qual traçado traria menos impacto ao meio ambiente, ou teria necessidade de causar menos desapropriação. A consideração de quanto seria desmatado ou quantas famílias deveriam ser realocadas influencia, inclusive, no projeto financeiro da construção.

Um exemplo sólido dessa hipótese trata-se da rodovia Rodoanel, que sofreu atraso em sua construção principalmente devido às questões ambientais. O maior problema girou em torno da questão dos resíduos da construção da rodovia, que poderiam causar assoreamento nos mananciais da região. Sendo assim, um estudo mais aprofundado que levasse em consideração um raio de dispersão possível desses resíduos para a criação do *buffer*, a partir da informação gerada no mapa DS seria possível estimar quais áreas do manancial estariam mais vulneráveis. Dessa forma seria possível determinar, inclusive, onde devem ser focadas as campanhas de mitigação, como obras de drenagem.

Outro exemplo para utilidade do mapa IR trata-se de aplicações de agricultura intensiva moderna para aplicação de fertilizante na plantação. As rotas das máquinas responsáveis pela dispersão do fertilizante poderiam ser estudadas de forma a maximizar a quantidade de áreas de plantação atingidas. Outro caso é, dado uma rota definida, investigar se toda a plantação está sendo alcançada (no caso ideal, todas as feições trariam na coluna “qtd_sal” o valor de 100%), e, caso não esteja, determinar quais são essas áreas para conduzir a aplicação manual, por exemplo.

Para esse mesmo caso da agricultura, o mapa DS também poderia ter serventia. No caso de aplicação de fertilizante, por exemplo, o mapa DS poderia indicar as área de maior concentração de fertilizante e nortear a escolha de onde ficaria cada cultivo, já que algumas plantas tem necessidade de mais fertilizantes que outras. A mesma lógica poderia ser utilizada para a aplicação de agrotóxico.

Mais uma possível aplicação para o mapa DS é voltada para ecoturismo. Neste caso, as feições lineares poderiam ser trilhas ecológicas e o poluente seria o lixo deixado ao longo delas pelos turistas. Nesse caso a questão temporal é bastante importante já que existem períodos do ano que as trilhas são muito mais utilizadas, como durante o verão e as férias escolares.

Essas são apenas algumas ideias onde a aplicação dos mapas confeccionados pelo *plugin* RSMB poderia ser útil. De um modo geral, sempre que envolve estudo de áreas de feições lineares dentro de polígonos ou concentração em polígonos de algo distribuído ao longo de feições lineares (podendo ser considerado um tempo de investigação), os mapas podem ser utilizados.

Ainda mais relevante que a possibilidade de utilização dessa solução para outras finalidades, a contribuição bastante relevante deste trabalho é a disponibilização do código de execução do *plugin*. Tanto a escrita do código quanto a metodologia adotada podem ser úteis como suporte para trabalhos de desenvolvimento futuros (disponível em <https://github.com/laisbaroni/works/monografia>). Sendo assim, inúmeras outras soluções podem ser produzidas a partir da incrementação, edição, ou utilização de fragmentos do código aqui desenvolvido.

5.2 Questões em Aberto

Observamos alguns pontos do código onde seria vantajoso alguma mudança, adaptação ou melhoria. Por motivos diversos não conseguimos implementar esses aperfeiçoamentos nessa primeira versão do *plugin* RSMB, mas deixamos sinalizados com a intenção de fazê-los em uma versão futura.

A mais importante questão com relação à essa versão do *plugin* RSMP é que a função responsável por apagar as pastas de arquivos “bacias_individuais” e “buffer_individuais”, criadas durante a confecção do mapa IR, não executa corretamente no sistema operacional Windows. Algumas alternativas foram tentadas para resolver essa questão, mas todas esbarraram no mesmo problema de permissão. Os arquivos das pastas que deseja-se apagar por algum motivo constam

como estando em uso e, por isso, não podem ser deletados.

É conhecido o fato de que não são administradas reprojeções internas no plugin, o que pode levar a erros de computação caso o usuário entre com dados cujas projeções não sejam métricas. Uma vez que nos dados de testes não abordaram outras projeções é possível admitir que falhas ocorram nestas condições.

Outra questão de bastante importância era disponibilizar ao usuário uma ferramenta de ajuda para orientar a utilização do *plugin* RSMB. A implementação de um “help” não só traria a descrição do que significa cada campo e quais os requisitos de entrada mas também traria uma descrição resumida dos processos utilizados para a confecção dos mapas. Isso esclareceria para o usuário como a ferramenta funciona, eliminando a ideia de processamento em “caixa preta”, ou seja, um sistema fechado onde é conhecida as entradas, as saídas, mas não a manipulação feita nos dados.

Por último, ficou pendente a conferência dos campos de data do mapa DS. Como mostrado na seção 3.3.3, procuramos proteger os campos da interface para evitar erros no preenchimento dos dados necessários para a confecção do mapa. Porém, entre as restrições verificadas, não foi incluída a conferência se, nas datas escolhidas para o período de investigação a ser considerado, a data de início é antes da data de fim. Isso é fundamental para a definição do intervalo.

5.3 Proposta de Trabalhos Futuros

Nessa seção são apresentadas possibilidades vislumbradas de aprimoramento do *plugin*, tanto em melhoramento das funções que já estão sendo executadas, ou seja, a confecção dos mapas IR e DS, quanto na expectativa incrementar uma nova funcionalidade ao *plugin*, ou seja, a criação de um terceiro produto, um mapa animado.

5.3.1 Utilidades a Serem Implementadas

A primeira questão se trata das unidades de saída representadas no mapa DS. Nas opções da interface o usuário pode apenas escolher a unidade que está sendo dada de entrada (referente a unidade representada na arquivo de sal), mas não pode escolher a unidade de saída. Seria interessante que se o usuário desse como entrada a unidade de tonelada por milha e pudesse ter no mapa final a quantidade de sal em cada bacia em quilogramas, por exemplo.

Um passo ainda além com relação às unidades do mapa DS seria admitir que, dado uma entrada em unidade de solução salina (exemplo: galão por milha) e uma concentração dessa solução a saída pudesse ser em unidade sólida de quantidade de sal por bacia (exemplo: quilograma). Isso envolveria uma complexidade a mais na conversão, mas seria de grande utilidade.

Quando é apertado o botão “ok” para a confecção dos mapas, o processamento se inicia. Enquanto o código está sendo executado, na janela da interface fica a mensagem de “o programa não está respondendo”. Isso significa que o QGIS está ocupado e não pode ser editado, mas o fato de ele não estar respondendo, no entanto, é uma afirmativa equivocada. O QGIS está sim respondendo, executando as funções solicitadas. Consideramos, então, que seria interessante dar um aviso mais claro ao usuário de que o mapa está em confecção. Disponibilizar uma barra de progresso que mostrasse o quanto foi executado do código em porcentagem seria o ideal para isso.

Outra implementação que poderia ser interessante é a de fornecer ao usuário a possibilidade de que ele pudesse ter acesso aos arquivos intermediários do processamento. Há a possibilidade de que seja interessante ao usuário ter, por exemplo, o shapefile do *buffer* das estradas dentro dos polígonos (caso do mapa IR). Como esses arquivos já estão sendo gerados, não seria custoso oferecer a opção de que ele possa ser salvo em disco. Isso poderia servir também como meio de depuração dos processos executados.

Por último, uma vez que sabemos da possibilidade da utilização deste *plugin* em outras aplicações, seria interessante generalizar sua apresentação na interface. Isso incluiria mudar a descrição do *plugin* e generalizar os nomes dos campo. Por exemplo, onde está “*roads file*” ficaria “*line vector layer*” e analogamente para os outros campos.

5.3.2 Criação de Mapa Animado

Além das melhorias sinalizadas na seção anterior, vislumbra-se a possibilidade de criação de um terceiro mapa a partir do *plugin* RSMB. Este mapa, chamado preliminarmente de **mapa animado de distribuição de sal** (mapa ADS), seria um aprimoramento do mapa DS, pois traria o mesmo conteúdo mas de forma animada.

A ideia é que o usuário pudesse escolher uma janela de observação e a visualização do mapa fosse sendo atualizada nessas janelas ao longo de todo o período contemplado pelo dado. Por exemplo, ao escolher o intervalo de um dia o usuário observaria a quantidade de sal despejada em cada polígono no primeiro dia, depois no segundo, terceiro e assim sucessivamente até o último dia contido no dado de sal. A sobreposição desses mapas para os resultados de cada dia resultariam em uma visualização animada, como um vídeo.

Entre as dificuldades para aderir a criação nesse mapa no *plugin* estão a necessidade de utilização de um *plugin* externo e a complexidade no formato de dado necessário. Existe um *plugin* QGIS voltado para a animação de mapas, chamado “Time Manager Plugin”, o qual exige que, para cada mapa da animação exista um conjunto de feições correspondentes ((QGIS PYTHON PLUGINS REPOSITORY, 2019)). Ou seja, as feições poligonais estariam repetidas tantas vezes quanto fosse necessário para atender às janelas de observação.

O grande diferencial do mapa ADS com relação ao mapa DS seria a maior agilidade na

comparação do fenômeno no tempo. Enquanto no mapa DS seria necessário criar um mapa de cada vez no *plugin*, no mapa ADS todos os mapas seriam criados de uma vez e já sobrepostos para a comparação. Em contrapartida, caso a intenção seja a visualização do fenômeno em apenas um intervalo de tempo, por exemplo, ao longo de todo o inverno, o mapa ADS não se faz necessário.

6 CONCLUSÕES

Visando os objetivos propostos neste trabalho, a partir dos métodos adotados, realizou-se a criação do *plugin* Road Salt Map Builder (RSMB). O *plugin* desenvolvido confecciona de forma automática mapas orientados para o estudo de impacto ambiental causado por sal de degelo em estradas. Esses mapas tem o propósito de apresentar informações sobre indicadores de risco potencial ao poluente e sobre a expressividade em termos de acúmulo de poluente em uma bacia ou parte dela.

Foram apresentadas todas as etapas de desenvolvimento do *plugin*, os desafios na escrita do código, as rotinas implementadas e ainda propostas para aprimoramento em trabalhos futuros. Os arquivos onde estão escritos os códigos foram disponibilizados em anexo a este trabalho de forma que fique acessível para consulta ou possível reutilização.

Uma prova de conceito foi realizada utilizando dados da bacia do Lago George, Nova Iorque, para verificar e validar o correto funcionamento do *plugin*. Na apresentação dos resultados os resultados intermediários são mostrados para observar o desempenho de cada função individualmente e a verificação com os resultados esperados foram realizadas. Nenhum problema de execução foi encontrado nos testes.

Mostra-se a cartografia como ciência de ampla utilidade, já que seus recursos servem, inclusive, para apoiar estudos de aplicações ambientais como a problemática do impacto causado por sal de degelo de estradas. Desde a definição da proposta, passando pela implementação das rotinas e a utilização de Sistema de Informação Geográfica, até e a interpretação dos mapas gerados, a cartografia se mostra presente e seus conceitos fundamentais para a realização deste trabalho.

Realizações futuras incluem dar continuidade ao trabalho para implementação das melhorias apontadas e para o desenvolvimento do terceiro produto produzido pelo *plugin*, o mapa animado de distribuição de sal, proposto na seção 5.3.2. Entende-se também que estudos futuros podem propor novas configurações ou parametrizações para prover suporte a demais aplicações com características semelhantes.

Estima-se que a publicação do *plugin* RSMB num repositório oficial possa ocorrer em um futuro breve. Isto é importante para que o RSMB possa estar acessível a toda a comunidade de usuários QGIS. Ressalta-se que do ponto de vista atual o *plugin* deve ser classificado como experimental e que o *feedback* da comunidade será importante para sua maturação.

REFERÊNCIAS

- ARCHELA, R. S. *et al.* Abordagem metodológica para cartografia ambiental. *GEOGRAFIA (Londrina)*, v. 11, n. 1, p. 55–62, 2002.
- ARCHELA, R. S.; THÉRY, H. Orientação metodológica para construção e leitura de mapas temáticos. *Confins [Online]*, v. 3, 2008.
- CASTI, E. *Reflexive cartography: A new perspective in mapping*. [S.l.]: Elsevier, 2015. v. 6.
- CLEAR ROADS. *Intro to Clear Roads*. 2019. Disponível em: <<https://clearroads.org/intro-to-clear-roads/>>. Acesso em: 20 jun. 2019.
- DUGAN, H. A. *et al.* Long-term chloride concentrations in north american and european freshwater lakes. *Scientific data*, Nature Publishing Group, v. 4, p. 170101, 2017.
- FREE SOFTWARE FOUNDATION. *Licença Pública Geral GNU*. 2019. Disponível em: <<https://www.gnu.org/licenses/licenses.html>>. Acesso em: 19 jun. 2019.
- GUELKE, L. Cartographic communication and geographic understanding. *Cartographica: The International Journal for Geographic Information and Geovisualization*, University of Toronto Press, v. 14, n. 1, p. 129–145, 1977.
- ICY ROAD SAFETY. *Icy Road Accident Statistics*. 2019. Disponível em: <<http://icyroadsafety.com/fatalitystats.shtml>>. Acesso em: 19 jun. 2019.
- KELLY V.R., F. S. W. K. Road salt the problem, the solution, and how to get there. Cary Institute of Ecosystem Studies, 2019.
- LOBBEN, A. Classification and application of cartographic animation. *The Professional Geographer*, Taylor & Francis, v. 55, n. 3, p. 318–328, 2003.
- MARTINELLI, M. Cartografia ambiental: uma cartografia diferente? *Revista do Departamento de Geografia*, v. 7, p. 61–80, 1994.
- MARTINELLI, M. *Mapas da geografia e cartografia temática*. [S.l.]: Editora Contexto, 2003.
- MARTINELLI, M. Cartografia dinâmica: tempo e espaço nos mapas. *GEOUSP: Espaço e Tempo [Online]*, n. 18, p. 53–66, 2005.
- MENEZES, P. M. L. de; FERNANDES, M. do C. *Roteiro de cartografia*. [S.l.]: Oficina de Textos, 2016.
- NATIONAL WEATHER SERVICE. *Alerts*. 2019. Disponível em: <<https://www.weather.gov/>>. Acesso em: 20 jun. 2019.
- NOVOTNY, E. V.; MURPHY, D.; STEFAN, H. G. Increase of urban lake salinity by road deicing salt. *Science of the Total Environment*, Elsevier, v. 406, n. 1-2, p. 131–144, 2008.
- NYC'S WEBSITE. *Political and Administrative Districts Download and Metadata*. 2019. Disponível em: <<https://www1.nyc.gov/site/planning/data-maps/open-data/districts-download-metadata.page>>. Acesso em: 20 jun. 2019.

NYS ADIRONDACK PARK AGENCY. *Wetland Covertypes of the Lake Champlain and Lake George Watersheds*. 2019. Disponível em: <<https://gis.ny.gov/gisdata/inventories/details.cfm?DSID=1222>>. Acesso em: 20 jun. 2019.

OSM FOUNDATION. *OpenStreetMap*. 2019. Disponível em: <<https://www.openstreetmap.org/>>. Acesso em: 20 jun. 2019.

PETERSON, M. P. *Interactive and animated cartography*. [S.l.]: Prentice Hall, 1995.

QGIS PROJECT. *A Free and Open Source Geographic Information System*. 2019. Disponível em: <<https://www.qgis.org>>. Acesso em: 19 jun. 2019.

QGIS PYTHON API. *QGIS Python API documentation project*. 2019. Disponível em: <<https://qgis.org/pyqgis/master/>>. Acesso em: 11 mai. 2019.

QGIS PYTHON PLUGINS REPOSITORY. *TimeManager plugin*. 2019. Disponível em: <<https://plugins.qgis.org/plugins/timemanager/>>. Acesso em: 07 jun. 2019.

ROYCE, W. W. Managing the development of large software systems: concepts and techniques. In: IEEE COMPUTER SOCIETY PRESS. *Proceedings of the 9th international conference on Software Engineering*. [S.l.], 1987. p. 328–338.

SALGADO, G. *et al.* Integração do sensoriamento remoto e sistema de informações geográficas para análise temporal do uso da terra: Parque municipal da lagoa do peri, florianópolis-sc. Florianópolis, SC, 2002.

SAMPAIO, T. V. M. Cartografia temática. Programa de Pós-Graduação em Geografia - UFPR, 2019.

SLOCUM, T. A. *et al.* Thematic cartography and geographic visualization. Prentice hall, 2008.

TOBLER, W. R. Choropleth maps without class intervals. *Geographical analysis*, Blackwell Publishing Ltd Oxford, UK, v. 5, n. 3, p. 262–265, 1973.

APÊNDICE A – Arquivos do python

A.1 Arquivos principais

A.1.1 Camada de modelo (roadsalt.py)

```
# -*- coding: utf-8 -*-
#
# roadsalt.py
#
# Authors:      Lais Baroni <laisrbaroni@gmail.com> (1)
#               Alvaro Bueno <alvarobb10@gmail.com> (2)
#               Irving Badolato <irvingbadolato@eng.uerj.br> (1)
#
# Copyright 2019 (1) CARTO/UERJ
#                   (2) IBM
#
# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston,
# MA 02110-1301, USA.

# Pacotes e modulos em uso
from qgis.core import *
from qgis.gui import *
from PyQt5.QtGui import *
from PyQt5.QtCore import *
from qgis.analysis import *
from processing.algs.qgis.QgisAlgorithm import *
import processing
import sys
import os
import qgis.utils
```

```

import glob
import shutil
import time

# Secao dedicada as funcoes do plugin em desenvolvimento

def loadData(streets_filename, basins_filename, salt_filename = ''):
    return {"street" : QgsVectorLayer(streets_filename, ''),
            "basin" : QgsVectorLayer(basins_filename, ''),
            "salt" : QgsVectorLayer(salt_filename, '')}

def showData(filename, field = None, iface = None):
    if iface:
        layer = iface.addVectorLayer(filename, "", "ogr")
        renderer = QgsGraduatedSymbolRenderer()
        renderer.setClassAttribute(field)
        layer.setRenderer(renderer)
        layer.renderer().updateClasses(layer,
                                        QgsGraduatedSymbolRenderer.Jenks, 5)
        layer.renderer().updateColorRamp(QgsGradientColorRamp(Qt.
                                                               white, Qt.darkRed))
        iface.layerTreeView().refreshLayerSymbology(layer.id())
        iface.mapCanvas().refreshAllLayers()

def clear(dir_trabalho):
    path1 = os.path.join(dir_trabalho, "buffer_individuais")
    path2 = os.path.join(dir_trabalho, "bacias_individuais")
    filesToRemove1 = [os.path.join(path1,f1) for f1 in os.listdir(path1)
                      ]
    for f1 in filesToRemove1:
        os.remove(f1)
    filesToRemove2 = [os.path.join(path2,f2) for f2 in os.listdir(path2)
                      ]
    for f2 in filesToRemove2:
        os.remove(f2)
    os.rmdir(os.path.join(dir_trabalho, "bacias_individuais"))
    os.rmdir(os.path.join(dir_trabalho, "buffer_individuais"))

def stage1(vector, tam_buffer, unit="m"):
    if unit=="m":
        mybuffer = tam_buffer
    if unit=="km":
        mybuffer = tam_buffer * 1000
    if unit=="mile":
        mybuffer = tam_buffer * 1609.34
    args = {'INPUT': vector,
            'DISTANCE': mybuffer,

```

```

        'SEGMENTS': 10,
        'DISSOLVE': True,
        'END_CAP_STYLE': 0,
        'JOIN_STYLE': 0,
        'MITER_LIMIT': 10,
        'OUTPUT': 'memory:buffer'}
    return processing.run("native:buffer",args) ['OUTPUT']

def stage2(vector):
    args = {'INPUT': vector,
            'FIELD_NAME': 'IDapp',
            'FIELD_TYPE': 1,
            'FIELD_LENGTH': 10,
            'FIELD_PRECISION': 0,
            'NEW_FIELD': True,
            'FORMULA': '$id',
            'OUTPUT': 'memory:bacias'}
    return processing.run("qgis:fieldcalculator",args) ['OUTPUT']

def stage3(entrada, dir_trabalho):
    args = {'INPUT': entrada,
            'FIELD': 'IDapp',
            'OUTPUT': os.path.join(dir_trabalho, 'bacias_individuais')}
    return processing.run("qgis:splitvectorlayer",args) ['OUTPUT_LAYERS']

def stage4(vectorList, vector, dir_trabalho):
    os.mkdir(os.path.join(dir_trabalho, 'buffer_individuais'))
    result = []
    basepath = os.path.join(dir_trabalho, "buffer_individuais/buffer_")
    for item in vectorList:
        filename = basepath + os.path.basename(item)
        args = {'INPUT': item,
                'OVERLAY': vector,
                'OUTPUT': filename}
        result.append(processing.run("native:clip",args) ['OUTPUT'])
    return result

def stage5(entradas):
    args = {'LAYERS': entradas,
            'OUTPUT': 'memory:unidos'}
    return processing.run("native:mergevectorlayers",args) ['OUTPUT']

def stage6(entrada):
    args = {'INPUT': entrada,

```

```

        'FIELD_NAME': 'areaBuf',
        'FIELD_TYPE': 0,
        'FIELD_LENGTH': 10,
        'FIELD_PRECISION': 2,
        'NEW_FIELD': True,
        'FORMULA': '$area',
        'OUTPUT': 'memory:buffer_area'}
    return processing.run("qgis:fieldcalculator",args) ['OUTPUT']

def stage7(entrada):
    args = {'INPUT': entrada,
            'FIELD_NAME': 'areaBac',
            'FIELD_TYPE': 0,
            'FIELD_LENGTH': 10,
            'FIELD_PRECISION': 2,
            'NEW_FIELD': True,
            'FORMULA': '$area',
            'OUTPUT': 'memory:bacia_area'}
    return processing.run("qgis:fieldcalculator",args) ['OUTPUT']

def stage8(entrada1, entrada2):
    args = {'INPUT': entrada1,
            'FIELD': 'IDapp',
            'INPUT_2': entrada2,
            'FIELD_2': 'IDapp',
            'FIELDS_TO_COPY': 'areaBuf',
            'METHOD': 0,
            'DISCARD_NONMATCHING': False,
            'PREFIX': '',
            'OUTPUT': 'memory:bacia_areas'}
    return processing.run("native:joinattributestable",args) ['OUTPUT']

def stage9(entrada):
    args = {'INPUT': entrada,
            'FIELD_NAME': 'areaPor',
            'FIELD_TYPE': 0,
            'FIELD_LENGTH': 10,
            'FIELD_PRECISION': 4,
            'NEW_FIELD': True,
            'FORMULA': '"areaBuf"/"areaBac" * 100',
            'OUTPUT': 'memory:result'}
    return processing.run("qgis:fieldcalculator",args) ['OUTPUT']

def stage10(entrada, arq_destino):
    args = {'INPUT': entrada,
            'FIELD_NAME': 'areaPor',
            'FIELD_TYPE': 0,

```

```

        'FIELD_LENGTH': 10,
        'FIELD_PRECISION': 4,
        'NEW_FIELD': False,
        'FORMULA': 'if("areaPor" is null, 0, "areaPor")',
        'OUTPUT': arq_destino}

    return processing.run("qgis:fieldcalculator", args) ['OUTPUT']

def method_1(dir_trabalho, arq_entrada, arq_saida, opcoes):
    # Carrega os vetores [0]
    entrada = loadData(arq_entrada["street"], arq_entrada["basin"])
    # Cria o buffer de estradas [1]
    parte1 = stage1(entrada["street"], opcoes["buffer"], opcoes["unit"])
    # Cria o campo id nas bacias [2]
    parte2 = stage2(entrada["basin"])
    # Dividir as subbacias em arquivos diferentes [3]
    parte3 = stage3(parte2, dir_trabalho)
    # Cortar as bacias de acordo com a camada de buffer [4]
    parte4 = stage4(parte3, parte1, dir_trabalho)
    # Unir todos os recortes das estradas com buffer [5]
    parte5 = stage5(parte4)
    # Criar o campo com a area no arquivo de buffer [6]
    parte6 = stage6(parte5)
    # Criar o campo com a area no arquivo de bacias [7]
    parte7 = stage7(parte2)
    # Unir tabelas por atributo - unir o campo com a area da estrada no
    # vetor de bacia [8]
    parte8 = stage8(parte7, parte6)
    # Criar campo com a porcentagem de area do buffer da estrada pelo
    # poligono [9]
    parte9 = stage9(parte8)
    # Preencher campos vazios com 0 [10]
    stage10(parte9, arq_saida)
    # carrega resultado
    if "iface" in opcoes:
        showData(arq_saida, "areaPor", iface=opcoes["iface"])
    # Liberando a memoria
    entrada, parte1, parte2, parte3, parte4, parte5, parte6, parte7,
    parte8, parte9 = None, None, None, None, None, None, None, None, None,
    None, None
    # remove as pastas intermediarias criadas
    clear(dir_trabalho)

def stageA(entrada, valueField, dateField, start, end):
    features = entrada.getFeatures()
    acum = 0
    for feat in features:

```

```

        if feat[dateField] >= start and feat[dateField] <= end:
            acum += feat[valueField]

    return acum

def stageB(lines, polygons):
    args = {'LINES': lines,
            'POLYGONS': polygons,
            'LEN_FIELD': 'tamEstr',
            'COUNT_FIELD': 'NEstr',
            'OUTPUT': "memory:result"}
    return processing.run("qgis:sumlinelengths", args) ['OUTPUT']

def stageC(vectorLayer, const, arq_destino, length = "tonmile"):
    if length=="galmile" or length=="tonmile":
        mylength = const * 0.000621371
    if length=="literkm" or length=="kgkm":
        mylength = const * 0.001
    args = {'INPUT': vectorLayer,
            'FIELD_NAME': 'qtd_sal',
            'FIELD_TYPE': 0,
            'FIELD_LENGTH': 10,
            'FIELD_PRECISION': 4,
            'NEW_FIELD': True,
            'FORMULA': '%f*tamEstr' % (mylength),
            'OUTPUT': arq_destino}
    final = processing.run("qgis:fieldcalculator", args) ['OUTPUT']

def method_2(dir_trabalho, arq_entrada, arq_saida, opt):
    # Carrega os vetores e dado de sal [0]
    data = loadData(arq_entrada["street"], arq_entrada["basin"],
                    arq_entrada["salt"])
    # Computa o acumulado de sal no intervalo [1]
    soma = stageA(data["salt"], opt["valueIdx"], opt["dateIdx"], opt["t0"],
                   opt["t1"])
    # Conta o tamanho das estradas (em metros) dentro dos polígonos [2]
    tamEstr = stageB(data["street"], data["basin"])
    # multiplicar a soma de sal com a quantidade de estradas [3]
    stageC(tamEstr, soma, arq_saida, opt["length"])
    # carrega resultado
    if "iface" in opt:
        showData(arq_saida, "qtd_sal", opt["iface"])

```

A.1.2 Camada de visualização (roadsalt_mapbuilder.py)

```

# -*- coding: utf-8 -*-
"""
/*****



RoadSaltMap
    A QGIS plugin
cria mapas que dão apoio ao estudo de impacto causado pela despejo de sal
nas estradas
Generated by Plugin Builder: http://g-sherman.github.io/Qgis-Plugin-
Builder/
-----
begin          : 2019-05-07
git sha        : $Format:%H$
copyright      : (C) 2019 by Lais Baroni UERJ
email          : laisrbaroni@gmail.com
*****



/*****



*
*
*
*   This program is free software; you can redistribute it and/or modify
*
*   it under the terms of the GNU General Public License as published by
*
*   the Free Software Foundation; either version 2 of the License, or
*
*   (at your option) any later version.
*
*
*
*****



"""

from PyQt5.QtCore import QSettings, QTranslator, qVersion, QCoreApplication
, QDate, Qt
from PyQt5.QtGui import QIcon
from PyQt5.QtWidgets import QAction, QFileDialog, QMessageBox
from qgis.core import *

# Initialize Qt resources from file resources.py
from .resources import *
# Import the code for the dialog
from .roadsalt_mapbuilder_dialog import RoadSaltMapDialog
from .roadsalt import method_1, method_2
from datetime import date, datetime

```

```

import os.path, csv


class RoadSaltMap:
    """QGIS Plugin Implementation."""

    def __init__(self, iface):
        """Constructor.

        :param iface: An interface instance that will be passed to this
                      class
                      which provides the hook by which you can manipulate the QGIS
                      application at run time.
        :type iface: QgsInterface
        """

        # Save reference to the QGIS interface
        self.iface = iface
        # initialize plugin directory
        self.plugin_dir = os.path.dirname(__file__)
        # initialize locale
        locale = QSettings().value('locale/userLocale')[0:2]
        locale_path = os.path.join(self.plugin_dir, 'i18n', 'RoadSaltMap_'
                                  '{}.qm'.format(locale))

        if os.path.exists(locale_path):
            self.translator = QTranslator()
            self.translator.load(locale_path)

        if qVersion() > '4.3.3':
            QCoreApplication.installTranslator(self.translator)

        # Declare instance attributes
        self.actions = []
        self.menu = self.tr(u'&Road_Salt_Map_Builder')

        # Check if plugin was started the first time in current QGIS
        # session
        # Must be set in initGui() to survive plugin reloads
        self.first_start = None

    # noinspection PyMethodMayBeStatic
    def tr(self, message):
        """Get the translation for a string using Qt translation API.

        We implement this ourselves since we do not inherit QObject.

        :param message: String for translation.
        """

```

```

:type message: str, QString

:returns: Translated version of message.
:rtype: QString
"""

# noinspection PyTypeChecker,PyArgumentList,PyCallByClass
return QCoreApplication.translate('RoadSaltMap', message)

def add_action(
    self,
    icon_path,
    text,
    callback,
    enabled_flag=True,
    add_to_menu=True,
    add_to_toolbar=True,
    status_tip=None,
    whats_this=None,
    parent=None):
    """Add a toolbar icon to the toolbar.

:param icon_path: Path to the icon for this action. Can be a
    resource
    path (e.g. ':/plugins/foo/bar.png') or a normal file system
    path.
:type icon_path: str

:param text: Text that should be shown in menu items for this
    action.
:type text: str

:param callback: Function to be called when the action is triggered
    .
:type callback: function

:param enabled_flag: A flag indicating if the action should be
    enabled
    by default. Defaults to True.
:type enabled_flag: bool

:param add_to_menu: Flag indicating whether the action should also
    be added to the menu. Defaults to True.
:type add_to_menu: bool

:param add_to_toolbar: Flag indicating whether the action should
    also

```

```

    be added to the toolbar. Defaults to True.
:type add_to_toolbar: bool

:param status_tip: Optional text to show in a popup when mouse
    pointer
    hovers over the action.
:type status_tip: str

:param parent: Parent widget for the new action. Defaults None.
:type parent: QWidget

:param whats_this: Optional text to show in the status bar when the
    mouse pointer hovers over the action.

:returns: The action that was created. Note that the action is also
    added to self.actions list.
:rtype: QAction
"""

icon = QIcon(icon_path)
action = QAction(icon, text, parent)
action.triggered.connect(callback)
action.setEnabled(enabled_flag)

if status_tip is not None:
    action.setStatusTip(status_tip)

if whats_this is not None:
    action.setWhatsThis(whats_this)

if add_to_toolbar:
    # Adds plugin icon to Plugins toolbar
    self iface.addToolBarIcon(action)

if add_to_menu:
    self iface.addPluginToVectorMenu(
        self.menu,
        action)

self.actions.append(action)

return action

def initGui(self):
    """Create the menu entries and toolbar icons inside the QGIS GUI.
    """

```

```

icon_path = ':/plugins/roadsalt_mapbuilder/icon.png'
self.addAction(
    icon_path,
    text=self.tr(u'&Road_Salt_Map_Builder'),
    callback=self.run,
    parent=self.iface mainWindow()))

# will be set False in run()
self.first_start = True

def unload(self):
    """Removes the plugin menu item and icon from QGIS GUI."""
    for action in self.actions:
        self.iface.removePluginVectorMenu(
            self.tr(u'&Road_Salt_Map_Builder'),
            action)
        self.iface.removeToolBarIcon(action)

#map 1
def select_outputmap1_file(self):
    filename, _filter = QFileDialog.getSaveFileName(self.dlg,
                                                    "Select_output_file_name","","*.shp")
    self.dlg.l_outMap1.setText(filename)

def select_basin1_file(self):
    filename, _filter = QFileDialog.getOpenFileName(self.dlg,
                                                    "Select_basin_file","","*.shp")
    self.dlg.l_inBasin_1.setText(filename)
    self.dlg.l_inBasin_2.setText(filename)

def select_road1_file(self):
    filename, _filter = QFileDialog.getOpenFileName(self.dlg,
                                                    "Select_road_file","","*.shp")
    self.dlg.l_inRoad_1.setText(filename)
    self.dlg.l_inRoad_2.setText(filename)

#map 2
def select_outputmap2_file(self):
    filename, _filter = QFileDialog.getSaveFileName(self.dlg,
                                                    "Select_output_file","","*.shp")
    self.dlg.l_outMap2.setText(filename)

def select_basin2_file(self):
    filename, _filter = QFileDialog.getOpenFileName(self.dlg,
                                                    "Select_basin_file","","*.shp")
    self.dlg.l_inBasin_1.setText(filename)
    self.dlg.l_inBasin_2.setText(filename)

```

```

def select_road2_file(self):
    filename, _filter = QFileDialog.getOpenFileName(self.dlg,
                                                    "Select_road_file_",
                                                    "", '*.shp')
    self.dlg.l_inRoad_1.setText(filename)
    self.dlg.l_inRoad_2.setText(filename)

def set_combobox_fields(self, filename, combobox, isDate = True):
    combobox.clear()
    sampleField = -1
    with open(filename) as csvfile:
        dialect = csv.Sniffer().sniff(csvfile.read(1024))#TESTAR
        csvfile.seek(0)
        reader = csv.DictReader(csvfile, delimiter=";")
        fields = reader.fieldnames
        for line in reader:
            for idx, field in enumerate(fields):
                try:
                    if isDate:
                        datetime.strptime(line[field], '%Y-%m-%d')
                    else:
                        float(line[field])
                    sampleField = idx
                    break
                except ValueError:
                    pass
            if sampleField > -1:
                combobox.addItems(fields)
                combobox.setCurrentIndex(idx)
            else:
                if isDate:
                    QMessageBox.critical(combobox, "Format_error",
                                         "File_has_no_date_column_with_valid_format_(YYYY-MM-DD).")
                else:
                    QMessageBox.critical(combobox, "Format_error",
                                         "File_has_no_numeric_values.")
                break
    return sampleField > -1

def set_date_intervals(self, filename, index, startE, endE):
    with open(filename) as csvfile:
        dialect = csv.Sniffer().sniff(csvfile.read(1024))#TESTAR
        csvfile.seek(0)
        reader = csv.reader(csvfile, delimiter=";")
        dates = []
        for i, line in enumerate(reader):

```

```

    if i == 0:
        continue
    try:
        date = datetime.strptime(line[index], '%Y-%m-%d')
        dates.append(date)
    except ValueError:
        QMessageBox.critical(None, "Format_error",
        "File_has_no_valid_date_at_line_%d." % (i+1))
        break
    minDate, maxDate = str(min(dates).date()), str(max(dates).date())
    startE.setDateRange(QDate.fromString(minDate, Qt.ISODate),
                         QDate.fromString(maxDate, Qt.ISODate))
    startE.setDate(QDate.fromString(minDate, Qt.ISODate))
    endE.setDateRange(QDate.fromString(minDate, Qt.ISODate),
                      QDate.fromString(maxDate, Qt.ISODate))
    endE.setDate(QDate.fromString(maxDate, Qt.ISODate))

def change_interval(self):
    filename = self.dlg.l_inSalt.text()
    self.set_date_intervals(filename, self.dlg.cb_colDate.currentIndex(),
                           self.dlg.de_starttime, self.dlg.de_endtime)

def select_salt_file(self):
    filename, _filter = QFileDialog.getOpenFileName(self.dlg,
                                                    "Select_salt_file_",
                                                    "", '*.csv')
    self.dlg.l_inSalt.setText(filename)
    if self.set_combobox_fields(filename, self.dlg.cb_colDate):
        self.set_date_intervals(filename, self.dlg.cb_colDate.
                               currentIndex(),
                               self.dlg.de_starttime, self.dlg.
                               de_endtime)
    self.set_combobox_fields(filename, self.dlg.cb_colSalt, isDate=False)
#to do: pensar em criar o arquivo csvt associado ao csv

def run_method_1(self):
    arq_bacias = self.dlg.l_inBasin_1.text()
    if arq_bacias == "" or QgsVectorLayer(arq_bacias, "").geometryType()
        != 2:
        return QMessageBox.information(None, "Error_(basins_file):",
        "Please_select_a_polygon_or_a_multipolygon_type_file")
    arq_estradas = self.dlg.l_inRoad_1.text()
    if arq_estradas == "" or QgsVectorLayer(arq_estradas, "").geometryType()
        != 1:
        return QMessageBox.information(None, "Error_(roads_file):",
        "Please_select_a_line_or_a_polyline_type_file")

```

```

    arq_entrada = {"street" : arq_estradas, "basin" : arq_bacias}
    tam_buffer = self.dlg_sb_BufferLen.value()
    if tam_buffer == 0:
        return QMessageBox.information(None, "Error_(Buffer_length):",
            "Please_choose_a_buffer_length" )
    unit=""
    if self.dlg.op_m.isChecked() == True:
        unit = "m"
    if self.dlg.op_km.isChecked() == True:
        unit = "km"
    if self.dlg.op_mile.isChecked() == True:
        unit = "mile"
    if unit == "":
        return QMessageBox.information(None, "Error_(Unit):", "Please_
            select_a_unit" )
    arq_saida = self.dlg.l_outMap1.text()
    if arq_saida == "":
        return QMessageBox.information(None, "Error_(Output):", "Please
            _set_a_output_file")
    dir_trabalho = os.path.dirname(arq_saida)
    opcoes = {"buffer" : tam_buffer, "unit" : unit, "iface": self iface
    }
    method_1(dir_trabalho, arq_entrada, arq_saida, opcoes)

def run_method_2(self):
    arq_bacias = self.dlg.l_inBasin_2.text()
    if arq_bacias == "" or QgsVectorLayer(arq_bacias, "").geometryType
        () != 2:
        return QMessageBox.information(None, "Error_(basins_file):", "
            Please_select_a_polygon_or_a_multipolygon_type_file")
    arq_estradas = self.dlg.l_inRoad_2.text()
    if arq_estradas == "" or QgsVectorLayer(arq_estradas, "").
        geometryType() != 1:
        return QMessageBox.information(None, "Error_(roads_file):", "
            Please_select_a_line_or_a_polyline_type_file")
    arq_sal = self.dlg.l_inSalt.text()
    if arq_sal == "":
        return QMessageBox.information(None, "Error_(salt_file):", "
            Please_select_a_valid_csv_file")
    length=""
    if self.dlg.op_galmile.isChecked() == True:
        length = "galmile"
    if self.dlg.op_literkm.isChecked() == True:
        length = "literkm"
    if self.dlg.op_tonmile.isChecked() == True:
        length = "tonmile"
    if self.dlg.op_kgkm.isChecked() == True:

```

```

        length = "kgkm"
if length == "":
    return QMessageBox.information(None, "Error_(Unit):", "Please_
        select_a_unit")
arg_entrada = {"street" : arg_estradas, "basin" : arg_bacias, "salt
    " : arg_sal}
arg_saida = self.dlg.l_outMap2.text()
if arg_saida == "":
    return QMessageBox.information(None, "Error_(Output):", "Please
        _set_a_output_file")
dir_trabalho = os.path.dirname(arg_saida)
t0 = datetime.strptime(self.dlg.de_starttime.date().toString(Qt.
    ISODate), '%Y-%m-%d')
t1 = datetime.strptime(self.dlg.de_endtime.date().toString(Qt.
    ISODate), '%Y-%m-%d')
if t0 == "" or t1 == "":
    return QMessageBox.information(None, "Error_(Time_interval):",
        "Please_choose_start_and_end_dates")
opcoes = {"valueIdx":self.dlg.cb_colSalt.currentIndex(),
            "dateIdx":self.dlg.cb_colDate.currentIndex(),
            "t0":t0, "t1":t1, "length":length, "iface": self.iface}
method_2(dir_trabalho, arg_entrada, arg_saida, opcoes)
#falta conferir os campos de data (start e end)
def run(self):
    """Run method that performs all the real work"""

    # Create the dialog with elements (after translation) and keep
    reference
    # Only create GUI ONCE in callback, so that it will only load when
    the plugin is started
    if self.first_start == True:
        self.first_start = False
        self.dlg = RoadSaltMapDialog()
        self.dlg.setWindowTitle("Road_Salt_Map_Builder")
#map 1
        self.dlg.tb_outMap1.clicked.connect(self.select_outputmap1_file
            )
        self.dlg.tb_inBasin_1.clicked.connect(self.select_basin1_file)
        self.dlg.tb_inRoad_1.clicked.connect(self.select_road1_file)
        self.dlg.pb_ok_1.clicked.connect(self.run_method_1)
#map 2
        self.dlg.tb_outMap2.clicked.connect(self.select_outputmap2_file
            )
        self.dlg.tb_inBasin_2.clicked.connect(self.select_basin2_file)
        self.dlg.tb_inRoad_2.clicked.connect(self.select_road2_file)
        self.dlg.tb_inSalt.clicked.connect(self.select_salt_file)
        self.dlg.pb_ok_2.clicked.connect(self.run_method_2)

```

```

        self.dlg.cb_colDate.currentIndexChanged.connect(self.
            change_interval)

        # Fetch the currently loaded layers
        layers = QgsProject.instance().layerTreeRoot().children()
        # Clear the contents of the comboBox and lineEdit from previous
        # runs

#map 1
        self.dlg.l_outMap1.clear()
        self.dlg.l_inBasin_1.clear()
        self.dlg.l_inRoad_1.clear()

#map 2
        self.dlg.l_outMap2.clear()
        self.dlg.l_inBasin_2.clear()
        self.dlg.l_inRoad_2.clear()
        self.dlg.l_inSalt.clear()
        self.dlg.cb_colSalt.clear()
        self.dlg.cb_colDate.clear()
        self.dlg.de_starttime.clear()
        self.dlg.de_endtime.clear()

        # show the dialog
        self.dlg.show()
        # Run the dialog event loop
        result = self.dlg.exec_()
        # See if OK was pressed
        if result:
            # Do something useful here - delete the line containing pass
            # and
            # substitute with your code.
            pass

```

A.1.3 Camada de controle (roadsalt_mapbuilder_dialog.py)

```

# -*- coding: utf-8 -*-
"""
*****
RoadSaltMapDialog
    A QGIS plugin
cria mapas que dão apoio ao estudo de impaco causado pela despejo de sal
nas estradas
Generated by Plugin Builder: http://g-sherman.github.io/Qgis-Plugin-
Builder/

```

```

-----
begin          : 2019-05-07
git sha        : $Format:%H$ 
copyright      : (C) 2019 by Lais Baroni UERJ
email          : laisrbaroni@gmail.com
*****
/*****



*
*
*   This program is free software; you can redistribute it and/or modify
*
*   it under the terms of the GNU General Public License as published by
*
*   the Free Software Foundation; either version 2 of the License, or
*
*   (at your option) any later version.
*
*
*



"""
import os

from PyQt5 import uic
from PyQt5 import QtWidgets

# This loads your .ui file so that PyQt can populate your plugin with the
# elements from Qt Designer
FORM_CLASS, _ = uic.loadUiType(os.path.join(
    os.path.dirname(__file__), 'roadsalt_mapbuilder_dialog_base.ui'))


class RoadSaltMapDialog(QtWidgets.QDialog, FORM_CLASS):
    def __init__(self, parent=None):
        """Constructor."""
        super(RoadSaltMapDialog, self).__init__(parent)
        # Set up the user interface from Designer through FORM_CLASS.
        # After self.setupUi() you can access any designer object by doing
        # self.<objectname>, and you can use autoconnect slots - see
        # http://qt-project.org/doc/qt-4.8/designer-using-a-ui-file.html
        # #widgets-and-dialogs-with-auto-connect
        self.setupUi(self)

```

A.2 Outros elementos

A.2.1 Inicializador de pacote (`__init__.py`)

```
# -*- coding: utf-8 -*-
"""
*****
RoadSaltMap
    A QGIS plugin
    cria mapas que dão apoio ao estudo de impacto causado pela despejo de sal
    nas estradas
Generated by Plugin Builder: http://g-sherman.github.io/Qgis-Plugin-
Builder/
-----
begin          : 2019-05-07
copyright      : (C) 2019 by Lais Baroni UERJ
email          : laisrbaroni@gmail.com
git sha        : $Format:%H$*
*****
*/
* *
*   This program is free software; you can redistribute it and/or modify
*   it under the terms of the GNU General Public License as published by
*   the Free Software Foundation; either version 2 of the License, or
*   (at your option) any later version.
*
*   *
*   *
* This script initializes the plugin, making it known to QGIS.
"""

# noinspection PyPep8Naming
def classFactory(iface):  # pylint: disable=invalid-name
    """Load RoadSaltMap class from file RoadSaltMap.
```

```

:param iface: A QGIS interface instance.
:type iface: QgsInterface
"""
#
from .roadsalt_mapbuilder import RoadSaltMap
return RoadSaltMap(iface)

```

A.2.2 Arquivo de metadados (metadata.txt)

```

# This file contains metadata for your plugin. Since
# version 2.0 of QGIS this is the proper way to supply
# information about a plugin. The old method of
# embedding metadata in __init__.py will
# is no longer supported since version 2.0.

# This file should be included when you package your plugin.# Mandatory
# items:

[general]
name=Road Salt Map Builder
qgisMinimumVersion=3.0
description=cria mapas que dão apoio ao estudo de impaco causado pela
despejo de sal nas estradas
version=0.1
author=Lais Baroni UERJ
email=laisrbaroni@gmail.com

about=Esse plugin tem objetivo de dar apoio ao estudo ambiental de despejo
de sal para descongelamento de estradas. Para tanto, confeciona mapas
que ajudem na identificação de áreas mais afetadas. Dois mapas são possí
veis: 1) dado um arquivo de estradas, um de polígonos (subbacias de uma
bacia hidrográfica, por exemplo) e um tamanho fixo de buffer para as
estradas, gera um arquivo com dados de proporção de área de estrada por
área do polígono e gera um mapa temático correspondente 2) dado um
arquivo de estradas, um de polígonos (subbacias de uma bacia hidrográ
fica, por exemplo) e uma tabela com a distribuição do sal no tempo (em
dias e por unidade de estrada), gera um arquivo com a soma de sal
despejado em cada polígono no intervalo de tempo determinao e gera um
mapa temático correspondente

tracker=http://bugs
repository=http://repo
# End of mandatory metadata

```

```
# Recommended items:

# Uncomment the following line and add your changelog:
# changelog=

# Tags are comma separated with spaces allowed
tags=python

homepage=http://homepage
category=Vector
icon=icon.png
# experimental flag
experimental=True

# deprecated flag (applies to the whole plugin, not just a single version)
deprecated=False
```

A.2.3 Arquivo de recursos (resources.qrc)

```
<RCC>
    <qresource prefix="/plugins/roadsalt_mapbuilder" >
        <file>icon.png</file>
    </qresource>
</RCC>
```

A.2.4 Exemplo de testes de desempenho (teste.py)

```
#plugins
from qgis.core import *
from qgis.gui import *
from PyQt5.QtGui import *
from PyQt5.QtCore import *
from qgis.analysis import *
from qgis.analysis import QgsGeometryAnalyzer
from processing.algs.qgis.QgisAlgorithm import *
import processing
import sys
import os
import qgis.utils

#inputs do usuário
```

```

estradas = "C:/Users/Lais/Documents/GitHub/monografia/Estradas/estradas.shp"
"
bacias = "C:/Users/Lais/Documents/GitHub/monografia/Subbacias/bacias_UTM.
shp"

#bacias: carrega o vetor (mas não mostra na tela)
bacias = QgsVectorLayer(bacias, '')
    #carrega na tela
bacias = iface.addVectorLayer(bacias, '', 'ogr')

#bacias: habilitando edição
caps = bacias.dataProvider().capabilities()

#bacias: criando campo id
    # Prepare processing framework
sys.path.append('C:/Users/Lais/.qgis2/python/plugins')
from processing.core.Processing import Processing
Processing.initialize()
from processing.tools import *

#oarâmetros da função para criação do campo id
parameters = {'INPUT': bacias,'FIELD_NAME':'bacias2','FIELD_TYPE':1,'
    FIELD_LENGTH': 10, 'FIELD_PRECISION':0,'FORMULA':'$rownum','OUTPUT': 'C
    :/Users/Lais/Documents/GitHub/monografia/Subbacias/output.shp' }
# 'TARGET_CRS': 'EPSG:32618' -- outro possível parâmetro

# criando a coluna (já tentei várias maneiras e não consigo fazer funcionar
    )
processing.run('qgis:fieldcalculator', parameters)

bool QgsGeometryAnalyzer::buffer( QgsVectorLayer* bacias, 'bacias2', 10,
dissolve, int bufferDistanceField, QProgressDialog* p )

#estradas: carrega o vetor (mas não mostra na tela)
estradas = QgsVectorLayer(estradas, '')
    #estradas: carrega na tela
estradas = iface.addVectorLayer(estradas, '', 'ogr')

#estradas: mostra os campos existentes na tabela de atributos do vetor
for field in estradas.fields():
    print(field.name(), field.type())

```

```

#####
mc = iface.mapCanvas()
layer = mc.currentLayer()

QgsGeometryAnalyzer().buffer(layer, bacias, 10, False, False, -1)
QgsGeometryAnalyzer().buffer(bacias, "Output.shp", 10, False, True, -1)

QgsGeometryAnalyzer::buffer(bacias, "Output.shp", 10, False, True, -1)

QgsGeometryAnalyzer().buffer(layer,"C:/Users/Lais/Documents/GitHub/
monografia/mapaFim_estradas/teste",bacias,10,False,True,-1)

#-----



#Pasta temporária
pastal = "C:/Users/Lais/Documents/GitHub/monografia/mapaFim_estradas/teste"

#bacias: adicionando novo campo (aqui só adiciona um campo com valores
vazios)
if caps & QgsVectorDataProvider.AddAttributes:
    res = bacias.dataProvider().addAttributes([QgsField('id', QVariant.
String)])
    bacias.updateFields()

https://github.com/qgis/QGIS-Processing/tree/master/scripts
https://opensourceoptions.com/python/pyqgis\_001\_loadingrasters.html
https://www.geodose.com/2018/09/qgis-python-tutorial-add-field-attribute.html
https://gist.github.com/pigreco/6b73be3d7e99faa1b46766a9314f9466

for field in bacias.fields():
    print(field.name(), field.type())


processing.runalg('qgis:fieldcalculator', input_layer, field_name,
    field_type, field_length, field_precision, new_field, formula,
    output_layer)

processing.algorithmHelp("qgis:fieldcalculator")

```

```
processing.run("native:buffer", {'INPUT': estradas,
    'DISTANCE': 10.0,
    'SEGMENTS': 10,
    'DISSOLVE': True,
    'END_CAP_STYLE': 0,
    'JOIN_STYLE': 0,
    'MITER_LIMIT': 10,
    'OUTPUT': 'C:/Users/Lais/Documents/GitHub/monografia/
        mapaFim_estradas/teste'})
```