UNIVERSIDADE FEDERAL DO PAMPA

REBECA EINHARDT FISS

BASE DE DADOS ORIENTADA A GRAFOS PARA ARMAZENAMENTO E MANIPULAÇÃO DE DADOS ESPAÇO-TEMPORAIS DE PRODUÇÃO AGRÍCOLA

Bagé
2021

REBECA EINHARDT FISS

BASE DE DADOS ORIENTADA A GRAFOS PARA ARMAZENAMENTO E MANIPULAÇÃO DE DADOS ESPAÇO-TEMPORAIS DE PRODUÇÃO AGRÍCOLA

Dissertação apresentada ao Programa de Pós-Graduação em Computação Aplicada como requisito parcial para a obtenção do título de Mestre em Computação Aplicada.

Orientadora: Ana Paula Lüdtke Ferreira
Coorientador: Naylor Bastiani Perez

Bagé
2021

Ficha catalográfica elaborada automaticamente com os dados fornecidos pelo(a) autor(a) através do Módulo de Biblioteca do Sistema GURI (Gestão Unificada de Recursos Institucionais).

F543b Fiss, Rebeca Einhardt

Base de dados orientada a grafos para armazenamento e manipulação de dados espaço-temporais de produção agrícola / Rebeca Einhardt Fiss.

103 p.

Dissertação (Mestrado) - Universidade Federal do Pampa, PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO APLICADA, 2024.

"Orientação: Ana Paula Lüdtke Ferreira; Coorientação: Naylor Bastiani Perez".

1. Agricultura de precisão. 2. Bases de dados NOSQL. 3. Geoestatística. 4. Neo4j. I. Título.

REBECA EINHARDT FISS

BASE DE DADOS ORIENTADA A GRAFOS PARA ARMAZENAMENTO E MANIPULAÇÃO DE DADOS ESPAÇO-TEMPORAIS DE PRODUÇÃO AGRÍCOLA

Dissertação apresentada ao Programa de Pós-graduação em Computação Aplicada da Universidade Federal do Pampa, como requisito parcial para obtenção do Título de Mestre em Computação Aplicada.

Dissertação defendida e aprovada em: 16 de abril de 2021.

Banca examinadora:

_____

Profª Drª Ana Paula Lüdtke Ferreira

Orientadora

Universidade Federal do Pampa

_____

Profª Drª Juliana Silva Herbert

UFCSPA

Prof. Dr. Marcos Corrêa Neves

EMBRAPA

Prof. Dr. Sandro da Silva Camargo

UNIPAMPA

Assinado eletronicamente por **ANA PAULA LUDTKE FERREIRA**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em 02/12/2021, às 18:57, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **SANDRO DA SILVA CAMARGO**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em 02/12/2021, às 20:38, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

A autenticidade deste documento pode ser conferida no site https://sei.unipampa.edu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **0684125** e o código CRC **41642C2D**.

AGRADECIMENTO

O primeiro agradecimento não poderia ser a outra pessoa que não meu marido, companheiro e amigo Everton. Que segurou as pontas em casa todo esse tempo, esteve ao meu lado nas noites de estudo e brigas com os algoritmos da dissertação. Que me deu apoio incondicional. Que foi muito mais do que um bom marido e ótimo pai. Agradeço muito ao meu filho, que durante esse período teve a sua mãe ausente em vários dias da semana por conta das viagens, e nos demais dias pelo trabalho e estudos, fez tudo que podia para auxiliar e reduzir a minha carga como mãe. Amo vocês!

Agradeço ao meu irmãozinho Rafael, que me cedeu gentilmente seu sofá, e pela parceria nos dias que passei em Bagé. Agradeço também à minha mãe, sogra, sogro, cunhados e demais familiares e amigos por entender todas as vezes que não pudemos nos fazer presentes nas atividades familiares por conta da sobrecarga de viagens, estudos e trabalho.

Agradeço aos colegas de trabalho e amigos que não mediram esforço para ajustar minhas atividades como docente e estudante, e estiveram sempre presentes quando necessário.

RESUMO

A produção agrícola tem tradição e relevância econômica para o Estado do Rio Grande do Sul, que conta com mais de 7,5 milhões de hectares destinados a lavouras temporárias. Esses números, contudo, ainda podem ser incrementados por meio do emprego de técnicas de agricultura de precisão. A agricultura de precisão trata a variabilidade espacial com de zonas de manejo diferenciadas, visando aumentar o rendimento de espaços onde a produção é inferior. Para a utilização dessas técnicas é necessário um maior conhecimento sobre as variabilidades existentes, que pode ser adquirido, por exemplo, por meio de mapas de fertilidade ou de produtividade. Esses dados, presentes nos mapas, possuem características tanto espaciais como temporais. Uma infraestrutura que armazene e manipule esses dados pode auxiliar no emprego dessas técnicas por pequenos produtores e, ainda, propiciar novas e melhores formas de visualizar as informações. O objetivo desse trabalho é a modelagem de um banco de dados espaço-temporal para o armazenamento de dados advindos da produção agrícola, e a construção de uma ferramenta web para manipulação das informações. O trabalho apresenta o modelo criado utilizando banco de dados de grafos, e compara seu desempenho com o banco de dados relacional, mostrando a superioridade do banco de dados de grafos nas operações de recuperação da informação. Além disso, é apresentado o AGROGRAPH, uma ferramenta web para inserção e consulta de dados georreferenciados com diferentes coletas temporais. A ferramenta conta ainda com a funcionalidade de krigagem sobre qualquer variável armazenada, para uma visualização espacializada de valores contínuos.

Palavras-chave: Agricultura de precisão. Bases de dados NOSQL. Geoestatística. Neo4j.

ABSTRACT

Agricultural production has tradition, and economic relevance for the State of Rio Grande do Sul, which has more than 7.5 million hectares for temporary crops. These numbers, however, can still be increased through the use of precision farming techniques. Precision agriculture aims to increase the production yield by dealing with spatial variability with differentiated management zones. Production maps are an effective tool to acquire knowledge regarding spatial variability. Production yield and other production-related variables have both spatial and temporal characteristics. An infrastructure that stores and manipulates this data can help small producers and researchers use these techniques by providing new and better ways of visualizing the information. This work aims to model a spatio-temporal database for storing data from agricultural production and build a web tool for manipulating the stored values. This work presents a data model created using a graph database, comparing its performance against a relational one, showing the graph database's superiority regarding information retrieval operations. We also present the AGROGRAPH system: a web tool for inserting and consulting georeferenced data with different temporal gathers. AGROGRAPH also implements the kriging geostatistical technique for any stored variable, allowing a spatialized display of continuous variable values.

Keywords: Geographic information systems, Geostatistics, Graph databases, NoSQL databases, Precision agriculture.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
BDTD	Biblioteca Digital Brasileira de Teses e Dissertações
BPMN	*Business Process Model and Notation*
CSS	*Cascading Style Sheets*
CSV	*Comma Separated Values*
EMBRAPA	Empresa Brasileira de Pesquisa Agropecuária
ER	Modelo de Entidade/Relacionamento
JSON	*JavaScript Object Notation*
K	Potássio
HTML	*Hyper Text Markup Language*
IEEE	*Institute of Electrical and Electronics Engineers*
MO	Matéria Orgânica
NDVI	Índice de Vegetação por Diferença Normalizada
NOSQL	*Not Only SQL*
P	Fósforo
PDF	*Portable Document Format*
PH	Potencial Hidrogeniônico
PHP	*PHP: Hypertext Preprocessor*
PIB	Produto Interno Bruto
RS	Rio Grande do Sul
SGBD	Sistema Gerenciador de Banco de Dados
SIG	Sistemas de Informação Geográfica
SQL	*Structured Query Language*
UML	*Unified Modelling Language*

UNIPAMPA	Universidade Federal do Pampa
WWW	*World Wide Web*
XML	*eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

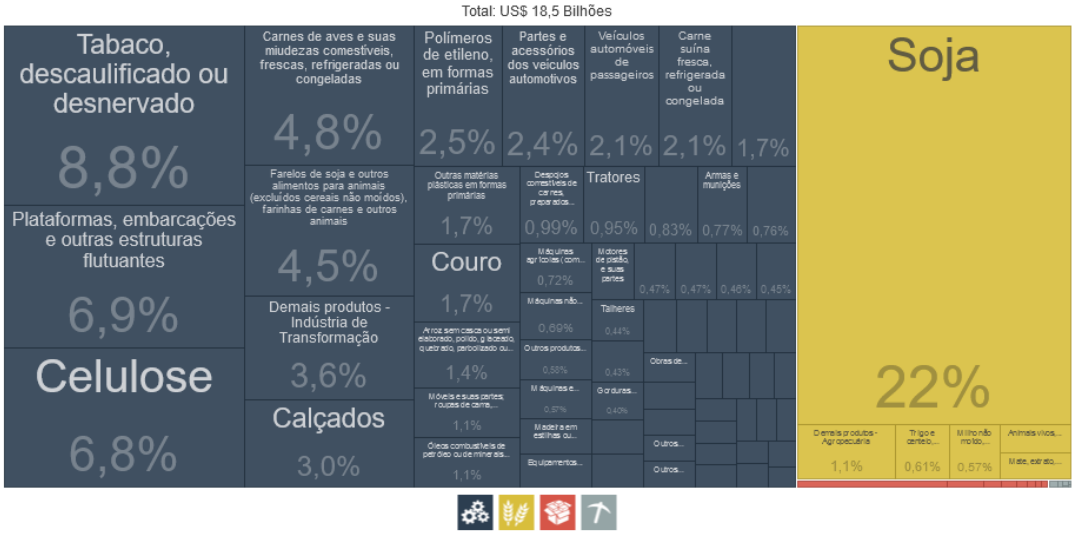
O Estado do Rio Grande do Sul (RS) se destaca pela produção agropecuária com efetiva participação nos indicadores de produção nacionais. Segundo o IBGE (Instituto Brasileiro de Geografia e estatística), no ano de 2019 o Estado contribuiu com pouco mais de 16% de toda a produção de soja do país – o equivalente a 18,5 milhões de toneladas, além de produzir 7 milhões de toneladas de arroz em casca e mais de 5 milhões de toneladas de milho. O estado conta com 35% de sua área total destinada a lavouras temporárias, de um total de 21,7 milhões de hectares destinados à produção agropecuária.

No ano de 2019, o estado foi o 4º no ranking brasileiro de exportações, somando mais de 18,5 bilhões de dólares de valor total. A Figura 1 apresenta um gráfico com os produtos exportados pelo RS, onde é verificável uma grande participação dos produtos básicos – em amarelo – e a participação relevante da soja, que movimentou mais de 4 bilhões de dólares durante o ano de 2019 (COMEX..., 2020). Além da participação nas exportações, o setor de produção agropecuária também colabora no Produto Interno Bruto (PIB) do RS, não apenas nos valores referentes à produção propriamente dita, mas também em relação aos segmentos econômicos derivados, como a agroindústria.

O termo *agricultura de precisão* diz respeito a "um sistema de gerenciamento agrícola baseado na variação espacial de propriedades do solo e das plantas encontradas nas lavouras e visa o aumento do lucro, sustentabilidade e proteção do ambiente" (MAPA, 2011). O que caracteriza a agricultura de precisão é o processo de coleta e análise de dados do sistema produtivo, com vistas a determinar ações de manejo que levem a um melhor resultado do sistema como um todo. No caso específico da agricultura, os dados coletados podem dizer respeito aos dados de colheita, produzidos pelas colheitadeiras ou outro sistema automatizado, ações de manejo realizadas nas fases pré e pós-plantio, levantamento de dados de culturas específicas, condições meteorológicas ou amostragem sistematizada de solos, de modo a fornecer dados sobre a variabilidade das culturas e solos em uma determinada área. Esses dados devem ser posteriormente processados por diferentes técnicas (estatística descritiva, geoestatística, inteligência artificial, entre outras) para então fornecer informações que podem ser dispostas em mapas ou em outro formato e que serão utilizadas para processos de tomada de decisão (BERNARDI et al., 2014; COELHO, 2005).

Os resultados da produção agropecuária apresentam características de variabilidade tanto espacial quanto temporal (COELHO, 2005). A variabilidade

Figura 1 – Produtos exportados pelo RS no ano de 2019



Fonte: (COMEX. . . , 2020)

espacial aparece quando diferentes regiões dentro da mesma área plantada apresentam produções muito diferentes, ainda que submetidas às mesmas práticas de manejo e condições meteorológicas; a variabilidade temporal é evidenciada pelas diferenças de produção em diferentes safras, na mesma área de plantio. Ainda que dados instantâneos de produção possam oferecer uma visão relevante ao produtor, é a análise dos dados coletados nas dimensões de espaço e tempo que pode levar a conclusões mais genéricas e acertadas sobre o processo produtivo.

A adoção de técnicas de agricultura de precisão na produção agrícola pode promover impactos positivos, tanto econômicos como ambientais, incrementando os números de produção agrícola do RS, citados anteriormente. Do ponto de vista econômico, o impacto está no aumento de valor da produção em relação aos custos técnicos e de serviços, uma vez que pode haver economia de recursos (pessoal, água, eletricidade, insumos de forma geral). Já do ponto de vista ambiental, a agricultura de precisão pode promover o aumento na qualidade do solo, menor consumo de água de irrigação, e a sustentabilidade dos sistemas agrícolas de produção (COELHO, 2005).

Os processos de coleta de dados podem ser organizados de diversas formas, mas a análise dos dados coletados requer que eles sejam mantidos de forma organizada, com fácil acesso e com possibilidade de transformação eficiente para servirem de entrada para outros sistemas. Sistemas gerenciadores de bancos de dados (SGBD) são, usualmente, a ferramenta de escolha para o armazenamento e a busca de informações em grandes quantidades de dados. Comumente, os dados armazenados em SGBD

são conceitualmente estruturados a partir de um modelo lógico de dados, descrito por meio de um diagrama de entidade-relacionamento, também conhecido como diagrama ou modelo ER, ou simplesmente ER. A partir do modelo ER são derivadas as tabelas do banco, que podem ser consultadas a partir de operações de álgebra relacional, por meio de uma linguagem de consulta. A linguagem de consulta padrão para SGBD relacionais é a linguagem SQL (*Structured Query Language* ou *Linguagem de Consulta Estruturada*). A utilização de SGBD permite que modelos de dados sejam descritos e implementados de forma portável e que as consultas sejam realizadas em alto nível, com total transparência em relação à implementação física do banco (ELMASRI; NAVATHE, 2011; SILBERSCHATZ; KORTH; SUDARSHAN, 2006; HEUSER, 2008).

Apesar de todas as vantagens da utilização de SGDB, implementados e disponíveis em diversas ferramentas proprietárias e livres, o modelo relacional apresenta limitações quando dados com pouca estrutura interna, informações esparsas e/ou características espaciais e temporais precisam ser descritos e armazenados. Ao implementar um banco de dados relacional a estrutura dos dados é pré-definida por construção. Tabelas usualmente representam entidades e seus atributos. Relacionamentos existentes entre entidades são codificadas como atributos. O modelo relacional é adequado quando as entidades possuem um número fixo de atributos e relacionamentos. Ainda assim, a normalização exigida nas tabelas pode tornar operações de busca que envolvam várias entidades computacionalmente custosa. Tecnologias de *datawarehouse* são usadas para minimizar esse problema, ao preço das informações recuperadas rapidamente poderem não estar sempre atualizadas.

Dados espacializados podem possuir tamanhos e estruturas variadas. A área de uma plantação, por exemplo, pode ser representada por um polígono. Como polígonos podem ter qualquer número de vértices, sua representação por meio de atributos fixos pode não ser trivial (CASANOVA et al., 2005). Uma representação relacional adequada de um polígono pode ser feita com uma tupla identificador-ordem-posição, mas a reconstrução do polígono exige consulta ao banco de dados, que opera com uma complexidade computacional mais do que o acesso direto. Por este motivo, algumas aplicações de bancos de dados foram incrementadas com extensões espaciais, para lidar diretamente com este tipo de dados de forma eficiente, como o PostGIS (POSTGIS, 2020) do PostgreSQL (POSTGRESQL, 2020) e o MySQL Spatial Extension do MySQL (MANUAL. . . , 2020).

Dentre as limitações dos bancos de dados relacionais, podemos destacar a

necessidade de um modelo estrutural dos dados, que não é facilmente modificável sem operações de migração completa dos dados armazenados para uma nova tabela com as modificações necessárias. A dificuldade de lidar com informações faltantes também pode levar a problemas de consistência de armazenamento e recuperação da informação. Dados agropecuários com frequência não são completos. Por exemplo, é possível haver análises de solo em determinada área de plantio. Essas análises, contudo, não são feitas em toda a área, somente em porções específicas. Novas e diferentes análises também podem ser realizadas em diferentes tempos e locais. Assim, associar atributos fixos a localizações em uma área não é uma estratégia adequada.

Como alternativa para melhora no desempenho do armazenamento e consultas em bancos cujos dados não possuem uma estrutura interna compatível com o modelo relacional, começaram a ser desenvolvidos na última década sistemas gerenciadores de bancos de dados com menores exigências de estruturação interna, que passaram a ser conhecidos pelo termo NOSQL (*Not Only SQL*) (ROBINSON; WEBBER; EIFREM, 2015; HAN et al., 2011; CATTELL, 2011). Os bancos de dados NOSQL são categorizados de acordo com suas estruturas de armazenamento. Correntemente, os modelos mais conhecidos são orientados a: chave-valor, documentos, colunas e grafos. A estrutura de armazenamento subjacente determina as propriedades e o uso associado de cada um desses modelos de banco (SADALAGE; FOWLER, 2013).

Os bancos do tipo chave-valor são amplamente utilizados para bancos em que não se tem uma estrutura padrão de dados, os relacionamentos entre dados não são explícitos no modelo, e dessa forma pode surgir a necessidade de se trabalhar com duplicidade de dados para especificar relações. (SULLIVAN, 2015; SADALAGE; FOWLER, 2013). Os bancos de dados de documentos possuem um esquema extremamente flexível, onde os dados são organizados em documentos sem nenhuma restrição estrutural, e os documentos são organizados em coleções, provendo uma estrutura hierárquica de informação. Modelos orientados a documentos foram desenvolvidos especificamente para armazenar dados da *world wide web* (WWW) que é caracterizada por um grande volume de dados heterogêneos e para aplicações onde a consistência dos dados não precisa ser garantida (MEIER; KAUFMANN, 2019; CATTELL, 2011). O modelo de banco de dados orientado a colunas é mais indicado para aplicações que trabalham com a necessidade de replicação de dados visto que, por fazer um controle de versão por meio de um *timestamp*, permite a existência de versões diferentes de um mesmo dado, até sua atualização (SADALAGE; FOWLER, 2013; CATTELL, 2011). Os bancos de dados

orientados a grafos, por outro lado, expressam os dados na forma de um grafo, o que deixa explicita a relação entre os dados representados como vértices, por meio das arestas. Este modelo de armazenamento pode ser utilizado para trabalhar com a inferência de dados, com base nas relações entre eles (FOWLER, 2015).

O armazenamento organizado dos dados de produção agrícola em uma estrutura espaço-temporal permite que esses dados possam ser minerados e analisados a partir de diferentes técnicas estatísticas e de inteligência artificial. Em particular, quando associados a dados de meteorologia (que também são descritos com características espaço-temporais), podem ser utilizados para a descoberta de conhecimento sobre as variáveis de produção com maior impacto no volume e valor de uma colheita em determinada área, com vistas a realizar a predição da produção agrícola em diferentes situações de manejo e de condições meteorológicas.

1.1 Justificativa

As dificuldades na implantação de técnicas de agricultura de precisão nas propriedades gaúchas e brasileiras são várias, mas a questão econômica se sobressai. A maior parte das propriedades gaúchas são pequenas (até 100 hectares) e com gestão familiar. A agricultura de precisão requer, além do maquinário e equipamento sensorial adequado à coleta do tipo de informação necessária, pessoal capacitado em sistemas de informação (tratamento e modelagem da informação, implantação de sistemas de software, gerenciamento de banco de dados, etc.), bastante distante da realidade desse tipo de propriedade, que quase sempre somente conta com mão de obra de baixo nível de escolaridade e, na sua maior parte, focada em processos de produção agrícola.

A Empresa Brasileira de Pesquisa Agropecuária (EMBRAPA), por outro lado, é uma empresa estatal que tem por missão "viabilizar soluções de pesquisa, desenvolvimento e inovação para a sustentabilidade da agricultura, em benefício da sociedade brasileira". Entre as atividades da EMBRAPA, estão a pesquisa e o desenvolvimento, bem como a transferência de tecnologia para os setores produtivos locais. O Programa de Pós-graduação em Computação Aplicada (PPGCAP) parceria entre a Universidade Federal do Pampa e a EMBRAPA Pecuária Sul, é capaz de fornecer a infraestrutura tecnológica e de serviços necessária para o suporte aos pequenos produtores rurais, ao mesmo tempo que pode receber dados de produção local para qualificar o desenvolvimento de novas pesquisas e produtos na área. Este projeto irá auxiliar

produtores e pesquisadores no armazenamento e análise de dados de produção agrícola, por meio de um sistema de bancos de dados acessível, que possa ser posteriormente integrado com outras ferramentas de análise de dados.

1.2 Objetivos

Este trabalho tem como objetivo construir um modelo de dados espaço-temporal na forma de grafo que será utilizado em uma plataforma para armazenamento de dados sobre colheita e demais variáveis de produção agrícola, de forma a servir de fonte para desenvolvimento de relatórios para o produtor e de entrada para sistemas de inferência e de apoio à decisão sobre os dados armazenados.

O trabalho pode, ainda, ser definido em termos dos seguintes objetivos específicos:

- Modelagem de um sistema computacional que permita a construção de bancos de dados espaço-temporais para o armazenamento de dados provenientes de produção agropecuária.
- Desenvolvimento de uma interface web intuitiva que permita ao usuário estabelecer as variáveis de relevância para sua propriedade, inserir dados, produzir relatórios e realizar consultas.
- Desenvolvimento de um conjunto de consultas espaço-temporais para a geração de relatórios e geração de dados para entrada em outros sistemas.
- Implementar métodos geoestatísticos para análise e interpolação de dados com características espaciais.
- Proporcionar uma estrutura unificada para armazenamento de dados sobre sistemas de produção agropecuários de forma que os dados coletados possam ser utilizados para descoberta de conhecimento e outras técnicas de inferência para predição de produção da cultura e apoio à tomada de decisão, em outros trabalhos de pesquisa.

1.3 Organização do trabalho

Este trabalho está organizado da seguinte forma: o presente Capítulo apresentou o contexto de execução e a justificativa para o desenvolvimento deste trabalho, juntamente com os seus objetivos gerais e específicos. O Capítulo 2 apresenta a fundamentação

teórica/técnica sobre assuntos pertinentes ao desenvolvimento do trabalho: a Seção 2.1 discute o armazenamento de dados com características espaço-temporais, a Seção 2.2 apresenta os bancos de dados NOSQL, a Seção 2.3 aborda os bancos de dados NOSQL orientado a grafos e a Seção 2.4 apresenta as definições da área de geoestatística e como essa última é aplicada à agricultura de precisão. Na sequência desse mesmo capítulo, na Seção 2.5, são apresentados os trabalhos correlatos, com considerações sobre seu impacto e relevância no projeto a ser desenvolvido.

O Capítulo 3 apresenta os procedimentos metodológicos e os materiais que foram empregados no desenvolvimento deste trabalho. A Seção 3.1 caracteriza o tipo e as fases da pesquisa, a Seção 3.2 descreve o ferramental tecnológico utilizado para o desenvolvimento do sistema AGROGRAPH e na Seção 3.3 são apresentadas as características e os dados utilizados para abastecer o banco de dados e realizar o teste do modelo criado.

No Capítulo 4 é apresentado o sistema AGROGRAPH, proposto neste trabalho. A Seção 4.1 descreve a arquitetura de três camadas utilizada para o desenvolvimento do sistema, a Seção 4.2 apresenta o modelo de dados que dá suporte ao sistema e na Seção 4.3 é apresentada a interface do sistema AGROGRAPH e suas funcionalidades.

O Capítulo 5 é composto por três seções, nas quais são apresentados os resultados dos teste do modelo de banco de dados desenvolvido (Seção 5.1), o funcionamento da operação de Krigagem no sistema AGROGRAPH (Seção 5.2) e a análise de desempenho comparativa entre o banco de dados relacional (MySQL) e o banco de dados de grafos (Neo4J) (Seção 5.3), usado neste trabalho. O último capítulo (Capítulo 6) apresenta as considerações finais a respeito do trabalho (Seção 6.1) e discute possibilidades de trabalhos futuros (Seção 6.2). Na sequência são apresentadas as referências bibliográficas utilizadas. Ao final do documentos são encontrados os Apêndices A, B, C, D e E, com os algoritmos desenvolvidos para a implementação da Krigagem e para os testes de desempenho entre os diferentes paradigmas de bancos de dados.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Armazenamento de dados espaço-temporais

Dados oriundos de produção agrícola possuem dois aspectos que devem ser considerados em sua análise: o aspecto espacial, que diz respeito à localização onde o dado foi coletado; e o aspecto temporal, que representa o momento da coleta. O armazenamento dos dados pode ocorrer de diversas formas, por meio de modelos de representação distintos, de acordo com o tipo de dado e das aplicações que se deseja construir. A escolha do tipo de representação a usar para uma aplicação determina a complexidade dos algoritmos de inserção e busca dos dados lá inseridos. Dessa forma, é importante que a escolha seja guiada por critérios que minimizem a complexidade dos algoritmos que serão usados com mais frequência sobre a base de dados construída.

Na literatura encontram-se diversos modelos de representação de dados espaço-temporais nos quais a representação do tempo quase sempre determina a escolha do modelo. Os modelos de representação tratam o aspecto espacial sempre como um dado geolocalizado, representado normalmente na forma vetorial, porém o aspecto temporal precisa ser tratado de formas diferentes de acordo com a sua natureza, por exemplo, na forma de instantâneo, onde o tempo é apenas um valor associado aos dados, ou na forma de validade, onde o tempo representa apenas se aqueles valores são os valores correntes ou não. Em Pelekis et al. (2004) diversos modelos de representação são apresentados e discutidos. Os modelos que mais se adaptam à representação de dados de produção agrícola são aqueles em que o tempo se dá na forma de instantâneo, uma vez que a informação de tempo existe apenas no momento da coleta dos dados. O modelo de instantâneo, apresentado em Langran e Chrisman (1988), é um dos modelos mais simples de ser aplicado a dados de produção agrícola. Nesse modelo o tempo é representado de forma discreta, com adição de data e hora aos dados, sem levar em consideração a relação temporal entre eles no momento de armazenamento; ou seja, não existe uma ordenação entre passado e futuro dos dados. Consultas sobre relações temporais nesse modelo podem ser construídas de forma limitada, por meio da linguagem de manipulação de dados. . Os bancos de dados deste modelo podem ser utilizados de forma estática, onde todos os dados são armazenados a cada instante representado; ou diferencial, onde apenas as alterações que ocorreram entre os instantes são armazenadas novamente, diminuindo assim o espaço de armazenamento, e também a complexidade de algumas consultas.

O modelo de três domínios (YUAN, 1994) representa os dados de valor, tempo e espaço em separado, criando ligação entre eles para representar os processos e fenômenos geográficos, permitindo o armazenamento de forma estática e diferencial. A separação dos aspectos para armazenar os dados permite representar tanto informações de instantâneo como informações de trajetória e objetos em movimento. Esse modelo pode ser utilizado para representar dados de produção agrícola, desde que de forma estática, uma vez que sempre que ocorre uma medição, toda a área é amostrada novamente, e não apenas as modificações. Os modelos em que se trata o tempo de forma contínua ou com elementos em constante modificação – como os modelos de gráfico de histórico e de objetos em movimento – não se adaptam aos dados de produção agrícola, visto que não existe uma medição constante, ou em um curto espaço de tempo, para grande parte das características da produção.

A escolha do modelo de representação de dados espaço-temporal guia a escolha do tipo SGBD ou do paradigma de banco de dados a utilizar. O modelo de instantâneo, por simplesmente adicionar um atributo temporal ao dado, é fácil de ser traduzido em qualquer paradigma de banco de dados. O modelo de três domínios, por trabalhar com os três aspectos em separado, criando uma relação entre eles, quando aplicado a um banco de dados relacional exige a utilização de tabelas e relações para representar um único dado, o que pode gerar uma diminuição no desempenho das consultas feitas sobre o banco.

Do ponto de vista de armazenamento dos dados em sistemas, muitos trabalhos na literatura realizam o armazenamento dos dados espaço-temporais por meio de bancos de dados relacionais. Exemplos são (LIAN et al., 2010; SCHÄFER, 2012; WALID; EZZEDINE, 2017), entre muitos outros. A implementação dá-se, usualmente, por meio de alguma extensão espacial para um banco de dados relacional. Cita-se, como exemplo, a extensão PostGIS (POSTGIS, 2020) para o banco PostgreSQL (POSTGRESQL, 2020) ou o banco MySQL (MANUAL. . . , 2020).

A estruturação de um banco de dados inicia no processo de modelagem, a partir de um modelo ER onde se descreve – por meio de entidades, relacionamentos e atributos – os elementos do mundo real sobre os quais se deseja guardar informações (SILBERSCHATZ; KORTH; SUDARSHAN, 2006; HEUSER, 2008). O processo de implantação do banco de dados passa pela tradução do modelo conceitual para um modelo lógico onde os dados são representados por meio de tabelas e chaves primárias e estrangeiras, que serão utilizadas para fazer a interligação entre os dados contidos nas tabelas (SILBERSCHATZ; KORTH; SUDARSHAN, 2006; ELMASRI; NAVATHE,

2011).

Uma das características do banco de dados relacional é a necessidade da estruturação prévia do modelo de dados, exigindo uma análise detalhada dos dados que deverão ser armazenados e de suas relações. Caso o sistema sofra alguma atualização, modificando os dados que serão armazenados ou a informação que será retornada, isso implica na necessidade de alterações em tabelas, incluindo atributos, ou até mesmo novas tabelas, e em alguns casos, migração total dos dados do banco de dados (FOWLER, 2015).

A necessidade de antecipação dos dados armazenados tem implicações importantes no armazenamento de dados de produção agrícola. Nesses sistemas, o modelo de dados precisa ser flexível o suficiente para permitir a inserção de novos dados referentes a qualquer variável do ambiente ou do sistema de produção. Ou seja, seria necessário, antes de desenvolver a aplicação, ter conhecimento sobre todas os dados que poderiam ser coletados, formato e tipo desses dados. Alterações em atributos de entidades exigem uma atualização do banco de dados, com inserção de novas colunas para armazenamento desses dados, o que é inviável na prática. A flexibilidade no armazenamento dos dados de produção pode ser conseguida por meio da utilização de um banco de dados NOSQL (FOWLER, 2015; MEIER; KAUFMANN, 2019; SADALAGE; FOWLER, 2013; SULLIVAN, 2015), propostos e construídos justamente para superar essa falta de flexibilidade dos modelos relacionais.

2.2 Bancos de Dados NOSQL

O termo NOSQL (*Not Only SQL*) é utilizado para se referir a bancos de dados que não utilizam uma estrutura baseada em tabelas com tuplas fixas para representação dos dados e, por conseguinte, não fazem consultas com a linguagem SQL (SADALAGE; FOWLER, 2013; MEIER; KAUFMANN, 2019). As novas abordagem de armazenamento de dados baseadas nesse paradigma surgiram para suprir a necessidade de armazenamento de grandes quantidades de dados heterogêneos, com ênfase na disponibilidade dos dados e escalabilidade das aplicações (SULLIVAN, 2015).

NOSQL é uma terminologia de referência a uma coleção de diferentes implementações sobre diferentes princípios de organização de dados. Os modelos mais conhecidos de bancos de dados NOSQL seguem a organização dos dados na forma de *chave-valor*, *colunas*, *documentos* e *grafos*. Em NoSQL (2020) são registradas 225 (duzentos e vinte e cinco) SGBD disponíveis no mercado que fazem uso do paradigma

NOSQL.

O modelo chave-valor foi o precursor dos bancos de dados NOSQL para o armazenamento de dados de forma não relacional (HECHT; JABLONSKI, 2011; SULLIVAN, 2015), e teve seu início com o Amazon Dynamo DB (SERVICES, 2020) e o Google BigTable (GOOGLE, 2020). No modelo chave-valor, os dados são armazenados em pares com a especificação do dado que será armazenado e o seu valor respectivo. Com esse modelo de armazenamento, os dados são livres de estrutura e podem ser de qualquer tipo: inteiro, *string*, matriz, etc. (SADALAGE; FOWLER, 2013). Os possíveis relacionamentos existentes entre os dados não são armazenados no banco, sendo necessário que a aplicação mantenha essas relações e garanta sua consistência. A inexistência de relacionamentos explícitos pode exigir a replicação de dados, que não seria necessária em bancos de dados relacionais (HAN et al., 2011). O modelo chave-valor tem fácil implementação: os dados podem ser armazenados e acessados rapidamente por meio de funções *set* e *get*, respectivamente. Uma desvantagem é que não permite consultas mais elaboradas que envolvam relacionamentos (LÓSCIO; OLIVEIRA; PONTES, 2011). Em NoSQL (2020) são apresentadas mais de 60 ferramentas que utilizam esta forma de armazenamento de dados, e podemos destacar DynamoDB (SERVICES, 2020), Azure Table Storage (MICROSOFT, 2020), Riak (RIAK, 2020), Redis (REDIS, 2020) e Aerospike (AEROSPYKE, 2020).

Os bancos de dados baseados em colunas têm uma estrutura semelhante aos bancos de dados relacionais: os dados são estruturados em colunas (uma unidade composta por chave e valor), super colunas (agrupamentos de dados em colunas) e as famílias de colunas (conjunto de super colunas - formando uma estrutura semelhante a uma tabela do banco de dados relacional) (SADALAGE; FOWLER, 2013). Neste paradigma, ao invés de agrupar os dados por linhas, como acontece nos bancos de dados relacionais, os dados são agrupados por colunas, o que permite que cada linha tenha um conjunto diferente de colunas, permitindo que atributos possam não ter valor, de forma transparente e estruturada. Cada dado possui uma tripla de indexação (linha, coluna e *timestamp*), linhas e colunas são identificadas por uma chave, e o *timestamp* é utilizado para controlar a versão do dado. Pela sua estrutura, esse paradigma é indicado para bancos de dados distribuídos, e permite a duplicidade de valores, motivo para a utilização do *timestamp* (SADALAGE; FOWLER, 2013; CATTELL, 2011). Em NoSQL (2020) são apresentadas 23 ferramentas que utilizam esta forma de armazenamento de dados, e podemos destacar o Hadoop (FUNDATION, 2020c), MapR (MAPR, 2020) e o Cassandra

(FUNDATION, 2020a).

O terceiro modelo é estruturado em documentos, fazendo uso da notação JSON (*JavaScript Object Notation*) ou XML (*eXtensible Markup Language*), com indexação por chave e valores (CATTELL, 2011; FOWLER, 2015). O modelo é o mais flexível de todos, não necessitando de uma estrutura de organização pré-definida para o armazenamento dos dados. O documento é a unidade básica do banco de dados, o equivalente a uma linha do banco de dados relacional. Os documentos são organizados em coleções, podendo ser mesclados em um documento único, e as buscas podem ocorrer tanto por chave, como por valores (SADALAGE; FOWLER, 2013). Esse paradigma é indicado para o armazenamento de grande volume de dados ou armazenamento de arquivos estruturados, como *logs* e postagens de blogs, mas cuja estrutura não precisa ser previamente definida. Em NoSQL (2020) são apresentadas 40 ferramentas que utilizam esta forma de armazenamento de dados, e podemos destacar o MongoDB (MONGODB, 2020), CouchDB (FUNDATION, 2020b) e o Elastic (FUNDATION, 2020d).

Dados de produção agrícola podem ser armazenados em estruturas do tipo chave-valor, visto que quase todos os dados possuem pouca ou nenhuma estruturação interna, além de informações referentes à localização a ao período ou data de coleta. As relações entre os dados, por outro lado, são conhecidas ou podem ser encontradas por mecanismos de descoberta de conhecimento; contudo, esses relacionamentos não são similares àqueles implementados por bases de dados relacionais. A possibilidade de desenvolvimento de consultas mais elaboradas, onde se possa relacionar valores de uma variável às outras variáveis contudo, é restrita em bases com pouca estrutura interna. Pensando na implementação do sistema utilizando um banco de dados do tipo chave-valor, o armazenamento poderia ser orientado pela localização, onde todas as medidas e as datas de medição poderiam estar a ela associadas. Da mesma forma, o armazenamento dos dados por meio de um banco de dados orientado a colunas apresentaria a limitação de não realizar relacionamentos explícitos entre os dados. O armazenamento dos dados utilizando um banco de dados orientado a documentos é uma implementação pouco viável no contexto da agricultura, pois fornece uma estrutura genérica demais para esse tipo de aplicação. Embora qualquer modelo de armazenamento de dados possa ser aplicado a qualquer contexto, com as devidas adaptações, as relações implícitas entre as variáveis agrícolas, seus valores e localizações, podem ser tornadas explícitas por meio de grafos.

Um grafo é uma representação visual de uma relação formalmente definida, que pode ser enriquecida com rótulos, tipos e outras estruturas. Grafos, pela sua

característica visual, são uma base adequada para representar e trocar informações entre desenvolvedores e usuários de sistemas. Muitas representações gráficas utilizadas em Computação para documentação e validação de sistemas, como UML (*Unified Modelling Language*), podem ser formalizados como grafos (FERREIRA, 2005). A Seção 2.3 apresenta os princípios dos bancos de dados orientados a grafos, que serão usados no decorrer deste trabalho.

2.3 Bancos de dados orientados a grafos

Bancos de dados orientados a grafos (*graph databases*) constituem um dos tipos mais conhecidos de banco de dados NOSQL. Nesse paradigma, os dados são armazenados na forma de grafos. Correspondentemente, os elementos usados para a representação de informações são nodos (vértices), arcos (arestas) e propriedades, que podem ser ligadas aos nodos ou às arestas. Os nodos representam os objetos modelados, as arestas estabelecem as ligações entre eles e tanto os nodos quanto as arestas podem ter propriedades, que podem ser distintas, mesmo em elementos do mesmo tipo. O foco para este tipo de implementação é a relação entre os dados por conta da sua estrutura de armazenamento (SADALAGE; FOWLER, 2013), permitindo a representação de relações complexas, e a inferência de informações a partir de algoritmos de grafos (CATTELL, 2011; MEIER; KAUFMANN, 2019).

Banco de dados orientados a grafos são especialmente apropriados nas situações em que o armazenamento envolve registros com poucos atributos, mas que apresentam diversos tipos de relações entre os objetos armazenados. Bancos de dados relacionais podem expressar as conexões entre os dados por meio das chaves estrangeiras, mas o processo de junções entre os dados é computacionalmente custoso; nos bancos de dados de grafos esse processo de relacionamento é nativo e o seu principal objetivo, apresentando melhor desempenho nas consultas (SADALAGE; FOWLER, 2013; FISS; FERREIRA; PEREZ, 2020).

Um dos principais interesses referentes ao armazenamento de dados agropecuários é a inferência de relações existentes entre os dados armazenados, que pode ser realizada a partir da própria estrutura do grafo resultante da inserção dos dados. Outra característica dos dados agropecuários que favorece o uso de uma representação com grafos é a possibilidade de que novos nodos, arestas e propriedades possam ser inseridos a qualquer tempo, sem necessidade de alteração dos dados já disponíveis na base. Essa flexibilidade

é importante, visto que os dados são coletados em diferentes espaços e tempos e de forma parcial em relação às áreas consideradas.

Em Rockenbach et al. (2018) é feita uma comparação entre cinco sistemas gerenciadores de bancos de dados de grafos em relação a características mercadológicas, de projeto e de manutenção. O trabalho concluiu que a ferramenta Neo4J (NEO4J..., 2020a) é o banco que se destaca frente aos demais, por ser mais maduro e garantir a consistência dos dados utilizando *locks*. Por essas razões, além da existência de farta documentação, a ferramenta Neo4j foi a escolha para implementação neste trabalho.

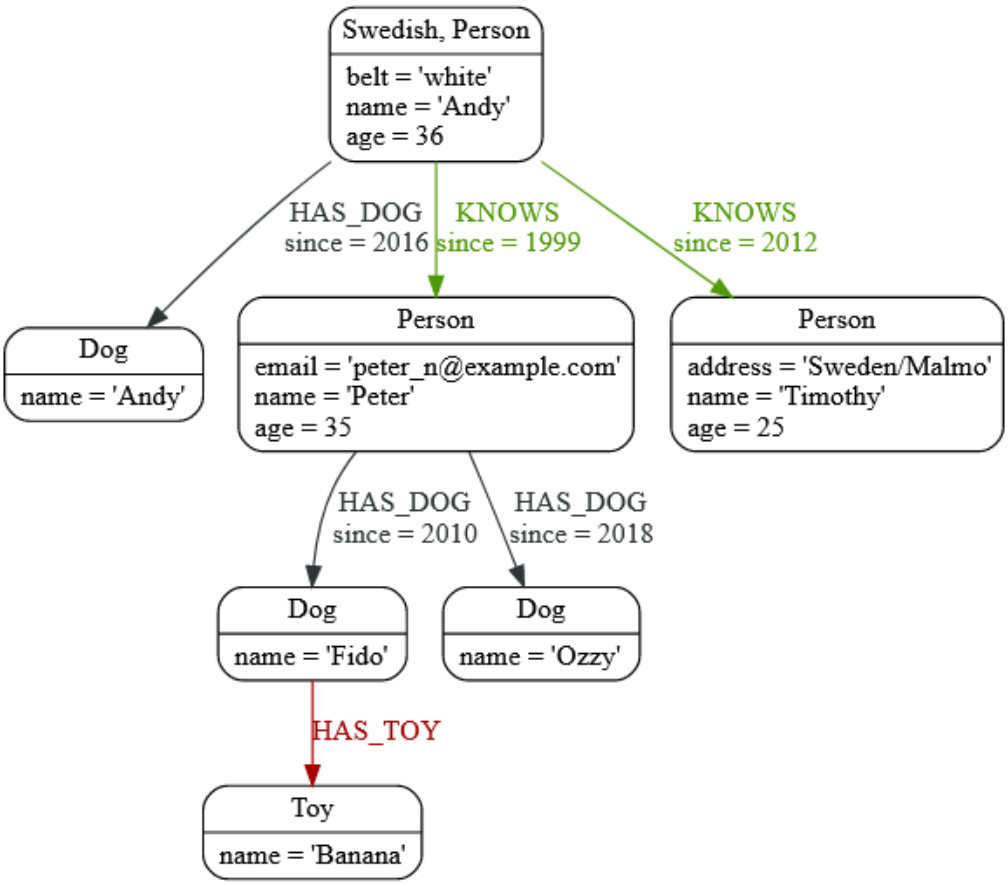
A linguagem Cypher é a linguagem de consulta utilizada pelo Neo4j para manipulação do banco de dados. A linguagem Cypher permite a inclusão, exclusão, alteração e consulta dos dados armazenados. Diferentemente da linguagem SQL, usada em SGBD relacionais, a linguagem Cypher não possui comandos para a definição de esquema de armazenamento, uma vez que os bancos de dados de grafos não possuem essa característica (Neo4J, 2020). Os comandos da linguagem são parcialmente baseados na sintaxe das linguagens SQL, SPARQL, Python e Haskell (Neo4J, 2020).

Toda a informação armazenada está na forma de nodos, arestas ou propriedades. A sintaxe e a semântica informais da linguagem Cypher serão apresentadas a partir do exemplo de Neo4J (2020)[1], cuja estrutura está apresentada na Figura 2. O grafo representado tem sete nodos e seis arestas. Na primeira linha de cada nodo é descrito o seu tipo e, abaixo, suas propriedades, que podem ser vistas como os atributos do nodo, no modelo relacional. As arestas também possuem tipo (apresentado em letras maiúsculas) e, abaixo do nome do tipo da aresta, estão representadas as suas propriedades. Note-se que não existe um esquema de dados similar a um diagrama ER (que também pode ser formalizado como um grafo). A informação e a estrutura estão fundidas na representação dos bancos de dados orientados a grafos.

A recuperação dos dados armazenados no banco é feita com o comando MATCH, que opera de forma similar ao comando SELECT da linguagem SQL. O modificador WHERE pode ser utilizado para definição dos critérios de seleção. O comando RETURN deve ser utilizado em conjunto com o comando MATCH e define quais informações serão apresentadas como resultado da consulta. É possível retornar nodos e arestas, apenas nodos, apenas arestas ou propriedades. Os nodos são representados entre parênteses e as arestas entre colchetes. As informações que estão entre chaves são as propriedades do nodo ou da aresta. As propriedades são armazenadas na forma de chave-valor. Na

[1] https://neo4j.com/docs/cypher-manual/current/clauses/where/

Figura 2 – Grafo para ilustração dos comandos da linguagem Cypher



Fonte: (Neo4J, 2020)

Figura 3 temos cinco exemplos de consultas. A primeira consulta retorna todo o conteúdo do banco de dados, nodos, arestas e propriedades; na segunda, foi selecionado apenas o atributo *name* de todos os nodos: caso um nodo não tenha este atributo, o valor *null* é retornado; na terceira consulta apenas nodos do tipo *Dog* são selecionados e o atributo *name* destes nodos é retornado; a quarta consulta, diferente das anteriores, especifica uma relação, e apenas nodos que satisfaçam esta relação são selecionados: neste caso, são apresentados todas as pessoas e seus respectivos cães. Caso um nodo *Person* não tenha uma relação *HAS_DOG* com um nodo *Dog*, estes nodos não serão retornados pela consulta; na última consulta, é realizado um filtro nos resultados por meio da cláusula WHERE. Essa cláusula permite definir consultas restritas sobre qualquer propriedade, tanto de nodos como de relações; nesse caso, foi sobre a propriedade *since* da relação *HAS_DOG*.

A criação de novos nodos e arestas é feita com o comando CREATE. Na Figura 4 tem-se um exemplo de criação de nodo do tipo *Dog* e de uma relação completa

Figura 3 – Exemplos de consulta utilizando a linguagem Cypher

```
--Retornar todo o conteúdo do banco de dados
MATCH (n)
RETURN n

--Apresentar a propriedade name de todos os nodos
MATCH (n)
RETURN n.name

--Apresentar a propriedade name apenas dos nodos do tipo Dog
MATCH (d:Dog)
RETURN d.name

-- Apresentar as pessoas e seus respectivos cachorros
MATCH (p:Person)-[r:HAS_DOG]->(d:Dog)
RETURN p,r,d

--Apresentar o nome do dono e do cachorro, que tenham
--uma relação iniciada em 2008
MATCH (p:Person)-[r:HAS_DOG]->(d:Dog)
WHERE r.since = 2018
RETURN p.name, d.name
```

Fonte: Autora (2020)

HAS_DOG. A criação de uma relação completa é feita com a criação do nodo *Person* e do nodo *Dog* em conjunto com a aresta entre os nodos. Para se criar uma aresta entre nodos já existentes no banco, é necessário realizar uma consulta MATCH para selecionar os nodos e, após, criar a relação entre estes.

A alteração de informações já existentes, inclusão de novas propriedades em elementos ou exclusão de elementos existentes no banco de dados é feita com um primeiro comando MATCH, para selecionar o elemento ou elementos alvo da ação, seguida da operação desejada. Na Figura 5 temos seis exemplos de *queries* escritas em Cypher: as três primeiras alteram uma propriedade de um elemento e, para isso, utilizam o comando SET; as duas primeiras *queries* possuem o mesmo formato, mas podem apresentar efeitos diferentes, dependendo da estrutura atual do banco de dados: caso uma propriedade utilizada no escopo do comando SET ainda não exista, ela é criada e recebe o valor definido, porém, se esta propriedade já existe no elemento, o valor desta propriedade é alterado para o valor novo. Caso se queira excluir uma propriedade de algum elemento basta atribuir o valor *NULL* a ela. O comando DELETE é utilizado para excluir um elemento do banco de dados, que pode ser uma aresta, como no quarto comando, um nodo, ou ainda um nodo e todas as arestas que fazem ligação com este nodo, como no último exemplo.

Figura 4 – Exemplos de *queries* de criação de nodos e arestas da linguagem Cypher

```
--Criar nodo Dog
CREATE (fido:Dog {name:'Fido'})

--Criar relação completa (2 nodos e 1 aresta)
CREATE (p:Person {email:'peter_n@example.com',
                  name: 'Peter', age:35})
        -[h:HAS_DOG {since:2010}]->(d:Dog {name:'Fido'})

--Criar aresta HAS_TOY entre Fido e Banana
MATCH (d:Dog), (t:Toy)
WHERE d.name='Fido' AND t.name='Banana'
CREATE (d)-[r:HAS_TOY]->(t)
```

Fonte: Autora (2020)

Figura 5 – Exemplos de *queries* de criação de nodos e arestas da linguagem Cypher

```
--Criando a propriedade idade = 5 para o cachorro Fido
MATCH (d:Dog {name:'Fido'})
SET d.age=5

--Alterando a idade do Fido para 8
MATCH (d:Dog {name:'Fido'})
SET d.age=8

--Remover a idade de Fido
MATCH (d:Dog {name:'Fido'})
SET d.age=NULL

--Remover a aresta entre Fido e Peter
MATCH (p:Person {name:'Peter'})-[r]->(d:Dog {name:'Fido'})
DELETE r

--Remover o nodo Fido
MATCH (d:Dog {name:'Fido'})
DELETE d

--Remover o nodo Peter e todas as arestas ligadas a ele
MATCH (p:Person {name:'Peter'})
DETACH DELETE p
```

Fonte: Autora (2020)

A linguagem Cypher, assim como a linguagem SQL, é bastante intuitiva e de fácil compreensão. Uma das principais diferenças entre as linguagens é o tratamento das relações entre os dados, uma vez que a Cypher pode realizar consultas e manipulações sobre um único dado, ou sobre relações inteiras, por meio de seus nodos e arestas de ligação, sem a necessidade de operações de junção. O fato de utilizar expressões semelhantes ao SQL também favorece o aprendizado da linguagem, pois quem já tem conhecimento sobre SQL compreende com mais facilidade as *queries* desenvolvidas em Cypher.

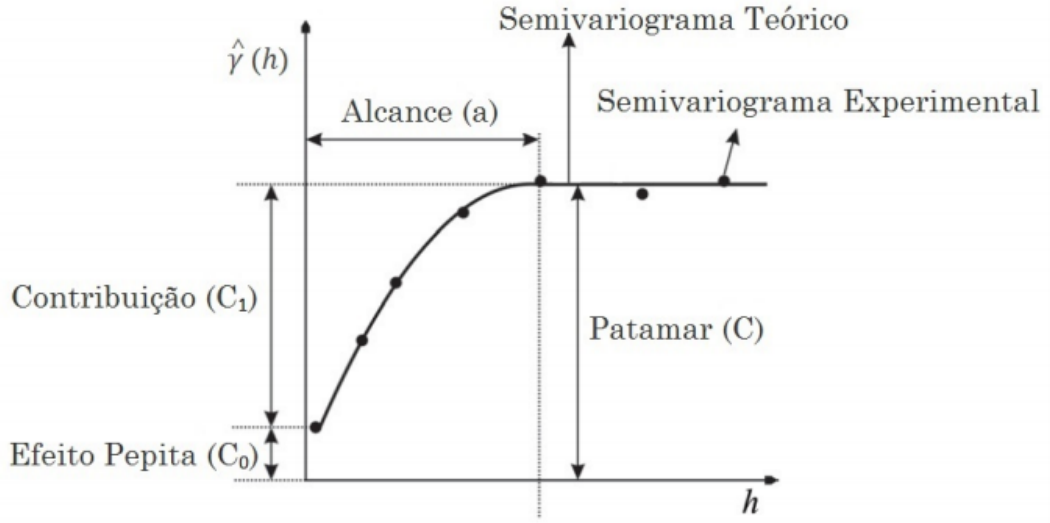
2.4 Geoestatística aplicada à agricultura de precisão

A variabilidade espacial da produção agrícola está intimamente ligada à variabilidade espacial dos atributos físico-químicos do solo (COELHO, 2005). Um dos princípios da agricultura de precisão é tratar essa variabilidade para maximizar a produção (MAPA, 2011). Os valores associados às variáveis coletadas, contudo, não estão disponíveis para a totalidade das áreas plantadas: a mensuração dos valores das variáveis relacionadas ao solo ocorre em pontos específicos e distribuídos pela área de interesse, no intuito de manter custos controlados e, ao mesmo tempo, tentar caracterizar a variabilidade existente. Algoritmos que operam sobre uma área e que necessitam de valores em todos os pontos exigem que a informação coletada seja extrapolada para as localizações em que os dados não foram obtidos. A inferência de valores em áreas com dados desconhecidos é usualmente feita por algoritmos de interpolação (BERNARDI et al., 2014).

Métodos comuns de interpolação não levam em consideração a distribuição espacial das amostras, assumindo-se que os valores são independentes entre si. A independência entre valores, contudo, não expressa a realidade de atributos de solo (SRIVASTAVA, 1996). Como alternativa, Krige (1951) apresenta a geoestatística, em que a dependência espacial das amostras é observada para o cálculo de interpolação. O estudo surgiu da observação em minas de extração de ouro, onde foi observado que os valores de concentração de ouro estavam relacionados com a distância entre as amostras.

A geoestatística pode ser resumida em um algoritmo com três passos: (i) cálculo de semivariância, (ii) construção e ajuste do semivariograma e (iii) interpolação por krigagem (TRANGMAR; YOST; UEHARA, 1986). A semivariância é a medida de dependência dos valores medidos entre duas amostras em função da distância entre elas.

Figura 6 – Gráfico de semivariograma



Fonte: (TEIXEIRA; SCALON, 2013)

Para exemplificar, quanto mais próximas fisicamente forem as amostras, maior será a semelhança entre os valores observados, ou seja, menor será a variância entre os valores, e quanto mais distantes são as amostras maior será a variância entre os valores observados.

O semivariograma é uma função que expressa os valores de semivariância $\hat{\gamma}(h)$ em função da distância h. Os parâmetros do semivariograma estão apresentados na Figura 6: (i) efeito pepita (C_0), ou o valor da semivariância para a distância zero, neste caso, atribuída aleatoriamente; (ii) contribuição (C_1), ou a variância medida entre o efeito pepita e o patamar; e (iii) patamar (C), ou o valor onde o semivariograma se estabiliza. O alcance (a) representa a distância entre os pontos em que existe uma correlação entre os valores observados. A curva traçada no gráfico representa o semivariograma teórico, e os pontos plotados o semivariograma experimental (TRANGMAR; YOST; UEHARA, 1986; MATHERON, 1963).

O semivariograma experimental é aquele obtido por meio do cálculo da semivariância. Os valores do semivariograma experimental são ajustados a um semivariograma teórico. Este modelo teórico fornece a estimativa de forma contínua para os valores da semivariância. Os modelos teóricos mais utilizados são os modelos Exponencial, Esférico e Gaussiano (TRANGMAR; YOST; UEHARA, 1986).

De posse do modelo de semivariograma que melhor se ajusta aos dados coletados, é possível realizar a interpolação para extrapolar os dados coletados às outras localizações. Esse processo de interpolação é realizado por meio da krigagem (BERNARDI et al.,

2014). A krigagem parte dos dados coletados e do modelo de semivariograma ajustado para realizar então a estimação dos valores nas demais localizações em que não foram realizadas medições (TRANGMAR; YOST; UEHARA, 1986; MATHERON, 1963).

A utilização da krigagem como método de interpolação para dados de agricultura se deve ao fato da conhecida dependência espacial entre os valores observados, e outros métodos de interpolação não fazem uso dessa dependência (BERNARDI et al., 2014). Existem modelos diferentes de krigagem, como a krigagem ordinária, algoritmo mais utilizado, onde não há a necessidade de se conhecer a média dos valores medidos, e a krigagem simples, onde a média dos valores é levada em consideração para realizar a predição, resultando em valores mais precisos (SOARES, 2006). Após o processo de interpolação os resultados são espacializados, e normalmente visualizados na forma de mapas de isolinhas em um sistema de informações geográficas (BERNARDI et al., 2014).

2.5 Trabalhos relacionados

O armazenamento de dados de agricultura de precisão de forma integrada utilizando bancos de dados é um problema enfrentado por pesquisadores e produtores. A diversidade de formatos e de métodos de coleta de dados não permite o estabelecimento rígido de parâmetros e modelos de dados. A pesquisa sobre os trabalhos relacionados objetivou conhecer o arcabouço teórico relativo aos bancos de dados e às ferramentas existentes para o armazenamento e manipulação de dados com foco na agricultura de precisão. Os critérios de análise usados foram o paradigma de banco de dados, os modelos de dados utilizado e as funcionalidades integradas por meio de uma ferramenta computacional acessível pela *web*.

Narciso et al. (2005) apresenta uma ferramenta web desenvolvida com o propósito de armazenar dados referentes à produção de milho. Dados georreferenciados obtidos em experimentos em campo foram armazenados em um banco de dados MySQL, apenas usando as referências de latitude, longitude e altitude. Mesmo com as coordenadas, o aspecto espacial não é explorado na ferramenta, tratando os dados apenas como atributos numéricos. O banco de dados armazena informações sobre fertilidade de solo, clima, produtividade entre outros, porém não apresenta o modelo de dados utilizado. Não é possível realizar a inserção de novos dados, apenas a consulta, por meio de uma ferramenta web desenvolvida em HTML e PHP, com o resultado apresentado na forma de tabela. A base de dados organizada tem como objetivo fornecer dados que

serão utilizados em um sistema de recomendação de adubação e correção de solo para maximizar a produção de milho. O sistema solicita ao usuário que sejam informados parâmetros de fertilidade de solo, dados de ambiente e a produtividade desejada, para então apresentar as correções necessárias para a produção. O sistema não leva em consideração a variabilidade espacial dos dados, trabalhando apenas com um único valor para cada parâmetro. O trabalho não apresenta como são realizados os cálculos para recomendações, apenas apresenta a ferramenta desenvolvida e a forma de interação com o usuário. O sistema apresenta algumas limitações, uma vez que não permite que o usuário insira seus dados, não possibilita a visualização dos dados de forma espacializada, além de apresentar os valores de uma única variável por vez no momento das consultas. O aspecto temporal não é considerado no banco de dados, a comparação do mesmo ponto em momentos diferentes não é possível de se realizar.

Em Rocha et al. (2015) é apresentada uma proposta de um webGIS para agricultura de precisão. O objetivo do trabalho é propor a utilização de uma interface webGIS para a visualização de dados de agricultura de precisão georreferenciados. A interface web foi desenvolvida utilizando o *framework OpenLayers* para a apresentação dos dados espaciais. A ferramenta foi abastecida com dados georreferenciados nos formatos matricial e vetorial, transformados em *GeoJSON* para armazenamento no *framework* na forma de arquivo. Além de apresentar os dados no formato de camadas ao usuário, que podem ser sobrepostas, a ferramenta também permite medir distâncias e áreas. A ferramenta não permite a consulta sobre dados ou geração de relatórios, nem mesmo a inserção de novos dados. O trabalho concluiu que a a utilização de uma ferramenta web com aplicação SIG pode ser utilizada para a visualização de dados georreferenciados dentro do contexto da agricultura de precisão. Apesar de demonstrar a possibilidade de utilizar um webGIS, a ferramenta desenvolvida permite somente a visualização de camadas de dados já existentes, sem a possibilidade de filtrar os dados a serem mostrados com base em valores ou localização, cruzar informações de camadas diferentes, gerar relatórios, ou acompanhar a evolução de uma área, já que o aspecto temporal não foi considerado.

3 MATERIAL E MÉTODOS

3.1 Caracterização e fases da pesquisa

O trabalho de pesquisa desenvolvido neste trabalho caracteriza-se como experimental e exploratório, com vistas à construção de um conjunto de funcionalidades em software que permitem atingir os objetivos apresentados na Seção 1.2. O desenvolvimento do trabalho foi composto pelas seguintes fases:

1. Estudo dos princípios e teorias relacionados ao armazenamento de dados com características espaciais e temporais por meio da literatura e trabalhos correlatos.
2. Estudo dos métodos geoestatísticos para análise de dados e suas aplicações, visando posterior implementação dos métodos incorporados à ferramenta.
3. Modelagem de um banco de dados espaço-temporal para armazenar e manipular os dados utilizando o bancos de dados de grafos Neo4J, conforme descrito na Seção 4.2
4. Análise e comparação de ferramentas para armazenamento de dados com características espaço-temporais, comparando desempenho para o armazenamento e manipulação em bancos de dados relacional e de grafo, conforme Seção 5.3
5. Desenvolvimento de uma interface web para armazenamento e manipulação dos dados em um banco de dados conforme o modelo proposto, seu resultado é apresentado na Seção 4.3
6. Desenvolvimento de consultas espaço-temporais e aplicação de métodos de análise geoestatística como funcionalidades da ferramenta, conforme Seção 5.2.

3.2 Ferramental tecnológico

Para a análise e comparação de ferramentas para armazenamento e manipulação de dados com características espaço-temporais foram escolhidos dois SGBD com paradigmas diferentes, o MySQL (relacional) (MANUAL. . . , 2020) e o Neo4J (grafos) (NEO4J. . . , 2020a). Com o objetivo de inferir a complexidade das operações de inserção e recuperação de dados e comparar o desempenho das abordagens relacional e de grafos foram criados nas duas ferramentas bancos de dados em uma perspectiva espaço-temporal, ou seja: dados são georreferenciados e possuem informações sobre

data de coleta. Para a inserção dos dados em ambos os bancos de dados, e a medição de tempo de execução das operações de inserção e recuperação dos dados foram desenvolvidos algoritmos utilizando a linguagem Python (PYTHON. . . , 2020), os gráficos para ilustração dos resultados foram desenvolvidos com a linguagem R (R Core Team, 2013).

Com base nos resultados da análise, a modelagem do banco de dados espaço-temporal foi realizada utilizando o Neo4J 3.5.16 em conjunto com o *plugin* Neo4J Spatial 0.26.2 (NEO4J. . . , 2020b), comparando alguns modelos propostos em termos de resultados de consultas em dados simulados, para a escolha do modelo que melhor representa os dados de valores, dados espaciais e dados temporais.

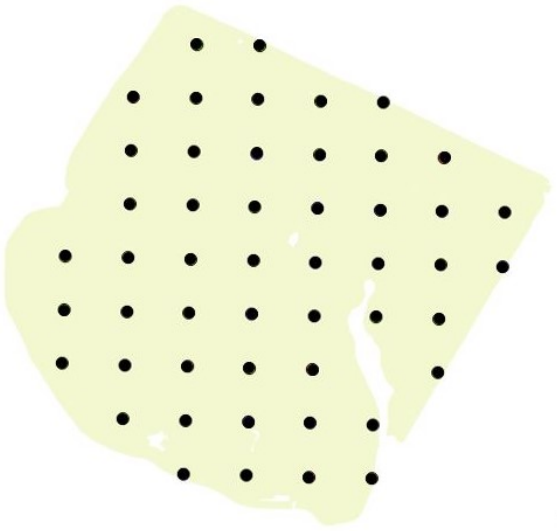
A interface web foi desenvolvida com a linguagem PHP (*PHP: Hypertext Preprocessor*) (The PHP Group, 2020), e integração com o banco Neo4J foi feita com a biblioteca *neo4j-php-client* (GRAPHWARE, 2019). Para a interface gráfica foi utilizado um *template* do Bootstrap (Bootstrap Team, 2020) e a linguagem Javascript (Mozilla, 2020) para a execução de alguns *scripts*. A inserção e recuperação dos dados armazenados no Neo4J foi implementada com a linguagem *Cypher* (Neo4J, 2020), nativa do SGBD. A visualização dos resultados das consultas espaciais fez uso da API (*Application Programming Interface*) do Google Maps (Google Developers, 2020). A aplicação dos métodos geoestatísticos de semivarograma e krigagem fez uso dos métodos da biblioteca *RGeostats* (MINES ParisTech / ARMINES, 2020) da linguagem R. Para ilustração dos fluxos dos módulos do AGROGRAPHfoi utilizada a notação BPMN (*Business Process Model and Notation* por meio da ferramenta Lucidchart (Lucid Software Inc, 2020).

3.3 Dados para teste do modelo

Para a realização dos testes do modelo de dados foram inseridos no sistema dados oriundos de um talhão existente na Embrapa Pecuária Sul, no município de Bagé (RS), com área de 12,49 hectares, com seus pontos representados na Figura 7. Na área de estudo são desenvolvidas culturas de soja e pastagem, de acordo com a época do ano. Os dados inseridos na base são os seguintes:

Análise de solos : Medição de diversos atributos de fertilidade de solos em 50 pontos georreferenciados. Foram utilizadas as medidas de Argila, índice SMP, P (Fósforo), K (Potássio), MO (matéria orgânica) e Relações Ca/Mg (Cálcio/Magnésio)

Figura 7 – Representação da área e dos 50 pontos demarcados onde foram executadas as coletas de dados



Fonte: Autora (2021)

Resistência à penetração : Medição da resistência à penetração de 0 à 40 centímetros de profundidade em 50 pontos georreferenciados. Foi utilizado o valor da média de 0 à 40 centímetros.

Avaliação de massa e crop circle : Avaliação realizada em 50 pontos georreferenciados em uma lavoura de Azevém. Foram utilizadas as medidas de NDVI e Peso seco por hectare.

Os dados originais foram obtidos a partir de dados gerados no período de plantio e colheita das safras na área e por medições efetuadas em diferentes pontos do espaço e tempo. Os resultados do tratamento prévio dos dados foram importados das planilhas desenvolvidas em (FERREIRA, 2019).

Para explorar o aspecto temporal, como foram realizadas apenas uma medida de cada variável, foram utilizados os mesmos pontos e se criou outras 3 medições, por meio da geração valores de aleatórios dentro do intervalo dos valores originais.

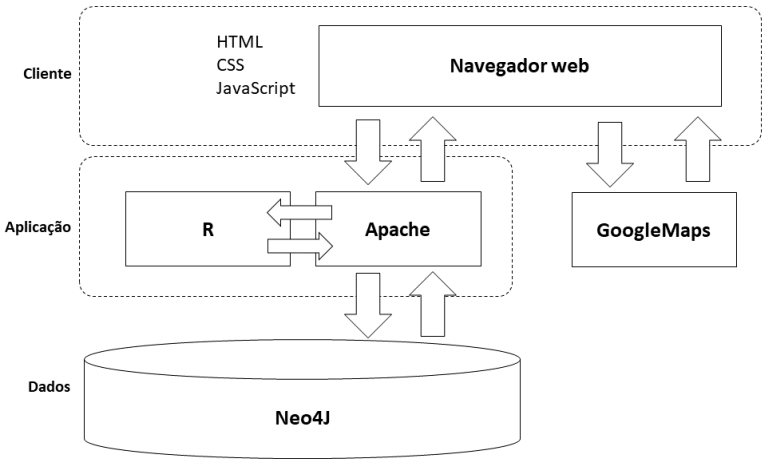
4 O SISTEMA AGROGRAPH

4.1 Estrutura do sistema AGROGRAPH

O sistema AGROGRAPH é um software construído para a interação entre o usuário de o banco de dados construído, contendo ainda a funcionalidade de interpolação por krigagem de quaisquer variáveis armazenadas no banco, dada uma área também escolhida pelo usuário.

A estrutura do sistema AGROGRAPH pode ser representada por três camadas, a saber: (i) camada cliente, (ii) camada de aplicação e (iii) camada de dados. A camada cliente é acessada por meio de um navegador web, a partir de qualquer dispositivo, sendo responsável por interpretar os códigos HTML, CSS e os *scripts* da linguagem JavaScript. Por meio da linguagem JavaScript, o navegador acessa os serviços disponíveis na API do Google Maps. A camada de aplicação do sistema é composta pelos servidores Apache, onde a lógica do sistema é implementada na linguagem PHP. Para o processo de krigagem é utilizado também um *script* da linguagem R, executado no servidor de aplicação do sistema. A camada de dados é composta pelo servidor de banco de dados Neo4j, que realiza sua comunicação com o nível de aplicação por meio da biblioteca GraphAware Neo4j PHP Client da linguagem PHP (GITHUB, 2020). A Figura 8 apresenta as três camadas e seus respectivos componentes.

Figura 8 – Representação de três camadas do sistema AGROGRAPH



Fonte: Autora (2021)

4.2 Modelo de dados

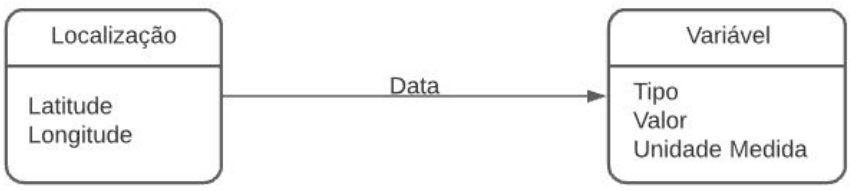
O paradigma de organização dos dados usados é orientado a grafos, implementado sobre a ferramenta Neo4j. Considera-se que os dados apresentam três atributos: sua localização georreferenciada, a data de captura do dado e o seu valor propriamente dito. Para a representação do aspecto espacial se restringiu o tipo de dado à ponto uma vez que usualmente as coletas de dados no âmbito agrícola é representada desta forma, mas independente do tipo de representação espacial é possível aplicar o mesmo modelo de dados. Os dados possuem os tipos e as representações apresentados na Tabela 1.

Tabela 1 – Tipos e representações dos dados armazenados no banco de dados

Aspecto	Tipo de dado	Representação
Espacial	Ponto	Latitude e longitude
Temporal	Data	dd-mm-aaaa
Característica	Variável medida	Tipo (*String*), valor (*Float*) e unidade (*String*)

Os três atributos devem ser traduzido em nodos e arestas para a inserção e recuperação na base de dados, mantendo a integridade. Para a criação do modelo de representação utilizou-se a premissa de que as consultas por localização e por variáveis e seus valores são mais comuns, sendo que a data normalmente é utilizada apenas como critério de seleção. Assim, optou-se pela modelagem utilizando dois nodos, um para representar o aspecto espacial e outro para a variável medida. O aspecto temporal é armazenado como propriedade da aresta de ligação entre estes dados. A modelagem nesse formato faz sentido, visto que o tempo é um atributo da medição do valor do dado e não do dado ou do valor em si. Na Figura 9 podemos observar o primeiro nodo que representa a localização, com as informações de latitude e longitude, o segundo nodo que representa a leitura de uma variável, com seu tipo, valor medido, e unidade de medida (opcional), interligados por uma aresta com a data de leitura do dado.

Figura 9 – Modelo de dados proposto



Fonte: Autora (2021)

Figura 10 – Interface web do sistema AGROGRAPH

Fonte: Autora (2021)

4.3 Interfaces e funcionalidades

A interação entre o usuário e a base de dados é feita por meio de uma interface web, que pode ser acessada a partir de um navegador. A interface desenvolvida tem como objetivo facilitar a interação entre usuário e a base de dados, permitindo que o usuário insira seus dados com características espaciais e temporais, faça consultas, visualizando na forma espacializada ou de relatório e aplique o método geoestatístico de krigagem sobre os dados, que pode ser utilizada para a definição de zonas de manejo ou outras aplicações. Uma das premissas para o desenvolvimento é de que a ferramenta seja simples e de fácil manuseio para não especialistas em Computação ou Geoestatística.

O *template* da interface é simples e responsivo, podendo ser acessado por qualquer tipo de dispositivo, incluindo dispositivos móveis. A apresentação dos dados geolocalizados é feita por meio da API do Google Maps (Google Developers, 2020) . A razão para essa escolha deve-se à familiaridade dos usuários com os produtos da Google™, possibilitando simplicidade e facilidade de operação. A API do Google Maps é difundida na maior parte dos aplicativos e sistemas web que usam geolocalização sendo, provavelmente, de conhecimento dos futuros usuários do sistema. A interface do sistema AGROGRAPH, apresentada na Figura 10, é dividida em três áreas com funcionalidades específicas, a saber: (i) inserção de dados, (ii) consultas e (iii) krigagem, conforme podemos visualizar no *sitemap* da ferramenta, apresentado na Figura 11.

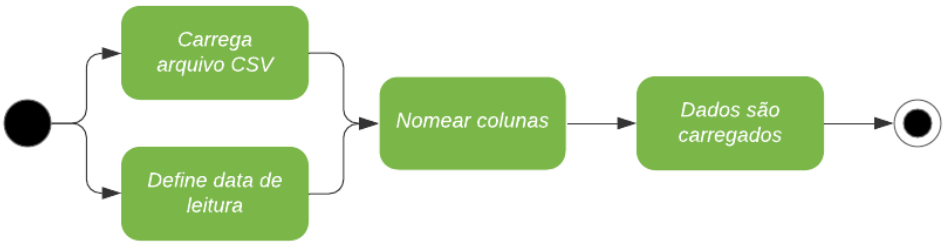
A inserção de dados pode ser realizada de duas formas distintas: por meio de

Figura 11 – *Sitemap* da ferramenta web

Fonte: Autora (2021)

um arquivo CSV (*comma separated values*) ou pela operação direta do sistema, com inserção manual. Em ambos os métodos, o aspecto espacial precisa estar no formato decimal de latitude e longitude, e as coordenadas serão utilizadas para referenciar o ponto geográfico, que receberá as leituras de variáveis posteriormente. Além das coordenadas, o aspecto temporal é inserido na forma de data, e as variáveis terão um tipo, uma unidade de medida (opcional) e o seu valor. Os levantamentos de dados usualmente registram seus resultados em uma planilha eletrônica, contendo os valores medidos e suas respectivas localizações. Dessa forma, optou-se por permitir a submissão de um arquivo CSV contendo a localização do ponto com suas coordenadas de latitude e longitude, os valores das propriedades medidas no respectivo ponto e a data de aquisição dos valores. Os arquivos CSV têm formato de texto puro, podendo ser gerados a partir de qualquer software de planilha eletrônica. O formato de texto puro permite a correção de eventuais erros de registro diretamente no arquivo, em editores de texto disponíveis nos sistemas mais usados. O armazenamento pelo arquivo CSV é realizado conforme o fluxo da Figura 12. Na primeira etapa o usuário deverá selecionar o arquivo e a data de aquisição dos dados. Este arquivo é carregado, e, na tela seguinte, se o arquivo possuir cabeçalho, o usuário deverá confirmar o nome das colunas, caso não possua, deverá preencher os campos com os nomes das colunas, e se desejar, inserir a unidade de medida da variável. Após este processo os dados são armazenados no banco. A Figura 13 apresenta a interface

Figura 12 – Fluxo do processo de armazenamento de dados por arquivo CSV



Fonte: Autora (2021)

Figura 13 – Interface de carregamento do arquivo CSV

Fonte: Autora (2021)

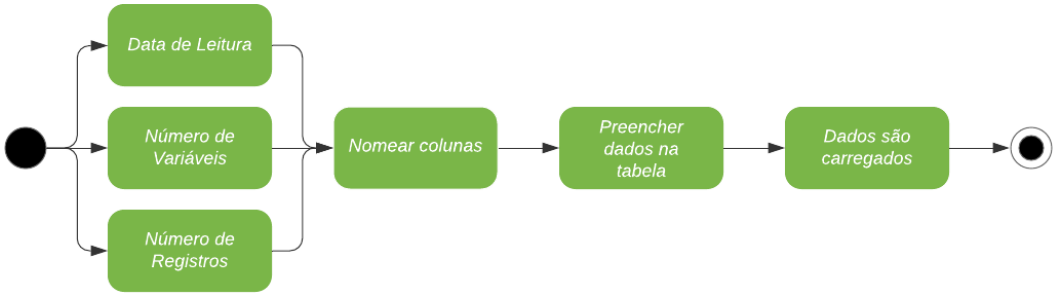
de carregamento do arquivo CSV e na Figura 14 a interface de definição das variáveis listadas no arquivo.

Caso os dados não estejam organizados em um arquivo na forma de tabela, é possível inseri-los manualmente. O armazenamento manual tem um processo similar, e está documentado na Figura 15 o usuário primeiro irá definir a data de leitura das medidas, o número de variáveis que ele deseja armazenar, e o número de registros que serão armazenados (Figura 16). Na sequência será apresentado um formulário onde o usuário deverá preencher o nome das variáveis e as respectivas unidades de medida que desejar (Figura 17), para, posteriormente, preencher a tabela com os dados coletados (Figura 18). Após o envio do formulário os dados são armazenados no banco.

Figura 14 – Interface de definição das variáveis do arquivo CSV

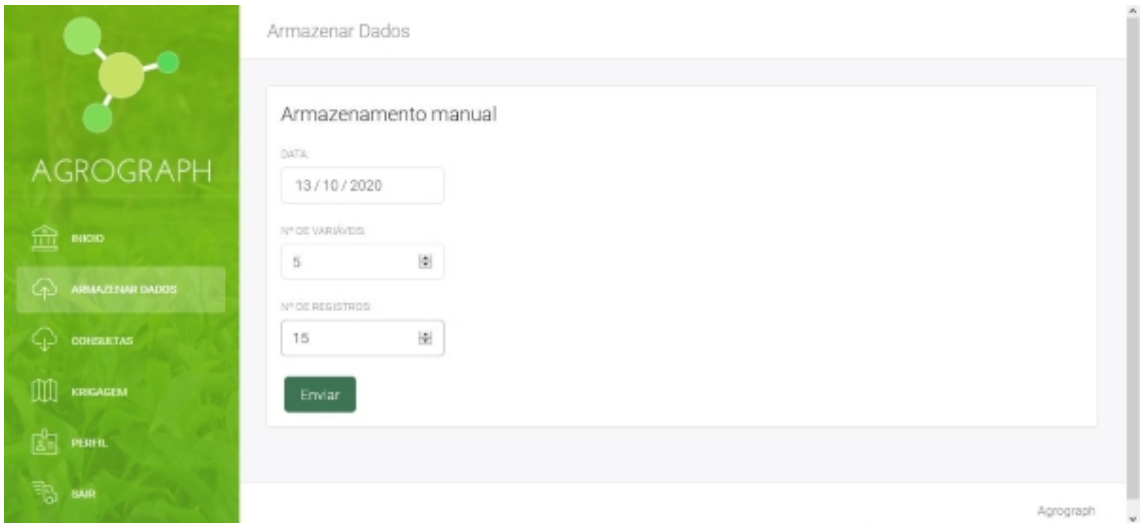
Fonte: Autora (2021)

Figura 15 – Fluxo do processo de armazenamento de dados de forma manual



Fonte: Autora (2021)

Figura 16 – Interface para definição dos primeiros parâmetros para armazenamento manual



Fonte: Autora (2021)

Figura 17 – Interface para descrição das variáveis que serão inseridas

Fonte: Autora (2021)

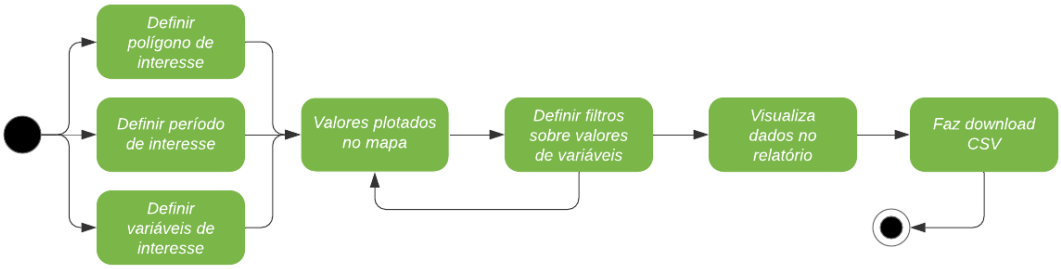
Figura 18 – Interface para digitação dos dados que serão inseridos

Fonte: Autora (2021)

Uma das restrições da versão *community* do Neo4J é não poder definir uma tupla de duas propriedades de um nodo como *unique*, dessa forma, para garantir que um nodo seja único, seja ele de localização, ou de variável, é necessário sempre realizar uma busca no banco de dados antes de realizar a inserção. A necessidade de sempre realizar uma consulta antes de uma inserção de valores faz com que esse processo não tenha a melhor performance.

O segundo módulo do sistema AGROGRAPH é o módulo de consultas. As consultas podem ser feitas a partir dos valores, tempo de registros ou localização geográfica das variáveis armazenadas no banco. O usuário poderá utilizar quaisquer dos três aspectos dos dados para realizar a filtragem da informação no banco. Na Figura 19 podemos ver o fluxo do procedimento para a realização de consultas na ferramenta. As Figura 20 e Figura 21 apresentam a interface utilizada para realizar consultas, o primeiro aspecto que pode ser definido é o espacial, o usuário poderá selecionar no mapa disponível na tela os pontos que delimitam o polígono de interesse; caso não queira utilizar a dimensão espacial na sua consulta, basta não selecionar os pontos delimitantes. O aspecto temporal pode ser explorado selecionando um intervalo de datas, definindo as datas inicial e final, ou apenas uma delas. A escolha das variáveis que serão apresentadas é realizada por meio de um *checklist* com todas as variáveis que constam no banco de dados. Todos os critérios são opcionais, e caso o usuário não selecione algum deles, não será levado em consideração no momento da consulta. Quando definido o polígono de interesse no mapa, as coordenadas dos vértices selecionados são apresentadas abaixo da listagem com

Figura 19 – Fluxo do processo de consulta aos dados armazenados



Fonte: Autora (2021)

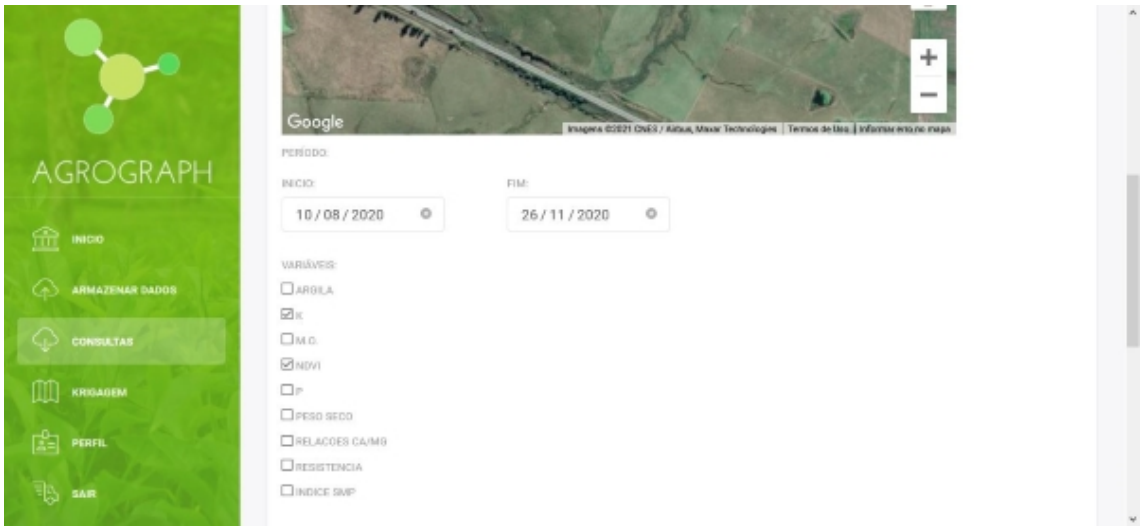
Figura 20 – Interface para definição dos parâmetros de consulta (espacial)

Fonte: Autora (2021)

as variáveis, conforme é possível ver na Figura 22, e se necessário, o usuário pode fazer ajuste nos valores das coordenas antes de enviar para a execução da consulta.

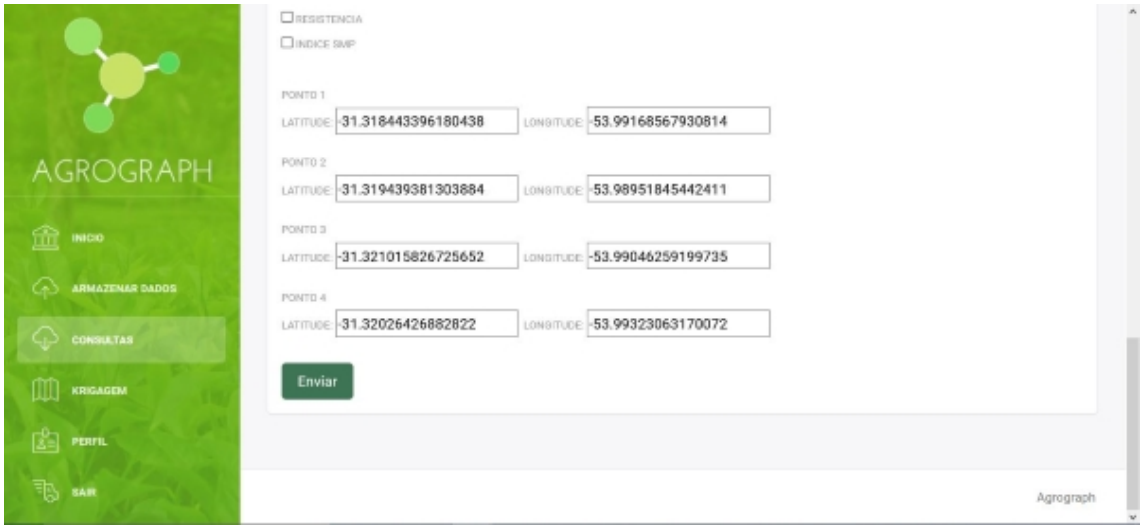
Os resultados das consultas são apresentados na forma de um mapa utilizando a API do Google Maps, com os marcadores nas localizações retornadas e com todas as variáveis da consulta apresentadas no *label*, com os respectivos valores e datas de aquisição, conforme Figura 23. O usuário pode ainda incluir filtros sobre os valores de propriedades constantes no banco de dados, modificando assim os resultados apresentados, utilizando as relações de maior ($>$), menor ($>$) ou igual ($=$). Os resultados das consultas também podem ser apresentados na forma de tabela, com possibilidade de *download* dos dados no formato CSV, conforme Figura 24 . Os dados recuperados pela ferramenta podem servir como entrada em outros sistemas, como de definições de zona de manejo ou de predição de produção.

Figura 21 – Interface para definição dos parâmetros de consulta (temporal e variável)



Fonte: Autora (2021)

Figura 22 – Interface para definição dos parâmetros de consulta (ajuste dos pontos)



Fonte: Autora (2021)

Figura 23 – Interface de apresentação do resultado da consulta

Fonte: Autora (2021)

Figura 24 – Interface de apresentação do resultado da consulta na forma de relatório

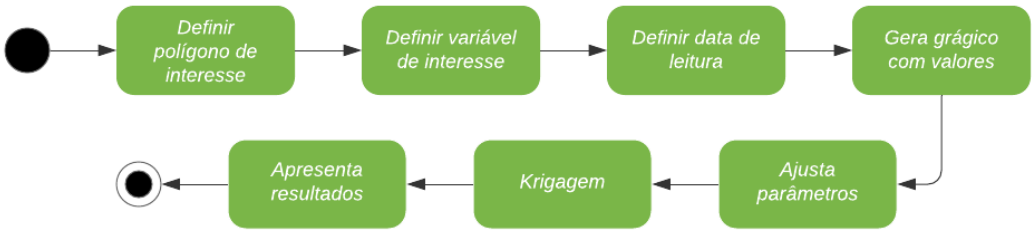
REGISTRO	LATITUDE	LONGITUDE	DATA	TIPO	VALOR
1	-31.320993425	-53.989974	21/09/2020	K	100.66683953442
2	-31.320993425	-53.989974	22/09/2020	K	156.79375718906
3	-31.320993425	-53.989974	25/09/2020	K	96
4	-31.320993425	-53.989974	21/09/2020	P	38.256046221498
5	-31.320993425	-53.989974	22/09/2020	P	14.886159674451
6	-31.320993425	-53.989974	25/09/2020	P	10.9
7	-31.320968544444	-53.991041161111	21/09/2020	K	120.08188197762

Fonte: Autora (2021)

A krigagem (Seção 2.4) é o método de interpolação utilizado para inferir os valores nos pontos não medidos levando em consideração a localização das amostras e é comumente utilizado para a geração de mapas na agricultura de precisão. Para o processo de krigagem, que está descrito no fluxograma da Figura 25, o usuário deverá selecionar a localização que tem como interesse, por meio da seleção dos vértices do polígono no mapa, assim como na consulta, a propriedade da qual se deseja a interpolação e a data de leitura dos dados, conforme apresentado na Figura 36. É importante destacar, que para que não haja erros na consulta ao banco e no processo de krigagem, após a escolha do polígono, é feita uma busca no banco de quais variáveis existem registrada naquela região, e o usuário só poderá escolher uma por meio de uma caixa de seleção, para a definição da data é realizado o mesmo processo.

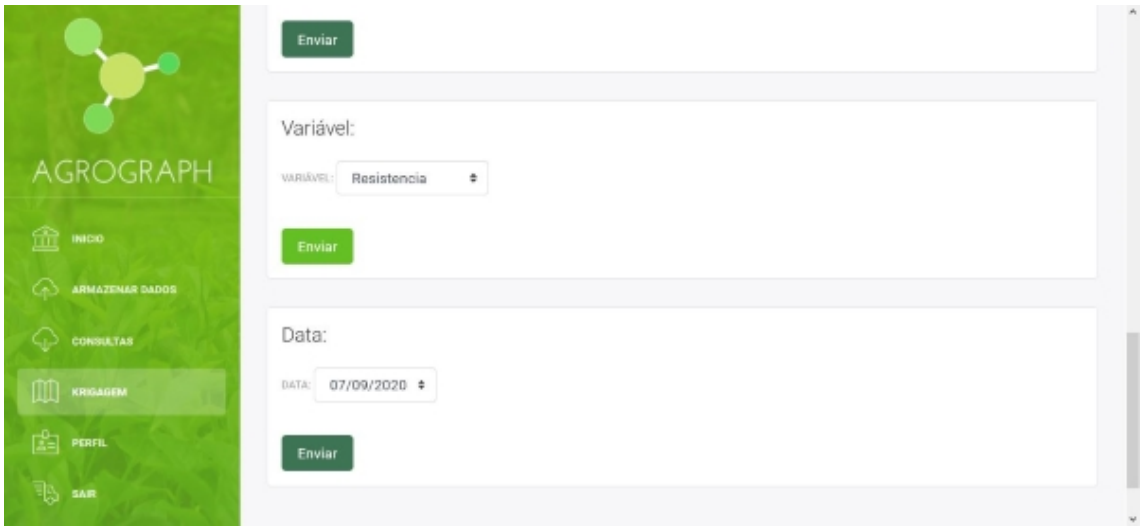
Os valores constantes no banco que satisfazem as condições selecionadas anteriormente são plotadas em um gráfico, para que o usuário posa selecionar o ajuste de alguns parâmetros, se desejar , conforme Figura 27. O usuário poderá modificar o critério de direção da variação, entre isotropia ou anisotropia, e no caso de anisotropia, o usuário deverá indicar o número de direções da variação e o angulo de correção. O usuário pode ainda definir o valor de *nlag*, número de atrasos que devem ser considerados no cálculo da krigagem, caso esse valor não seja definido, o algoritmo irá utilizar o *nlag* padrão. Após o ajuste dos parâmetros, é realizado o processo de krigagem pelo *script* na linguagem R. O resultado da interpolação é apresentado ao usuário na forma de três imagens, a primeira com o gráfico do variograma ajustado ao modelo, a segunda contendo os valores estimados pelo processo de interpolação , e a última o desvio padrão calculado para o mapa de interpolação. A visualização dos dados na forma de um mapa interpolado facilita o processo de tomada de decisão por parte do usuário e pode ser usado como entrada em outras ferramentas de análise de dados. O usuário poderá salvar as imagens resultantes do processo de krigagem ou o *shapefile* com os pontos resultantes do algoritmo de interpolação.

Figura 25 – Fluxo do processo de krigagem



Fonte: Autora (2021)

Figura 26 – Interface de seleção dos atributos para Krigagem



Fonte: Autora (2021)

Figura 27 – Interface de plotagem dos dados observados para definição de parâmetros

Fonte: Autora (2021)

5 RESULTADOS E DISCUSSÃO

5.1 Procedimentos de teste

O modelo de Banco de Dados desenvolvido foi testado sob dois aspectos: (i) verificar se o modelo se adapta ao problema, definindo modelos de testes para verificar o comportamento de nodos e arestas em consultas, para que não haja inconsistências nos resultados, e (ii) consulta por intervalo de valores, uma vez que qualquer medição por meio de sensores, como a localização por GPS, sempre apresenta um erro associado. Adicionalmente, por questões de representação numérica em computadores, valores reais obtidos de fontes diferentes serão necessariamente diferentes, ainda que muito próximos.

Para verificar se o modelo do banco de dados desenvolvido se adapta ao problema foi realizado um teste com base na metodologia descrita em Blanco e Tuya (2015). A metodologia consiste em definir modelos de testes baseados na especificação do sistema, que devem definir nodos e arestas de interesse para então verificar o comportamento desses elementos em consultas ao banco de dados.

Do ponto de vista dos nodos e arestas, podemos definir os seguintes modelos de teste de acordo com a aplicação:

1. Um nodo de localização pode estar relacionado a diversos nodos de variáveis por meio de uma aresta do tipo data, uma vez que a mesma localização poderá ser medida em momentos diferentes e para parâmetros diferentes. Por exemplo, no mesmo local podem ser medidos os valores de Fósforo (P) e Potássio (K).
2. Um nodo de variável pode estar relacionado a diversos nodos de localização por meio de uma data, uma vez que localizações diferentes podem ter o mesmo índice de um determinado parâmetro. Por exemplo, diversos pontos medidos possuem medida de teor de Argila no solo igual a 30.
3. Um nodo de localização pode estar relacionado ao mesmo nodo de variável por meio de datas diferentes. Por exemplo, uma vez que algumas propriedades do solo não possuem grande variabilidade ao longo do tempo, um mesmo local pode manter o mesmo valor medido ao longo do tempo.
4. Um nodo de variável pode estar relacionado a outro nodo variável se ligados por meio de uma mesma localização, permitindo assim relacionar medidas de propriedades de solo, de produção ou outra em um mesmo local.

Figura 28 – Primeira consulta de teste do modelo de dados

```
MATCH (n:Localizacao)←[d:Data]-(v:Variavel)
WHERE n.latitude = -31.320039977778
      AND n.longitude = -53.991545747222
RETURN v.tipo,v.valor, d.data
ORDER BY v.tipo
LIMIT 10
```

Fonte: Autora (2021)

5. Um nodo de localização pode estar relacionado a outro nodo de localização se ligados por meio de uma mesma variável. Pode-se, assim, localizar regiões com medidas semelhantes em relação a uma ou mais variáveis do banco.

Para comprovar que as especificações estão sendo atendidas pelo modelo foram desenvolvidas uma consulta na linguagem Cypher para cada modelo de teste, que posteriormente foram executadas no banco de dados abastecido com dados oriundo de um talhão da Embrapa Pecuária Sul, considerando 9 variáveis, em 50 pontos distintos, com uma medição e duas simulações de valores, para que se pudesse explorar também o aspecto temporal.

Note-se que a estrutura do modelo de dados (Seção 4.2) é idêntica para todas as variáveis do banco. Dessa forma, os modelos de teste acima listados são válidos para quaisquer variáveis ou relações do banco. Os testes apresentados a seguir usam um exemplo de variáveis/relações para cada um dos modelos de teste descritos, mas são válidos para quaisquer outras estruturas similares do banco.

Para o primeiro modelo de teste (alínea 1 da lista de modelos da página 51) realizou-se uma consulta ao banco onde se buscou quais variáveis estavam relacionadas a um determinado ponto, apresentando o nome da variável, o valor medido e a data em que foi realizada a leitura. A Figura 28 apresenta a *query* realizada e na Tabela 2 os resultados retornados pela consulta. Pelo resultado apresentado podemos verificar que o mesmo nodo de localização está relacionado com diversos nodos do tipo variável, garantindo o cumprimento do primeiro modelo de teste.

Para o segundo modelo de teste se realizou uma consulta ao banco onde se buscou quais localizações estavam relacionadas à um determinado nodo do tipo variável, apresentando sua latitude e longitude. Na Figura 29 está apresentada a *query* realizada e na Tabela 3 os resultados retornados pela consulta. Pelo resultado apresentado podemos visualizar que o mesmo nodo de variável está relacionado com diversos nodos do tipo

Tabela 2 – Retorno da primeira consulta de teste do modelo de dados

v.tipo	v.valor	d.data
Argila	29	2020-09-25
Argila	31.591331616044	2020-09-22
Argila	27.1178072430193	2020-09-21
K	80	2020-09-25
K	159.461538018659	2020-09-22
K	90.9866407252848	2020-09-21
M.O.	2.87133745914325	2020-09-22
M.O.	2.3	2020-09-25
M.O.	2.14708670331165	2020-09-21
NDVI	0.85647351934037	2020-09-04

Figura 29 – Segunda consulta de teste do modelo de dados

```
MATCH (n:Localizacao)←[d:Data]-(v:Variavel)
WHERE v.tipo = "Argila" AND v.valor = 30
RETURN n.latitude, n.longitude
LIMIT 10
```

Fonte: Autora (2021)

localização, garantindo assim o cumprimento do segundo modelo de teste.

Para o terceiro modelo de teste se realizou uma consulta ao banco onde foram escolhidos um local e um nodo variável e se buscou quais datas estavam relacionando os nodos, apresentando as datas. Na Figura 30 está apresentada a *query* realizada e na Tabela 4 os resultados retornados pela consulta. Pelo resultado apresentado podemos visualizar que os mesmos nodos de variável e localização estão relacionados por meio de duas arestas com datas distintas, garantindo assim o cumprimento do terceiro modelo de teste.

Para o quarto modelo de teste se realizou uma consulta ao banco onde, com base na

Tabela 3 – Retorno da segunda consulta de teste do modelo de dados

n.latitude	n.longitude
-31.319161175	-53.98991605
-31.320002636111	-53.993146469444
-31.321884669444	-53.991070147222
-31.319098963889	-53.992583897222
-31.320485594444	-53.992093819444
-31.320498038889	-53.991560244444
-31.320510483333	-53.991026666667
-31.320918761111	-53.993175475

Figura 30 – Terceira consulta de teste do modelo de dados

```
MATCH(n:Localizacao{latitude:-31.320039977778,
     longitude:-53.991545747222})←[d]-
     (v:Variavel{tipo:'Argila', valor:29})
RETURN d.data
```

Fonte: Autora (2021)

Tabela 4 – Retorno da terceira consulta de teste do modelo de dados

d.data
"2020-10-12"
"2020-09-25"

medição de Argila no solo registrada em um nodo variável, se buscou valores de NDVI em nodos do tipo variável que estejam relacionados com o mesmo nodo do tipo localização. Na Figura 31 está apresentada a *query* realizada e na Tabela 5 os resultados retornados pela consulta. Pelo resultado apresentado podemos visualizar valores de medição de NDVI que ocorreram na mesma localização onde foi constatada uma medida específica de Argila, garantindo assim o cumprimento do quarto modelo de teste.

Para o quinto modelo de teste se realizou uma consulta ao banco onde se buscou localizações com medições de variáveis iguais à de uma localização específica. Na Figura 32 está apresentada a *query* realizada e na Tabela 6 os resultados retornados pela consulta. Pelo resultado apresentado podemos visualizar outros nodos localização onde a medição de alguma variável é igual à medições que ocorreram na localização escolhida, garantindo assim o cumprimento do quinto modelo de teste.

Por meio das consultas realizadas acima foi possível verificar que os cinco modelos de teste foram aprovados, garantindo assim que o modelo de dados está adaptado às especificações do sistema.

Para verificar se o modelo do banco de dados permite realizar uma consulta por

Figura 31 – Quarta consulta de teste do modelo de dados

```
MATCH (v1:Variavel{tipo:'Argila', valor:29})-[d1:Data]→
      (l:Localizacao)
      ←[d2:Data]-(v2:Variavel{tipo:'NDVI'})
RETURN v2.valor
LIMIT 10
```

Fonte: Autora (2021)

Tabela 5 – Retorno da quarta consulta de teste do modelo de dados

v2.valor
0.64970969903656
0.723574524884536
0.708425581395349
0.802001466543543
0.661890490154112
0.870071739130435
0.84902351490856
0.749766250527903
0.791511111111111
0.851306387582791

Figura 32 – Quinta consulta de teste do modelo de dados

```
MATCH (n1:Localizacao{latitude:-31.320039977778,
      longitude:-53.991545747222})←[d1:Data]-
      (v:Variavel)-[d2:Data]→ (n2:Localizacao)
RETURN n2.latitude, n2.longitude, v.tipo, v.valor
ORDER BY n2.latitude, n2.longitude
LIMIT 10
```

Fonte: Autora (2021)

Tabela 6 – Retorno da quinta consulta de teste do modelo de dados

n2.latitude	n2.longitude	v.tipo	v.valor
-31.321872225	-53.991603730556	K	80
-31.321859780556	-53.992137313889	Argila	29
-31.321859780556	-53.992137313889	Argila	29
-31.321859780556	-53.992137313889	Argila	29
-31.321859780556	-53.992137313889	Relacoes Ca/Mg	2.3
-31.321859780556	-53.992137313889	Argila	29
-31.321426608333	-53.991055652778	K	80
-31.320968544444	-53.991041161111	Argila	29
-31.320968544444	-53.991041161111	indice SMP	5.7
-31.320968544444	-53.991041161111	Argila	29

Figura 33 – Sexta consulta de teste do modelo de dados

```
MATCH (n:Variavel{tipo:'Argila'})
WHERE n.valor ⩾ 28 AND n.valor ⩽29
RETURN n.tipo, n.valor
ORDER BY n.valor
LIMIT 10
```

Fonte: Autora (2021)

Tabela 7 – Retorno da sexta consulta de teste do modelo de dados

n.tipo	n.valor
Argila	28
Argila	28.0245816297829
Argila	28.1396016497165
Argila	28.3064218517393
Argila	28.3877764809877
Argila	28.4873595070094
Argila	28.4971598666161
Argila	28.7093668803573
Argila	28.7207413073629
Argila	28.7348505910486

intervalo de valores foram definidas outras três consultas:

6. Consulta por intervalo de valores de medição de variável, retornando os nodos que satisfazem à condição de um intervalo de valores medidos.
7. Consulta por raio de localização, definindo um ponto central e retornando as localizações próximas, delimitadas por um raio.
8. Consulta por região de localização, delimitando um polígono e retornando os nodos de localização internos ao polígono.

Para verificar que o modelo permite a consulta por um intervalo de valores de medição de uma variável se construiu a sexta consulta utilizando os operadores de maior igual (>=) e menor igual (<=). Os mesmos operadores podem também ser aplicados nos atributos de latitude e longitude dos nodos do tipo localização. Na Figura 33 a *query* realizada busca nodos da variável Argila com valores entre 28 e 29, retornando seu tipo e valor, e a Tabela 7 apresenta os valores retornados.

Para verificar a possibilidade de consulta a partir da definição de um ponto central p e um raio r e retornar quais nodos do tipo localização se encontram na área delimitada, utilizando o método *withinDistance* da extensão *Spatial* do Neo4J (NEO4J..., 2020b).

Figura 34 – Sétima consulta de teste do modelo de dados

```
CALL spatial.withinDistance('layer',
    {latitude:-31.32 , longitude: -53.99}, 1)
YIELD node AS p
RETURN p.latitude, p.longitude
LIMIT 10
```

Fonte: Autora (2021)

Tabela 8 – Retorno da sétima consulta de teste do modelo de dados

p.latitude	p.longitude
-31.3200773	-53.989945025
-31.319619236111	-53.989930536111
-31.320064861111	-53.9904786
-31.320089736111	-53.98941145
-31.320535361111	-53.989959511111
-31.3196068	-53.990464111111
-31.319631672222	-53.989396963889
-31.320522925	-53.990493088889
-31.319161175	-53.98991605
-31.320052422222	-53.991012175

A Figura 34 a *query* realizada busca nodos do tipo localização à uma distância de 1 quilometro do ponto de latitude = -31.32 e longitude = -53.99, retornando as coordenadas de latitude e longitude dos nodos que satisfazem a condição, e a Tabela 8 apresenta os valores retornados.

Para verificar a possibilidade de consulta a partir da definição de um polígono como área de interesse e retornar quais nodos do tipo localização se encontram na área delimitada, utilizando o método *intersects* da extensão *Spatial* do Neo4J (NEO4J..., 2020b). A Figura 35 a *query* realizada busca nodos do tipo localização que estão localizados dentro do polígono definido pelas coordenadas $POLYGON((-53.99 - 31.32, -53.98 - 31.32, -53.98 - 31.33, -53.99 - 31.33, -53.99 - 31.32))$, retornando as coordenadas de latitude e longitude dos nodos que satisfazem a condição, e a Tabela 9 apresenta os valores retornados.

Figura 35 – Oitava consulta de teste do modelo de dados

```
CALL spatial.intersects('layer',
'POLYGON((-53.99 -31.32, -53.98 -31.32, -53.98 -31.33,
 -53.99 -31.33, -53.99 -31.32 ))' )
YIELD node AS p
RETURN p.latitude, p.longitude
LIMIT 10
```

Fonte: Autora (2021)

Tabela 9 – Retorno da oitava consulta de teste do modelo de dados

p.latitude	p.longitude
-31.320089736111	-53.98941145
-31.3200773	-53.989945025
-31.320535361111	-53.989959511111
-31.320993425	-53.989974

5.2 Krigagem no AGROGRAPH

A interpolação por krigagem é o método utilizado para a interpolação de valores que possuam dependência espacial entre seus valores. O método leva em consideração a distribuição espacial dos valores de uma variável e não apenas os valores absolutos medidos. A representação dos valores calculados na forma de um mapa permite que o produtor ou pesquisador possa identificar áreas com características similares. Com o intuito de permitir que os dados constantes no banco de dados do AGROGRAPH fossem utilizados para interpolação sem a necessidade de exportação dos dados e utilização de um software SIG ou de análise de dados, foi disponibilizada uma área na ferramenta que permitisse o cálculo da krigagem. A operação é executada por um algoritmo desenvolvido na linguagem R, utilizando métodos da biblioteca *RGeostats*. O algoritmo recebe como parâmetros os dados armazenados no banco de dados e os parâmetros que o usuário pode ajustar e retorna o gráfico resultante do processo de krigagem. O *script* completo desenvolvido se encontra no Apêndice A. Toda a interação com o usuário ocorre pela interface do AGROGRAPH.

O processo de krigagem inicia pela seleção dos dados que serão interpolados e ajuste de alguns parâmetros:

1. Número de linhas/colunas para a criação da matriz resultante: É necessário definir o número de elementos que a matriz resultante terá, esse valor interfere na precisão

geográfica do modelos, uma vez que quanto menor for o número de elementos na matriz, maior será a área que esse ponto irá representar. O número *n* escolhido será usado para a definição de uma matriz com *n* linhas e *n* colunas. O algoritmo recebe valores inteiros maiores do que 1, e utiliza o valor 50 como padrão.

2. Número de atrasos (*nlag*): É o número de atrasos que serão considerados para cada direção do cálculo da krigagem, o *nlag* é utilizado para o cálculo do *lag*, distância entre os pontos que será considerada para o cálculo da interpolação, e definem a cobertura do variograma. O algoritmo recebe valor inteiro maior do que 1, e utiliza o valor 10 como padrão.
3. Número de direções e ângulo: A distribuição espacial dos atributos de solo pode ter características de isotropia ou anisotropia. Propriedades de solo com característica de isotropia de distribuição espacial são aquelas em que, independente da direção, a distribuição do elemento possui as mesmas características, ou seja, a propriedade de solo observada tem o mesmo comportamento de distribuição para as quatro direções. Já no caso de anisotropia, é necessário definir o número de direções em que existe distribuição espacial da propriedade observada, e ainda o ângulo de distribuição. Para o número de direções o algoritmo recebe valores entre 1 e 4, sendo 4 (isotropia) o valor padrão, e o valor do ângulo pode ser qualquer valor entre 0 e 180, sendo 0 o valor padrão.

Com os parâmetros ajustados, o processo de krigagem inicia pelo cálculo do variograma experimental e ajuste automático do modelo. O trecho do *script* que executa esse processo está na Figura 36. A função *vario.calc* realiza o cálculo do variograma por diversos métodos, recebe como parâmetros os dados, as direções que serão utilizadas no cálculo, a distância que deverá ser considerada e o *nlag*. Após o cálculo do variograma a função *model.auto* verifica os modelos criados, escolhendo assim o variograma que melhor se ajusta aos dados. Com o modelo ajustado é executada a função *kriging* que recebe como parâmetros os dados e o modelo, e retorna uma estrutura de banco de dados contendo o valor estimado e o desvio padrão para cada célula da matriz.

Os dados contidos em *db.grid* podem ser apresentados de algumas formas ao usuário. No caso da ferramenta, os valores calculados são apresentados na forma de um *plot*, conforme a , mas é possível ainda exportar o resultado na forma de *shapefile* composto por pontos que representam os elementos da matriz da krigagem, que fica disponível para download pelo usuário. Este arquivo pode ser utilizado posteriormente em softwares SIG para análises de dados.

Figura 36 – Trecho do *RScript* para cálculo do variograma e ajuste do modelo

```
# Calcula o variograma experimental e o ajuste automático
test.vario <- vario.calc(db.data,dirvect=dirvect,lag=diam/(2*nlag),nlag=nlag)
test.model <- model.auto(test.vario,draw=FALSE)

# Executa o Kriging
db.grid <- kriging(db.data,db.grid,test.model,test.neigh)
```

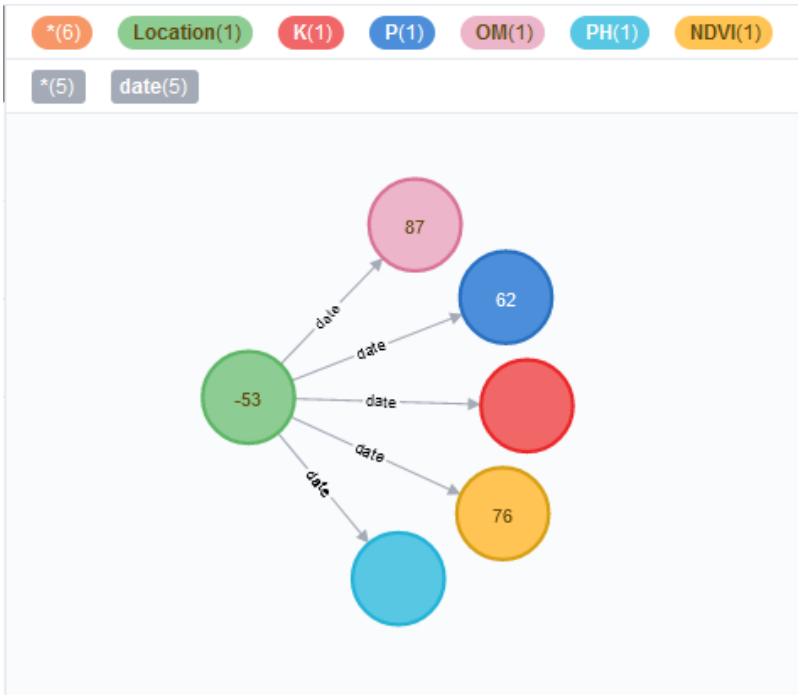
Fonte: Autora (2021)

5.3 Análise de desempenho MySQL x Neo4j

Com base no estudo inicial sobre o armazenamento de dados com características espaço-temporais, foram selecionadas duas possibilidades de ferramentas de armazenamento que satisfaziam as características necessárias para armazenamento de dados oriundos de variáveis agronômicas e de produção: o banco relacional MySQL (MANUAL..., 2020) e o banco orientado a grafos Neo4J (NEO4J..., 2020a). Com o objetivo de inferir a complexidade das operações de inserção e recuperação de dados e comparar o desempenho das abordagens relacional e de grafos, foram criados em cada uma das ferramentas bancos de dados em uma perspectiva espaço-temporal, ou seja: dados são georreferenciados e possuem informações sobre data de coleta. Como o foco da análise é no tempo de execução das operações de inserção e busca, os valores inseridos foram gerados aleatoriamente.

As Figuras 37 e 38 apresentam a modelagem do banco de dados de grafos e relacional, respectivamente. Para o banco de dados de grafos, é criado um nodo do tipo *location* onde são armazenadas as coordenadas geográficas, e este nodo é associado à uma camada de pontos geográficos posteriormente. Para cada variável medida é criado um nodo do tipo correspondente e seu respectivo valor. Uma aresta entre os nodos *location* e da variável com a data em que o valor foi obtido é criada. Na Figura 37 podemos ver um exemplo de um ponto (nodo em verde) associado à nodos de cinco variáveis distintas, no caso OM (matéria orgânica) em rosa, P (fósforo) em azul escuro, K (potássio) em vermelho, NDVI (índice de vegetação da diferença normalizada) em amarelo e PH (potencial hidrogênico) em azul claro. O modelo relacional no MySQL tem uma estrutura mais simples, com uma tabela para cada uma das variáveis medidas, e cada tabela possuí quatro colunas, sendo uma chave primária, a segunda do tipo ponto, onde será armazenada a localização, a data da coleta do dado e o valor da variável. Um índice foi adicionado à coluna localização, uma vez que a maioria das consultas utiliza do aspecto espacial como

Figura 37 – Modelo de dados do banco de dados de grafos.



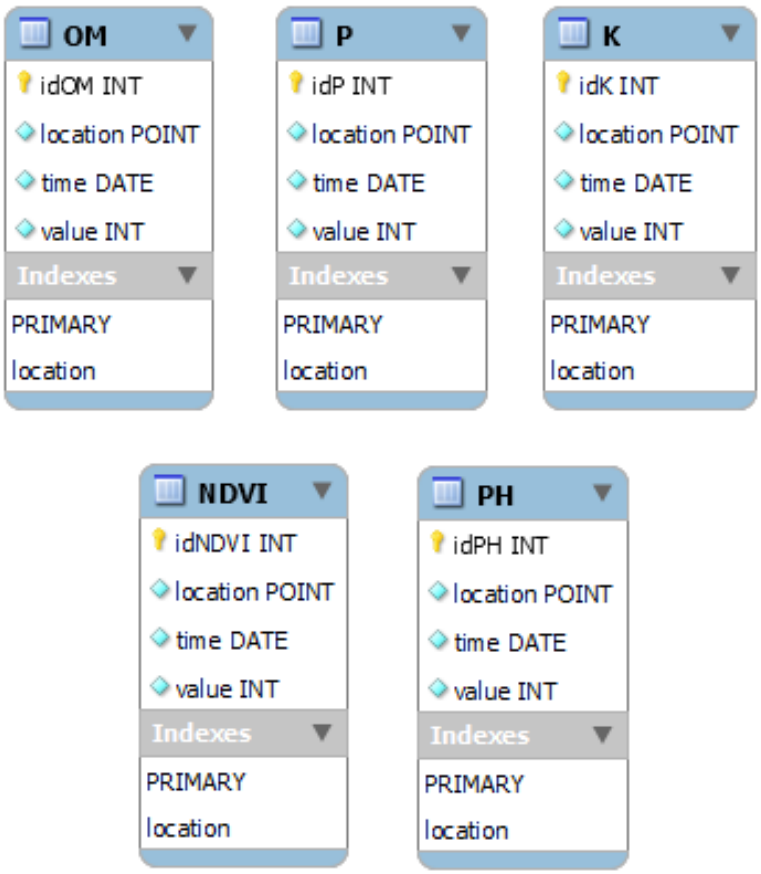
Fonte: Autora (2020)

referência. Na Figura 38 podemos ver a estrutura de cinco tabelas, OM, P, K, NDVI e PH.

A inferência da complexidade da operação de inserção foi realizada medindo o tempo de inserção e consulta de 100.000 (cem mil) registros de uma variável, considerando sua posição geográfica (latitude e longitude), data de aquisição e o valor da variável, em ambos os bancos de dados. Os valores inseridos são aleatórios, considerando uma área de de 100 hectares.

A inferência da função de complexidade das operações de recuperação dos dados foi realizada sobre um banco de dados composto por 5 variáveis distintas e incrementando o número de registros no banco de dados, até 4000 registros, considerando, para cada uma das variáveis, sua localização (latitude e longitude), data de aquisição e o valor da variável. As operações de consulta foram realizadas de três formas distintas: todos os valores armazenados, proximidade de um ponto e valor das variáveis em um ponto específico. Na primeira forma se apresentou todos os valores das variáveis sem realizar nenhuma filtragem, primeiro para uma variável, depois a junção dos resultados para duas variáveis, posteriormente três, quatro e cinco variáveis. Para o segundo tipo de consulta foi definido um ponto central (latitude e longitude) e se buscou quais os valores de uma determinada variável em pontos dentro de um raio r de distância, tendo duas dimensões para o problema, a distância do ponto e o número de variáveis. O valor de r iniciou em 0

Figura 38 – Modelo de dados do banco de dados relacional.



Fonte: Autora (2020)

metros e foi sendo incrementado até 10 quilômetros, em intervalos de 500 metros. Após as buscas para uma variável, o mesmo procedimento foi realizado para duas, três, quatro e cinco variáveis. Esse tipo de busca pode responder à questão "Quais as características desta região?".

Outra questão recorrente em agropecuária é buscar os locais com determinada característica ou as características de um determinado local. Para responder esse tipo de questionamento, foi realizada uma consulta para saber o valor de uma variável em uma localização específica, e posteriormente o valor de duas, três, quatro e cinco variáveis, as consultas foram realizadas dez vezes para se ter uma média de tempo de execução.

Para se inferir o impacto no tempo de execução do número de elementos registrados no banco de dados, todos as consultas foram realizadas nos bancos de dados inicialmente com 100 registros para cada variável, e este número foi sendo incrementado de 100 em 100 registros, até 4000 elementos registrados para cada variável.

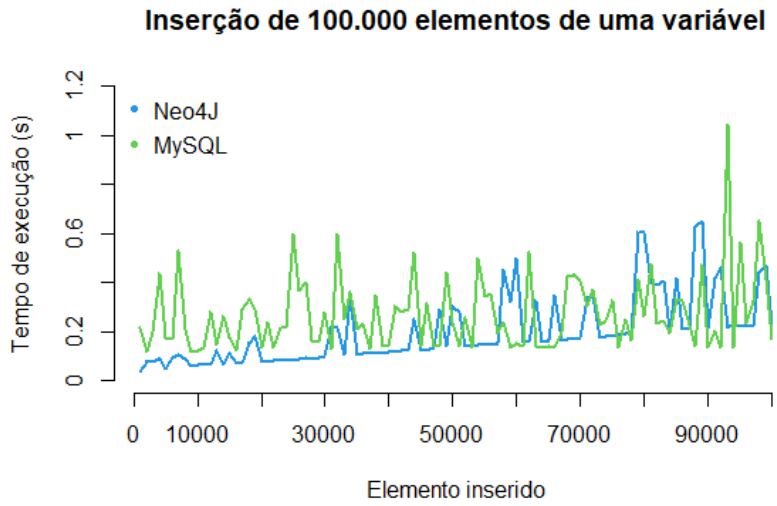
Todos os testes foram realizados utilizando um computador portátil, com sistema operacional Windows 10, processador Core i5 2.5 GHz 2.71 GHz e 8 GB de memória RAM. Para a automatização dos testes foram construídos algoritmos utilizando a linguagem Python (PYTHON..., 2020). O armazenamento dos dados foi realizado utilizando o MySQL Community Server 8.0.19 e o Neo4J 3.5.16 em conjunto com o plugin Neo4J Spatial 0.26.2 (NEO4J..., 2020b). Os gráficos para ilustração dos resultados foram desenvolvidos com a linguagem R (R Core Team, 2013).

Na primeira etapa do desenvolvimento da comparação entre os dois paradigmas de bancos de dados foram realizadas cem mil inserções de uma única variável para ver como se comportam os bancos de dados. Os algoritmos na linguagem Python estão registrados no Apêndice B e os resultados encontram-se apresentados na Figura 39. Para uma melhor visualização, foi calculada a média de mil inserções, totalizando cem pontos no gráfico.É possível visualizar no gráfico que os tempos de execução não apresentam uma curva definida que nos permita inferir a complexidade da operação, bem como as curvas se apresentam sobrepostas na maior parte do tempo, demonstrando que para o processo de inserção não se tem uma variação significativa entre os modelos. Até o elemento 30.000 é possível visualizar pelo gráfico que o Neo4J tem um tempo de execução menor. A Tabela 10 apresenta o tempo para a inserção do primeiro e do último elemento em cada SGBD, além dos valores estatísticos mínimo, máximo, média e o desvio padrão. É possível ver pela tabela que os tempo de inserção no banco de dados relacional é, na maioria das vezes, maior do que o tempo de inserção para os bancos de dados de grafos. Os *scripts* em

Tabela 10 – Tempo de execução das operações de inserção em segundos

SGBD	Primeiro Elemento	Último Elemento	Mínimo	Máximo	Média	Desvio Padrão
MySQL	0.192588	0.126692	0.0321	386	0.1219	4.509377
Neo4J	0.06882	0.264249	0.02394	245	0.13731	3.406393

Figura 39 – Operação de inserção para 100.000 elementos



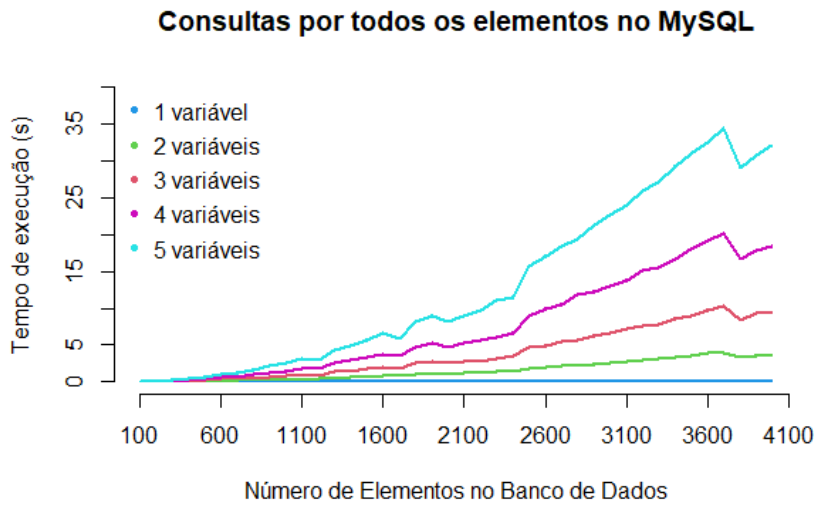
Fonte: Autora (2020)

Python utilizado para realizar os procedimentos de inserção estão transcritos no Apêndice B.

A primeira forma de consulta implementada foi a consulta por todos os elementos do banco, sem nenhuma filtragem, para 1, 2, 3, 4 e 5 variáveis, com o tamanho do banco de dados sendo incrementado de 100 em 100 elementos. Os *scripts* em Python utilizado para realizar as consultas estão transcritos no Apêndice C. A Figura 40 apresenta os tempos de execução para a consulta no banco de dados MySQL. Podemos notar que as curvas possuem características distintas de acordo com o número de variáveis consultadas. Para uma única variável, o tempo de execução tem característica constante, onde existe pouca variação no tempo de execução da consulta em função do número de elementos no banco de dados. Já para as curvas para 2, 3, 4 e 5 variáveis possuem características lineares, onde o tempo de execução vai aumentando gradativamente, de acordo com o aumento na quantidade de elementos armazenados no banco de dados. Os tempo de execução das consultas chega próximo a 35 segundos para a execução de uma *query* com 5 variáveis em um banco de dados com 4 mil elementos armazenados para cada variável.

Já para os dados armazenados no banco de dados de grafos, apresentado na Figura

Figura 40 – Consulta de todos os elementos no banco de dados MySQL



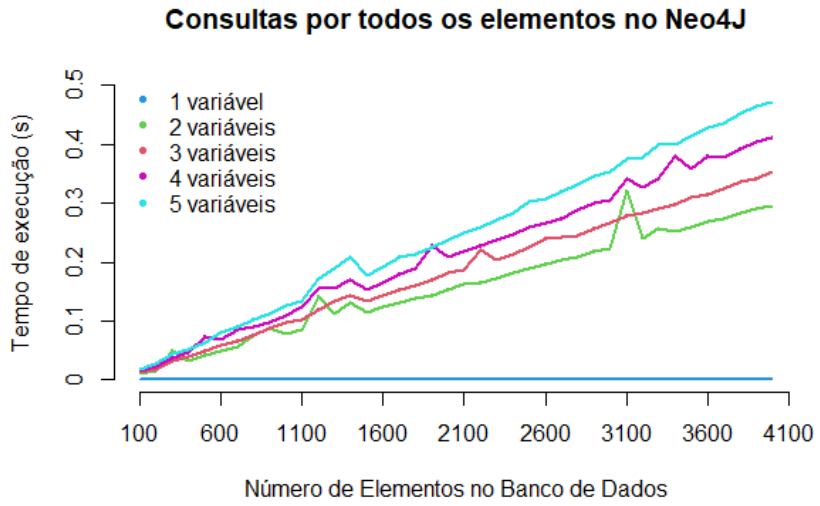
Fonte: Autora (2020)

41 o tempo de execução é muito menor, se comparado com o tempo de execução no banco de dados relacional, para 2, 3, 4 e 5 variáveis. As curvas nestes casos possuem características lineares, porém o tempo máximo de execução não chega à 0.5 segundos. Já para 1 única variável o tempo de execução é praticamente constante e igual a zero.

O segundo tipo de consulta executada foi em função da distância de um ponto central, buscando os valores para as variáveis dos pontos existentes na região. Para observar o tempo de consulta em relação ao tamanho do banco de dados, foi calculada a média para as consultas realizadas incrementando a distância. Os *scripts* em Python utilizado para realizar as consultas estão transcritos no Apêndice D. A figura 42 apresenta as curvas da média de tempo de execução das consultas no banco de dados MySQL. Podemos notar que a média do tempo de execução em bancos de dados de 4000 elementos com 5 variáveis chega a quase 20 segundos. As curvas seguem a mesma tendência das curvas de consulta de todos os elementos no banco de dados, conforme parágrafos acima. Na figura 43 temos as curvas para o banco de dados Neo4J. As curvas para múltiplas variáveis também seguem as mesmas tendências das consultas anteriores, porém para uma única variável é possível notar um pequeno incremento na média do tempo de execução, conforme a distância de r é incrementada. Mas, conforme as consultas anteriores, a maior média de tempo de execução para o banco de dados grafos é muito menor do que a do banco de dados relacional.

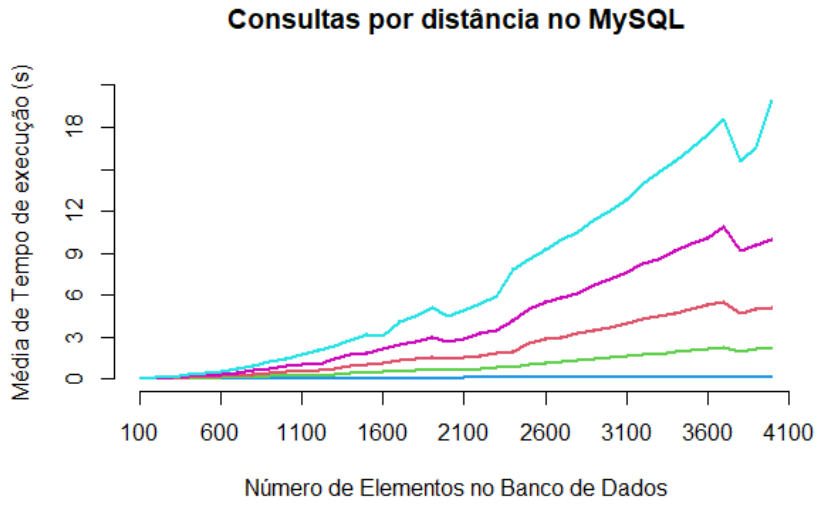
Para compreender a influencia da distância nas consultas as Figuras 44 e 45

Figura 41 – Consulta de todos os elementos no banco de dados Neo4J



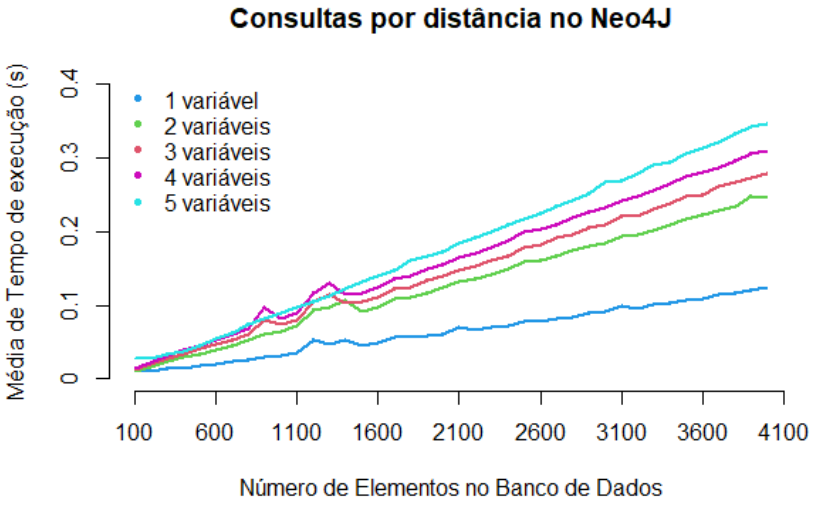
Fonte: Autora (2020)

Figura 42 – Consulta determinada pela distância de um ponto central no MySQL



Fonte: Autora (2020)

Figura 43 – Consulta determinada pela distância de um ponto central no Neo4J

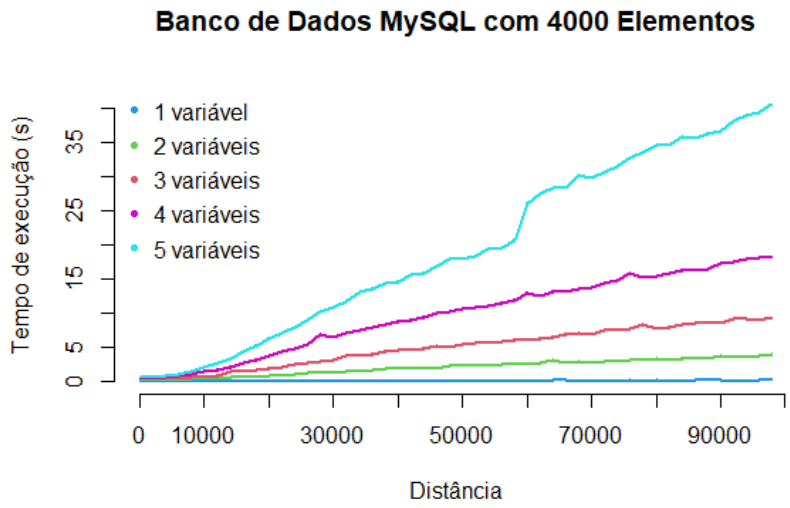


Fonte: Autora (2020)

apresentam as curvas do tempo de execução das *queries* em função do aumento no tamanho do raio *r* e consequente aumento no número de pontos retornados nos bancos de dados relacional e de grafos, respectivamente. Para o MySQL podemos ver que as consultas com uma variável apresentam uma curva constante, com pouquíssima variação, e as curvas para 2, 3, 4 e 5 variáveis possuem características mais lineares, com um aumento gradual no tempo de execução conforme a distância é incrementada. Se comparado o maior tempo de execução do banco de dados de grafos ao do banco de dados relacional, vemos que enquanto no modelo relacional o tempo de execução chega próximo à 42 segundos, no paradigma de grafos este valor não supera 0,8 segundos. As curvas para o Neo4J da figura 45 apresentam características lineares, em que conforme o valor de *r* é incrementado, o tempo de execução também segue esta tendência.

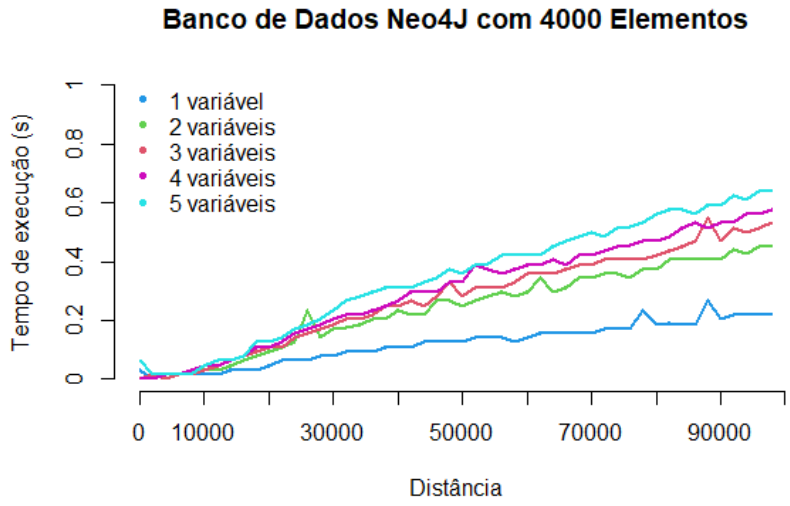
O último tipo de consulta implementada foi a consulta para os valores das variáveis de um ponto específico. Os *scripts* em Python utilizado para realizar as consultas estão transcritos no Apêndice E. As figuras 46 e 47 apresentam as curvas dos tempos de execução das consultas para determinar o valor de variáveis em um ponto específico, no MySQL e no Neo4J respectivamente. No primeiro momento, o que mais chama a atenção é o fato de que, de todos os tipos de consultas realizados até agora, esse é o único tipo de consulta em que a performance do banco de dados relacional é melhor do que do banco de dados de grafos, onde as consultas são quase 10 vezes mais rápidas. As curvas para o tempo de execução das consultas no MySQL apresentam características constante para 1

Figura 44 – Consulta de valores em um raio *r* no banco de dados MySQL



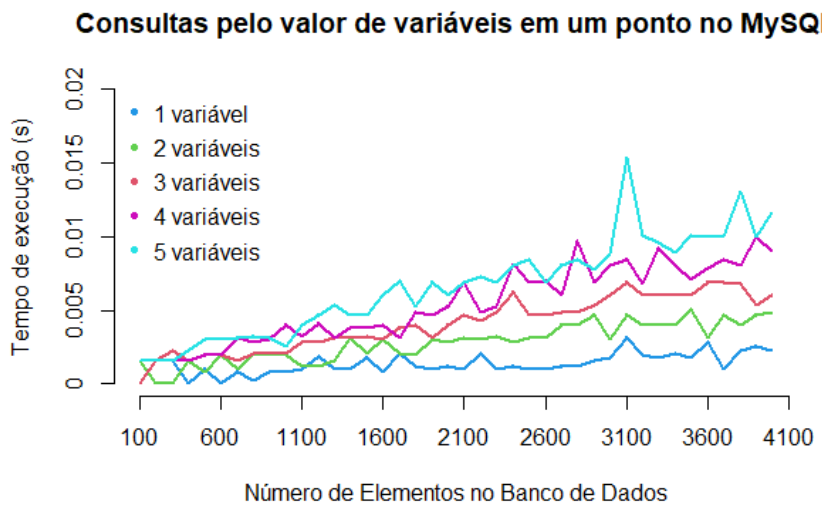
Fonte: Autora (2020)

Figura 45 – Consulta de valores em um raio *r* no banco de dados Neo4J



Fonte: Autora (2020)

Figura 46 – Consulta de valores de variáveis em um ponto específico no MySQL

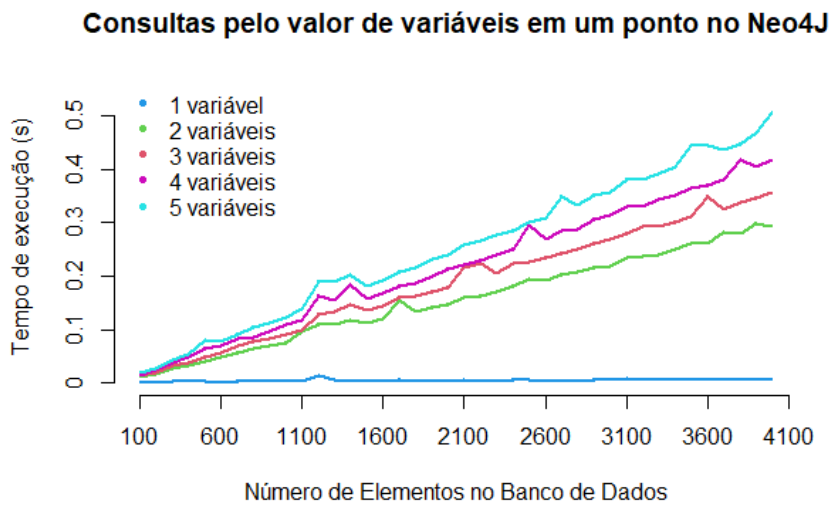


Fonte: Autora (2020)

variável, e para 2, 3, 4 e 5 variáveis as curvas são lineares. Já no caso do banco de dados de grafos as características são muito parecidas, porém é possível notar uma inclinação maior nas curvas para 2, 3, 4 e 5 variáveis.

Com base nos resultados observados nos testes realizados, para o sistema que está sendo proposto, a performance do banco de dados de grafos apresenta melhores resultados, uma vez que a inserção dos dados é realizada apenas uma única vez, porém as consultas são muito mais frequentes. Consultas para um único ponto também não são frequentes, do tipo que possuem melhor performance no banco relacional, pela característica de sistemas produtivos agrícolas, a característica de uma região é importante, e não de apenas um único ponto.

Figura 47 – Consulta de valores de variáveis em um ponto específico no Neo4J



Fonte: Autora (2020)

6 CONCLUSÕES

6.1 Considerações finais

A agricultura de precisão se utiliza de equipamentos e ferramentas para a coleta e análise espacializada de variáveis meteorológicas, agronômicas e de solo que têm influência sobre a produção agrícola. Os valores dessas variáveis são coletados em locais e tempos diferentes. A descoberta de conhecimento sobre as relações existentes entre os valores das diferentes variáveis e suas relações no espaço e no tempo exige um ferramental capaz de armazenar esses dados espaço-temporais, manipulá-los e gerar relatórios que possam servir de base na tomada de decisão, ou servirem como entrada em outros sistemas. O objetivo principal deste trabalho foi construir uma plataforma para armazenamento de dados espaço-temporais sobre colheita e demais variáveis de produção agrícola, de forma a servir de fonte para desenvolvimento de relatórios para o produtor e de entrada para sistemas de inferência e de apoio à decisão sobre os dados armazenados.

Após um estudo sobre o armazenamento de dados espaço-temporais, e suas formas de representação, se buscou um modelo de armazenamento utilizando banco de dados de grafos, em que se representasse os três aspectos (tempo, localização e valor) em separado mas com as devidas relações entre eles representadas. Esta forma de representação não é usual na bibliografia, porém é de fácil implementação em bancos de dados de grafos. O modelo passou por um processo de verificação e validação, onde foram inseridos tanto dados oriundos de um talhão de produção localizado na Embrapa Pecuária Sul como de dados simulados. Os testes foram feitos com a execução de diferentes modelos de consultas, relacionadas com o uso mais provável da ferramenta. Os experimentos mostraram que essa forma de representação se adapta a dados com características espaço-temporais. De posse desse modelo de representação, se passou para a etapa de avaliação de desempenho do banco de dados.

A segunda etapa trabalhada foi a comparação de dois paradigmas de bancos de dados para o armazenamento de dados espaço-temporais. A comparação foi realizada por meio de inserção e recuperação de registros no banco de dados relacional MySQL, e no banco de dados de grafos Neo4J. Os testes realizados e apresentados na Seção 5.3 permitem concluir que o armazenamento dos dados na forma de grafos, com os aspectos de localização, data e variável separadamente, tem um melhor desempenho nas consultas, retornando os valores em tempo linear independente da complexidade das consultas,

enquanto que o banco de dados relacional degrada seu desempenho na medida quem que dados de muitas variáveis diferentes precisam ser agrupados. Essa melhora no desempenho não ocorre no processo de inserção dos dados, e na consulta de um ponto específico, porém, se justifica a utilização do banco de dados de grafos, uma vez que o processo de inserção ocorre uma única vez, e a consulta sobre esses dados é realizada muitas vezes. Além disso, a busca por um único ponto não é usual na produção agrícola, onde normalmente o interesse é em uma região.

A construção de uma ferramenta *web* para o armazenamento e manipulação dos dados foi a última etapa de desenvolvimento do trabalho. A ferramenta denominada AGROGRAPH foi desenvolvida para que o modelo de banco de dados criado pudesse ser utilizado por usuários sem conhecimentos de linguagens de consulta de bancos de dados, como produtores rurais ou pesquisadores das áreas das Ciências Agrárias. O sistema permite o armazenamento de qualquer variável de produção, com localização espacial na forma de ponto. As consultas podem ser realizadas de forma visual, com os pontos plotados em um mapa, ou na forma de relatório. O método de geoestatística denominado *krigagem*, utilizado para a interpolação de valores de variáveis com relacionamento espacial, que prediz os valores nos locais onde a medição não foi realizada, foi implementado na ferramenta. Dessa forma, não é necessário utilizar um SIG (Sistema de Informações Geográficas) para o cálculo e visualização do resultado. A utilização de um banco de dados não relacional permite que tipos distintos de dados possam ser inseridos sem a necessidade de uma remodelagem do banco de dados.

Um dos maiores desafios relacionados a dados espaço-temporais é a sua forma de visualização e compreensão. A interface criada para inserção e recuperação de dados é de fácil manuseio e entendimento, faz uso de ferramentas de planilhas eletrônicas já conhecidas pelos usuários e onde a usabilidade foi levada em consideração para a elaboração de cada processo. A definição de consultas em duas etapas foi necessária para garantir a usabilidade do sistema, onde os critérios de quais dados serão visualizados são definidos em um primeiro momento, e os critérios de seleção sobre estes dados são definidos em um segundo momento por meio de filtros.

Uma das principais contribuições deste trabalho são a comparação de paradigmas diferentes de bancos de dados para o armazenamento de dados espaço-temporais, mostrando que no momento de armazenar os dados a performance dos bancos relacionais é melhor, mas consultas o banco de dados de grafos possuí um desempenho superior. Além disso, grafos são utilizados para a predição de valores, o que permite que os dados

armazenados dessa forma sejam utilizados em sistemas que façam predição de produção agrícola. Outra contribuição é a construção de uma ferramenta web para o armazenamento de dados de produção agrícola, com uma interface simples, que permite a inserção e consulta de dados de forma automatizada e visual em mapas, sem a necessidade de utilização de um software SIG.

6.2 Trabalhos futuros

O objetivo deste trabalho foi investigar o desempenho de bancos de dados de grafos no armazenamento de informações agrícolas. Em virtude do esforço necessário para a construção de um software funcional e completo, a funcionalidade do sistema pode ser incrementada ao longo do tempo. Algumas possibilidades são as seguintes:

1. Fazer uso da API da EMBRAPA que possui funcionalidades de *login* para os já usuários de outros sistemas, permitindo que um produtor possa usar o sistema com seus dados de registro já realizados. Essa funcionalidade vai permitir que exista uma certa garantia de que os dados inseridos correspondem a valores reais.
2. Possibilidade de inserção de dados a partir de arquivos *shapefile*, para permitir a inserção de dados oriundos de mapas de colheita gerados por implementos agrícolas sem necessidade de pré-processamento para transformação em dados matriciais. Esse processo é custoso e frequentemente feito de forma manual e requer conhecimentos de operação de softwares SIG.
3. Possibilidade de definição de áreas poligonais com características específcias. Essa forma de clusterização da informação pode ser útil em diversas aplicações.
4. Construção de um módulo de dados que armazene informações sobre ações de manejo realizados na propriedade.

Algumas funcionalidades existentes podem ser aprimoradas, como a tradução de outros formatos de latitude e longitude para o formato decimal, permitir o registro de horário de coleta de dados, seleção de colunas específicas do arquivo CSV, entre outras.

REFERÊNCIAS

AEROSPYKE. **Aerospyke - Next Generation, NoSQL Data Platform**. 2020. [Online; accessed 25-Junio-2020]. Disponível em: https://www.aerospike.com/.

BERNARDI, A. C. C. et al. (Ed.). **Agricultura de precisão: resultados de um novo olhar**. São Carlos: Embrapa Instrumentação, 2014.

BLANCO, R.; TUYA, J. A test model for graph database applications: An MDA-based approach. In: **Proceedings of the 6th International Workshop on Automating Test Case Design, Selection and Evaluation**. New York: Association for Computing Machinery, 2015. (A-TEST 2015), p. 8–15. ISBN 9781450338134. Disponível em: https://doi.org/10.1145/2804322.2804324.

Bootstrap Team. **Introduction Bootstrap v4.5**. 2020. [Online; accessed 25-July-2020]. Disponível em: https://getbootstrap.com/docs/4.5/getting-started/introduction/.

CASANOVA, M. A. et al. **Banco de dados geográficos**. Curitiba: MundoGEO, 2005.

CATTELL, R. Scalable sql and nosql data stores. **SIGMOD Rec.**, Association for Computing Machinery, New York, NY, USA, v. 39, n. 4, p. 12–27, maio 2011. ISSN 0163-5808. Disponível em: https://doi.org/10.1145/1978915.1978919.

COELHO, A. M. **Agricultura de precisão: manejo da variabilidade espacial e temporal dos solos e culturas**. Sete Lagoas: Embrapa Milho e Sorgo, 2005.

COMEX Stat - ComexVis. 2020. [Online; accessed 21-Junio-2020]. Disponível em: http://comexstat.mdic.gov.br/pt/comex-vis.

ELMASRI, R.; NAVATHE, S. B. **Sistemas de Banco de Dados**. São Paulo: Pearson Addison Wesley, 2011.

FERREIRA, A. P. L. **Object-oriented graph grammars**. Tese (Doutorado) — Universidade Federal do Rio Grande do Sul, Porto Alegre, 2005.

FERREIRA, J. S. A. **PREDIÇÃO DA VARIABILIDADE ESPACIAL DA PRODUTIVIDADE AGRÍCOLA COM MODELOS OCULTOS DE MARKOV**. Dissertação (Mestrado) — Programa de Pós-graduação em Computação Aplicada - Universidade Federal do Pampa, 2019.

FISS, R. E.; FERREIRA, A. P. L.; PEREZ, N. B. AnÁlise de consultas SQL e Cypher em dados de produÇÃo agrÍcola. In: **Anais da 7ª Conferência Ibero-americana de Computação Aplicaca (CIACA 2020)**. Lisboa: [s.n.], 2020. p. (to appear).

FOWLER, A. **NoSQL for Dummies**. Hoboken: John Wiley & Sons Inc, 2015.

FUNDATION, A. S. **Apache Cassandra**. 2020. [Online; accessed 25-Junio-2020]. Disponível em: https://cassandra.apache.org/.

FUNDATION, A. S. **Apache CouchDB**. 2020. [Online; accessed 25-Junio-2020]. Disponível em: https://couchdb.apache.org/.

FUNDATION, A. S. **Apache Hadoop**. 2020. [Online; accessed 25-Junio-2020]. Disponível em: http://hadoop.apache.org/.

FUNDATION, A. S. **Open Source Serach & Analytics - Elasticsearch**. 2020. [Online; accessed 25-Junio-2020]. Disponível em: https://www.elastic.co/pt/.

GITHUB. **GraphAware Neo4j PHP Client**. 2020. [Online; accessed 25-Nov-2020]. Disponível em: https://github.com/graphaware/neo4j-php-client.

GOOGLE. **Cloud BigTable**. 2020. [Online; accessed 25-July-2020]. Disponível em: https://cloud.google.com/bigtable.

Google Developers. **Plataforma do Google Maps**. 2020. [Online; accessed 14-September-2020]. Disponível em: https://developers.google.com/maps/documentation?hl=pt-br.

GRAPHWARE. **GitHub – graphware/neo4j-php-client**. 2019. [Online; accessed 20-July-2020]. Disponível em: https://github.com/graphaware/neo4j-php-client.

HAN, J. et al. Survey on nosql database. In: **2011 6th International Conference on Pervasive Computing and Applications**. [S.l.: s.n.], 2011. p. 363–366.

HECHT, R.; JABLONSKI, S. Nosql evaluation: A use case oriented survey. In: IEEE. **2011 International Conference on Cloud and Service Computing**. [S.l.], 2011. p. 336–341.

HEUSER, C. A. **Projeto de Banco de Dados**. Porto Alegre: Sagra Luzzato, 2008.

KRIGE, D. G. A statistical approach to some basic mine valuation problems on the witwatersrand. **Journal of the Southern African Institute of Mining and Metallurgy**, Southern African Institute of Mining and Metallurgy, v. 52, n. 6, p. 119–139, 1951.

LANGRAN, G.; CHRISMAN, N. R. A framework for temporal geographic information. **Cartographica: The International Journal for Geographic Information and Geovisualization**, University of Toronto Press, v. 25, n. 3, p. 1–14, 1988.

LIAN, J. et al. A gis-based spatial management and analysis system for rural socioeconomic statistic data. In: **2010 IEEE International Geoscience and Remote Sensing Symposium**. [S.l.: s.n.], 2010. p. 1691–1694. ISSN 2153-7003.

LÓSCIO, B. F.; OLIVEIRA, H. d.; PONTES, J. d. S. Nosql no desenvolvimento de aplicações web colaborativas. **VIII Simpósio Brasileiro de Sistemas Colaborativos**, sn, v. 10, n. 1, p. 11, 2011.

Lucid Software Inc. **Software online de diagramas e comunicação visual**. 2020. [Online; accessed 18-November-2020]. Disponível em: https://www.lucidchart.com.

MANUAL de Referência do MySQL 4.1. 2020. [Online; accessed 21-Junio-2020]. Disponível em: https://downloads.mysql.com/docs/refman-4.1-pt.pdf.

MAPA. **Agricultura de Precisão**. Brasília, DF, 2011. 36 p.

MAPR. **Industryś Next Generation Data Plataform for AI and Analytics**. 2020. [Online; accessed 25-Junio-2020]. Disponível em: https://mapr.com/.

MATHERON, G. Principles of geostatistics. **Economic Geology**, v. 58, n. 8, p. 1246–1266, 12 1963. ISSN 0361-0128. Disponível em: https://doi.org/10.2113/gsecongeo.58.8.1246.

MEIER, A.; KAUFMANN, M. **SQL & NoSQL Databases: Models, Languages, Consistency Options and Architectures for Big Data Management**. Wiesbaden: Spring Viewg, 2019.

MICROSOFT. **Documentação do Azure**. 2020. [Online; accessed 25-Junio-2020]. Disponível em: https://docs.microsoft.com/pt-br/azure/?product=featured.

MINES ParisTech / ARMINES. **RGeostats: The Geostatistical R Package**. Fontainebleau, France, 2020. Free download from: <http://cg.ensmp.fr/rgeostats>.

MONGODB. **The most popular database for moderns app**. 2020. [Online; accessed 25-Junio-2020]. Disponível em: https://www.mongodb.com/.

Mozilla. **Referência JavaScript**. 2020. [Online; accessed 25-July-2020]. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference.

NARCISO, M. G. et al. Banco de dados georeferenciado e sistema de apoio a decisão para a cultura do milho. In: IN: CONGRESSO BRASILEIRO DE AGROINFORMÁTICA, 5.; SIMPÓSIO BRASILEIRO DE .... **Embrapa Informática Agropecuária-Artigo em anais de congresso (ALICE)**. [S.l.], 2005.

Neo4J. **The Neo4J Cypher Manual v4.1**. 2020. [Online; accessed 20-July-2020]. Disponível em: https://neo4j.com/docs/cypher-manual/current/.

NEO4J Documentation. 2020. [Online; accessed 21-Junio-2020]. Disponível em: https://neo4j.com/docs/.

NEO4J Spatial. 2020. [Online; accessed 21-Junio-2020]. Disponível em: https://neo4j-contrib.github.io/spatial/.

NOSQL. 2020. [Online; accessed 25-Junio-2020]. Disponível em: https://hostingdata.co.uk/nosql-database/.

PELEKIS, N. et al. Literature review of spatio-temporal database models. **The Knowledge Engineering Review**, Cambridge University Press, v. 19, n. 3, p. 235–274, 2004.

POSTGIS. 2020. [Online; accessed 20-July-2020]. Disponível em: http://www.postgis.org/.

POSTGRESQL. **PostgreSQL: The World's most advanced open source database**. 2020. [Online; accessed 20-July-2020]. Disponível em: https://www.postgresql.org/.

PYTHON Tutorial. 2020. [Online; accessed 21-Junio-2020]. Disponível em: https://docs.python.org/3/tutorial/index.html.

R Core Team. **R: A Language and Environment for Statistical Computing**. Vienna, Austria, 2013. Disponível em: http://www.R-project.org/.

REDIS. **Redis**. 2020. [Online; accessed 25-Junio-2020]. Disponível em: https://redis.io/.

RIAK. **Entreprise NoSQL Database**. 2020. [Online; accessed 25-Junio-2020]. Disponível em: https://riak.com/index.html.

ROBINSON, I.; WEBBER, J.; EIFREM, E. **Graph Database**: New oportunities for connected data. Sebastopol: OŔeilly, 2015.

ROCHA, D. S. da et al. Aplicação de um sistema webgis na agricultura de precisão. **Ciência e Natura**, Universidade Federal de Santa Maria, v. 37, n. 3, p. 262–273, 2015.

ROCKENBACH, D. et al. Estudo comparativo de bancos de dados nosql. **Revista Eletrônica Argentina-Brasil de Tecnologias da Informação e da Comunicação**, v. 1, n. 8, 2018. ISSN 2446-7634. Disponível em: https://revistas.setrem.com.br/index.php/reabtic/article/view/286.

SADALAGE, P. J.; FOWLER, M. **NoSQL Essencial**: Um guia conciso para o mundo emergente da persistência poliglota. São Paulo: Novatec, 2013.

SCHÄFER, A. G. **Um banco de dados espaço-temporal para o monitoramento e modelagem do escoamento superficial em bacias hidrográficas no contexto do planejamento urbano**. Tese (Doutorado) — Universidade Federal de Santa Catarina, 2012.

SERVICES, A. W. **Amazon DynamoDB**. 2020. [Online; accessed 25-Junio-2020]. Disponível em: https://aws.amazon.com/pt/dynamodb/.

SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. **Sistema de Banco de Dados.** Rio de Janeiro: Elsevier, 2006.

SOARES, A. Geoestatística para as ciências da terra e do ambiente. 2ª edição. **Colecção Ensino da Ciência e Tecnologia. Lisboa: Instituto Superior Técnico**, 2006.

SRIVASTAVA, R. Describing spatial variability using geostatistical analysis. In: **Geostatistics for Environmental and Geotechnical Applications**. ASTM International, 1996. p. 13–13–7. Disponível em: https://doi.org/10.1520/stp16110s.

SULLIVAN, D. **NoSQl for Mere Mortals**. Michigan: Pearson Education, 2015.

TEIXEIRA, M. B. dos R.; SCALON, J. D. Comparac ao entre estimadores de semivari^ancia. **Rev. Bras. Biom**, v. 31, n. 2, p. 248–269, 2013.

The PHP Group. **PHP: Manual do PHP**. 2020. [Online; accessed 20-July-2020]. Disponível em: https://www.php.net/manual/pt_BR/index.php.

TRANGMAR, B. B.; YOST, R. S.; UEHARA, G. Application of geostatistics to spatial studies of soil properties. In: **Advances in agronomy**. [S.l.]: Elsevier, 1986. v. 38, p. 45–94.

WALID, F.; EZZEDINE, T. Design of a climate monitoring system based on sensor network. In: **2017 13th International Wireless Communications and Mobile Computing Conference (IWCMC)**. [S.l.: s.n.], 2017. p. 1791–1796. ISSN 2376-6506.

YUAN, M. Wildfire conceptual modeling for building gis space-time models. In: **proceedings of GIS/LIS**. [S.l.: s.n.], 1994. v. 94, p. 860–869.

APÊNDICE A – ALGORITMO DE KRIGAGEM EM R

```
library (RGeostats)

## Recebimento de parametros
args <- commandArgs(TRUE)
tx <- args[1]
ty <- args[2]
tv <- args[3]
ncel <- as.integer(args[4])
nlag <- as.integer(args[5])
ndir <- as.integer(args[6])
ang <- as.integer(args[7])

## transformar string recebida em vetores de numeros x, y e v
x<-str_split(tx,',')
y<-str_split(ty,',')
v<-str_split(tv,',')


for (i in 1:length(x)){
  x[[i]]<-as.double(x[[i]])
}

for (i in 1:length(y)){
  y[[i]]<-as.double(y[[i]])
}

for (i in 1:length(v)){
  v[[i]]<-as.double(v[[i]])
}

#Criacao db com os dados
rx <- extendrange(x)
ry <- extendrange(y)
db.data <- db.create(x=x,y=y,v=v)

#Criacao grid que ir receber os dados
extx <- rx[2] - rx[1]
exty <- ry[2] - ry[1]
diam <- sqrt(extx * extx + exty * exty)
```

```
db.grid <- db.create(flag.grid=TRUE,x0=c(rx[1],ry[1]),
                     nx=c(ncell,ncell),dx=c(extx/ncell,exty/ncell))

#Criacao da vizinhanca
test.neigh <- neigh.create(ndim=2,type=0)

dirvect <- (seq(1,ndir)-1) * 180 / ndir + ang

# Calcula o variograma experimental e o ajuste automtico
test.vario <- vario.calc(db.data, dirvect=dirvect, lag=diam/(2*nlag), nlag=nlag)
test.model <- model.auto(test.vario,draw=FALSE)

# Executa o Kriging
db.grid <- kriging(db.data,db.grid, test.model, test.neigh)
```

APÊNDICE B – FUNÇÕES EM PYTHON PARA INSERÇÃO DE DADOS NO MYSQL E NEO4J

```
def insert_mysql (pasta, arq):
    db = conect_database('bd3')
    mycursor = db.cursor()

    c = csv.writer(open(pasta + '\\' + arq + "_tempoSQL.csv", "w",
    newline=''), delimiter=',')

    i = 0

    with open(pasta + '\\' + arq + ".csv", "r") as file:
        reader = csv.reader(file)

        for row in reader:

            row[0] = float(row[0])
            row[1] = float(row[1])
            row[3] = float(row[3])

            insert = "insert into " + arq + " (location, time, value)
            VALUES (ST_geomfromtext('point(%s %s)', 4326), %s, %s)"
            values = [row[0], row[1], row[2], row[3]]

            ini = datetime.datetime.now()

            mycursor.execute(insert, values)
            db.commit()

            fim = datetime.datetime.now()

            c.writerow([i] + [(fim - ini).total_seconds()])

def insert_neo4j (pasta, arq):
    db = functions_neo4j.conecta_neo4j()

    c = csv.writer(open(pasta + '\\' + arq + "_tempo_neo4j.csv", "w",
    newline=''), delimiter=',')

```

```
    i = 0
39  #abre arquivo csv
    with open(pasta + '\\' + arq + ".csv", "r") as file :
41      reader = csv.reader( file )

43      for row in reader:
            #valores para as variaveis
45          lat = row[0]
            lon = row[1]
47          val = row[3]

49          ini = datetime.datetime.now()

51          #1 no location
            busca = functions_neo4j.busca_no_location(db, lat,lon)
53
            if len(busca) == 0: #vazio
55              location = functions_neo4j.cria_no_location(db,lat,lon)
                functions_neo4j.addLayer(db,lat,lon)
57          else:
                temp = busca[0]
59              location = temp.pop(0)

61          #2 no value
            busca2 = functions_neo4j.busca_no_value(db, arq, val)
63
            if len(busca2) == 0:
65              value = functions_neo4j.cria_no_value(db, arq, val)
            else:
67              temp2 = busca2[0]
                value = temp2.pop(0)
69
            #3 cria aresta
71          location.date(value, date='2020-02-20')

73          fim = datetime.datetime.now()

75          i += 1

77          c.writerow([i] + [(fim - ini).total_seconds()])
```

APÊNDICE C – FUNÇÕES EM PYTHON PARA CONSULTA DE TODOS OS ELEMENTOS

```
def consulta_todos_sql(pasta):

    arq = csv.writer(open('teste3/' + pasta+ "/consultatodos.csv", "w",
    newline=''), delimiter=',')

    db = cons_mysql.conect_database('bd3')
    c = db.cursor()

    #1 var
    i=0
    while i<10:
        i+=1
        ini = datetime.datetime.now()

        c.execute("Select * from K")
        res = c.fetchall()

        fim = datetime.datetime.now()

        arq.writerow(['1VAR'] + [(fim - ini).total_seconds()])

    #2 var
    i=0
    while i<10:
        i+=1
        ini = datetime.datetime.now()

        c.execute("select * from k left join p "\
                  "on k.location = p.location " \
                  "union all " \
                  "select * from k right join p "\
                  "on k.location = p.location " \
                  "where k.value IS NULL")
        res = c.fetchall()
        fim = datetime.datetime.now()

        arq.writerow(['2VAR'] + [(fim - ini).total_seconds()])
```

```
#3 var
i=0
while i<10:
    i+=1
    ini = datetime.datetime.now()

    c.execute("SELECT * "\
              "FROM K LEFT JOIN P on k.location = p.location "\
                  "LEFT JOIN OM on p.location = OM.location "\
              "union all "\
              "SELECT * "\
              "FROM K RIGHT JOIN P on k.location = p.location "\
                  "LEFT JOIN OM on p.location = OM.location "\
              "where k.value IS NULL "\
              "union all "\
              "SELECT * "\
              "FROM K RIGHT JOIN P on k.location = p.location "\
                  "RIGHT JOIN OM on K.location = OM.location "\
              "where k.value is null and p.value is null")
    res = c.fetchall()

    fim = datetime.datetime.now()

    arq.writerow(['3VAR'] + [(fim - ini).total_seconds()])


#4 var
i=0
while i<10:
    i+=1

    ini = datetime.datetime.now()

    c.execute("SELECT * "\
              "FROM K LEFT JOIN P on k.location = p.location "\
                  "LEFT JOIN OM on p.location = OM.location "\
                      "LEFT JOIN PH on OM.location = ph.location "\
              "union all "\
              "SELECT * "\
              "FROM K RIGHT JOIN P on k.location = p.location "\
```

```
79                      "LEFT JOIN OM on p.location = OM.location "\
                          "LEFT JOIN PH on OM.location = ph.location  "\
81                  "WHERE K.value IS NULL "\
                    "union  all  "\
83                  "SELECT * "\
                    "FROM K RIGHT JOIN P on k.location = p.location  "\
85                      "RIGHT JOIN OM on p.location = OM.location "\
                          "LEFT JOIN PH on OM.location = ph.location  "\
87                  "WHERE K.value IS NULL AND P.value IS NULL "\
                    "union  all  "\
89                  "SELECT * "\
                    "FROM K RIGHT JOIN P on k.location = p.location  "\
91                      "RIGHT JOIN OM on p.location = OM.location "\
                          "RIGHT JOIN PH on OM.location = ph.location "\
93                  "WHERE K.value IS NULL AND P.value IS NULL "\
                        "AND om.value is null")
95      res = c.fetchall()

97      fim = datetime.datetime.now()

99      arq.writerow(['4VAR'] + [(fim - ini).total_seconds()])

101 #5 var
    i=0
103 while i<10:
        i+=1
105     ini = datetime.datetime.now()

107     c.execute("SELECT * "\
                  "FROM K LEFT JOIN P on k.location = p.location  "\
109             "LEFT JOIN OM on p.location = OM.location "\
                  "LEFT JOIN PH on OM.location = ph.location  "\
111               "LEFT JOIN NDVI on ph.location = ndvi. location  "\
                  "UNION ALL "\
113               "SELECT *  "\
                  "FROM K RIGHT JOIN P on k.location = p.location  "\
115             "LEFT JOIN OM on p.location = OM.location "\
                  "LEFT JOIN PH on OM.location = ph.location  "\
117               "LEFT JOIN NDVI on ph.location = ndvi. location  "\
                  "WHERE K.value IS NULL "\
119               "UNION ALL "\
```

```
                    "SELECT * "\
                    "FROM K RIGHT JOIN P on k.location = p.location "\
                  "RIGHT JOIN OM on p.location = OM.location "\
                    "LEFT JOIN PH on OM.location = ph.location "\
                    "LEFT JOIN NDVI on ph.location = ndvi. location "\
                    "WHERE K.value IS NULL AND P.value IS NULL "\
                    "UNION ALL "\
                    "SELECT * "\
                    "FROM K RIGHT JOIN P on k.location = p.location "\
                  "RIGHT JOIN OM on p.location = OM.location "\
                    "RIGHT JOIN PH on OM.location = ph.location "\
                    "LEFT JOIN NDVI on ph.location = ndvi. location "\
                    "WHERE K.value IS NULL AND P.value IS NULL "\
                    "AND om.value is null "\
                    "UNION ALL "\
                    "SELECT * "\
                    "FROM K RIGHT JOIN P on k.location = p.location "\
                  "RIGHT JOIN OM on p.location = OM.location "\
                    "RIGHT JOIN PH on OM.location = ph.location "\
                    "RIGHT JOIN NDVI on ph.location = ndvi.location "\
                    "WHERE K.value IS NULL AND P.value IS NULL "\
                    "AND om.value is null AND ph.value is null ;")
        res = c.fetchall()

        fim = datetime.datetime.now()

        arq.writerow(['5VAR'] + [(fim - ini).total_seconds()])


def consulta_todos_neo(pasta):
    arq = csv.writer(open('teste3/' + pasta + "/consultatodosneo.csv",
    "w", newline=''), delimiter=',')

    db = functions_neo4j.conecta_neo4j()

    # 1 var
    i = 0
    while i < 10:
        i += 1

        query = "MATCH (pontos:location)" \
```

```
161                 "MATCH (pontos)-[:date]->(val:k) " \
                    "RETURN pontos.latitude, pontos.longitude, val.value"
163
            ini = datetime.datetime.now()
165         resul = db.query(query, returns=(str, str, str))
            fim = datetime.datetime.now()
167
            arq.writerow(['1VAR'] + [(fim - ini).total_seconds()])
169
        # 2 var
171     i = 0
        while i < 10:
173         i += 1

175         query = "MATCH (pontos:location) " \
                    "OPTIONAL MATCH (pontos)-[:date]->(k:k) " \
177                 "OPTIONAL MATCH (pontos)-[:date]->(p:p) " \
                    "RETURN pontos.latitude, pontos.longitude, "\
179                 "k.value, p.value"

181         ini = datetime.datetime.now()
            resul = db.query(query, returns=(str, str, str, str))
183         fim = datetime.datetime.now()

185         arq.writerow(['2VAR'] + [(fim - ini).total_seconds()])

187     # 3 var
        i = 0
189     while i < 10:
            i += 1
191
            query = "MATCH (pontos:location)" \
193                 "OPTIONAL MATCH (pontos)-[:date]->(k:k)" \
                    "OPTIONAL MATCH (pontos)-[:date]->(p:p)" \
195                 "OPTIONAL MATCH (pontos)-[:date]->(om:om)" \
                    "RETURN pontos.latitude, pontos.longitude, "\
197                 "k.value, p.value, om.value"

199         ini = datetime.datetime.now()
            resul = db.query(query, returns=(str, str, str, str, str))
201         fim = datetime.datetime.now()
```

```
203         arq.writerow(['3VAR'] + [(fim - ini).total_seconds()])

205     # 4 var
        i = 0
207     while i < 10:
            i += 1
209
            query = "MATCH (pontos:location)" \
211                     "OPTIONAL MATCH (pontos)-[:date]->(k:k)" \
                        "OPTIONAL MATCH (pontos)-[:date]->(p:p)" \
213                     "OPTIONAL MATCH (pontos)-[:date]->(om:om)" \
                        "OPTIONAL MATCH (pontos)-[:date]->(ph:ph)" \
215                     "RETURN pontos.latitude, pontos.longitude, "\
                        "k.value, p.value, om.value, ph.value"
217
            ini = datetime.datetime.now()
219         resul = db.query(query, returns=(str, str, str, str, str, str))
            fim = datetime.datetime.now()
221
            arq.writerow(['4VAR'] + [(fim - ini).total_seconds()])
223
        # 5 var
225     i = 0
        while i < 10:
227         i += 1

229         query = "MATCH (pontos:location)" \
                        "OPTIONAL MATCH (pontos)-[:date]->(k:k)" \
231                     "OPTIONAL MATCH (pontos)-[:date]->(p:p)" \
                        "OPTIONAL MATCH (pontos)-[:date]->(om:om)" \
233                     "OPTIONAL MATCH (pontos)-[:date]->(ph:ph)" \
                        "OPTIONAL MATCH (pontos)-[:date]->(ndvi:ndvi)" \
235                     "RETURN pontos.latitude, pontos.longitude, "\
                        "k.value, p.value, om.value, ph.value, ndvi.value"
237
            ini = datetime.datetime.now()
239         resul = db.query(query,
            returns=(str, str, str, str, str, str, str))
241         fim = datetime.datetime.now()
```

```
243             arq.writerow(['5VAR'] + [(fim - ini).total_seconds()])
```

APÊNDICE D – FUNÇÕES EM PYTHON PARA CONSULTA DE VARIÁVEIS PELA DISTÂNCIA DE PONTO CENTRAL

```
def consulta_distancia_sql (pasta):

    db = cons_mysql.conect_database('bd3')
    c = db.cursor()

    ###1 var
    arq = csv.writer(open('teste3/' + pasta + "/distancia_1var.csv",
    "w", newline=''), delimiter=',')

    dist = 0

    while dist < 100001:
        ini = datetime.datetime.now()
        c.execute("Select * from K "\
                  "WHERE st_distance_sphere(k.location, "\
                  "ST_geomfromtext('point(-53 -31)', 4326)) <="
                  + str(dist))
        res = c.fetchall()

        fim = datetime.datetime.now()
        arq.writerow([dist] + [(fim-ini).total_seconds()])
        dist += 500


    ###2 var
    arq = csv.writer(open('teste3/' + pasta + "/distancia_2var.csv",
    "w", newline=''), delimiter=',')

    dist = 0

    while dist < 100001:
        ini = datetime.datetime.now()

        c.execute("select * from k "\
                  "left join p on k.location = p.location " \
                  "where st_distance_sphere(k.location, "\
                  "ST_geomfromtext('point(-53 -31)', 4326)) <= "
```

```
38                  + str(dist) +
                    " union all " \
40                  " select * from k "\
                    " right join p on k.location = p.location " \
42                  "where k.value IS NULL AND st_distance_sphere( "\
                    "p.location, "\
44                  "ST_geomfromtext('point(-53 -31)', 4326)) <= "
                    + str(dist))
46          res = c.fetchall()
            fim = datetime.datetime.now()
48          arq.writerow([dist] + [(fim - ini).total_seconds()])
            dist += 500
50
    ###3 var
52  arq = csv.writer(open('teste3/' + pasta + "/distancia_3var.csv",
    "w", newline=''), delimiter=',')
54
    dist = 0
56
    while dist < 100001:
58          ini = datetime.datetime.now()

60          c.execute("SELECT * "\
                    "FROM K LEFT JOIN P on k.location = p.location "\
62                      "LEFT JOIN OM on p.location = OM.location "\
                    "WHERE st_distance_sphere(k.location,ST_geomfromtext"\
64                  "('point(-53 -31)', 4326)) <= " + str(dist) +
                    " union all "\
66                  "SELECT * "\
                    "FROM K RIGHT JOIN P on k.location = p.location "\
68                      "LEFT JOIN OM on p.location = OM.location "\
                    "where k.value IS NULL AND "\
70                  "st_distance_sphere(p.location, ST_geomfromtext"\
                    "('point(-53 -31)', 4326)) <= " + str(dist) +
72                  " union all "\
                    "SELECT * "\
74                  "FROM K RIGHT JOIN P on k.location = p.location "\
                        "RIGHT JOIN OM on K.location = OM.location "\
76                  "where k.value is null and p.value is null AND"\
                    "st_distance_sphere(om.location, ST_geomfromtext"\
78                  "('point(-53 -31)', 4326)) <= " + str(dist))
```

```
        res = c.fetchall()
        fim = datetime.datetime.now()
        arq.writerow([dist] + [(fim - ini).total_seconds()])
        dist += 500

###4 var
arq = csv.writer(open('teste3/' + pasta + "/distancia_4var.csv",
"w", newline=''), delimiter=',')

dist = 0

while dist < 100001:
    ini = datetime.datetime.now()

    c.execute("SELECT * "\
                "FROM K LEFT JOIN P on k.location = p.location "\
                    "LEFT JOIN OM on p.location = OM.location "\
                        "LEFT JOIN PH on OM.location = ph.location "\
                "where st_distance_sphere(k.location, "\
                "ST_geomfromtext('point(-53 -31)', 4326)) <= "
                + str(dist) +
                " union all "\
                "SELECT * "\
                "FROM K RIGHT JOIN P on k.location = p.location "\
                    "LEFT JOIN OM on p.location = OM.location "\
                        "LEFT JOIN PH on OM.location = ph.location "\
                "WHERE K.value IS NULL AND st_distance_sphere"\
                "(p.location, ST_geomfromtext('point(-53 -31)', "\
                "4326)) <= " + str(dist) +
                " union all "\
                "SELECT * "\
                "FROM K RIGHT JOIN P on k.location = p.location "\
                    "RIGHT JOIN OM on p.location = OM.location "\
                        "LEFT JOIN PH on OM.location = ph.location "\
                "WHERE K.value IS NULL AND P.value IS NULL AND "\
                " st_distance_sphere(om.location, ST_geomfromtext"\
                "('point(-53 -31)', 4326)) <= " + str(dist) +
                " union all "\
                "SELECT * "\
                "FROM K RIGHT JOIN P on k.location = p.location "\
                    "RIGHT JOIN OM on p.location = OM.location "\
```

```
                    "RIGHT JOIN PH on OM.location = ph.location "\
                "WHERE K.value IS NULL AND P.value IS NULL AND "
                "om.value is null AND st_distance_sphere"\
                "(ph.location, ST_geomfromtext('point(-53 -31)', "\
                "4326)) <= " + str(dist))
    res = c.fetchall()
    fim = datetime.datetime.now()
    arq.writerow([dist] + [(fim - ini).total_seconds()])
    dist += 500

###5 var
arq = csv.writer(open('teste3/' + pasta + "/distancia_5var.csv",
"w", newline=''), delimiter=',')

dist = 0

while dist < 100001:
    ini = datetime.datetime.now()

    c.execute("SELECT * "\
                "FROM K LEFT JOIN P on k.location = p.location "\
            "LEFT JOIN OM on p.location = OM.location "\
                "LEFT JOIN PH on OM.location = ph.location "\
                "LEFT JOIN NDVI on ph.location = ndvi.location "\
                "where st_distance_sphere(k.location, "\
                "ST_geomfromtext('point(-53 -31)', 4326)) <= "
                + str(dist) +
                " UNION ALL "\
                "SELECT * "\
                "FROM K RIGHT JOIN P on k.location = p.location "\
            "LEFT JOIN OM on p.location = OM.location "\
                "LEFT JOIN PH on OM.location = ph.location "\
                "LEFT JOIN NDVI on ph.location = ndvi.location "\
                "WHERE K.value IS NULL AND st_distance_sphere"\
                "(p.location, ST_geomfromtext('point(-53 -31)', "\
                "4326)) <= " + str(dist) +
                " UNION ALL "\
                "SELECT * "\
                "FROM K RIGHT JOIN P on k.location = p.location "\
            "RIGHT JOIN OM on p.location = OM.location "\
                "LEFT JOIN PH on OM.location = ph.location "\
```

```
                    "LEFT JOIN NDVI on ph.location = ndvi.location  "\
                    "WHERE K.value IS NULL AND P.value IS NULL AND "\
                    " st_distance_sphere(om.location, ST_geomfromtext"\
                    "('point(-53 -31)', 4326)) <= " + str(dist) +
                    " UNION ALL "\
                    "SELECT * "\
                    "FROM K RIGHT JOIN P on k.location = p.location "\
                  "RIGHT JOIN OM on p.location = OM.location "\
                    "RIGHT JOIN PH on OM.location = ph.location "\
                    "LEFT JOIN NDVI on ph.location = ndvi.location  "\
                    "WHERE K.value IS NULL AND P.value IS NULL "\
                    "AND om.value is null AND st_distance_sphere"\
                    "(ph.location, ST_geomfromtext('point(-53 -31)', "\
                    "4326)) <= " + str(dist) +
                    " UNION ALL "\
                    "SELECT * "\
                    "FROM K RIGHT JOIN P on k.location = p.location "\
                  "RIGHT JOIN OM on p.location = OM.location "\
                    "RIGHT JOIN PH on OM.location = ph.location "\
                    "RIGHT JOIN NDVI on ph.location = ndvi.location  "\
                    "WHERE K.value IS NULL AND P.value IS NULL "\
                    "AND om.value is null AND ph.value is null "\
                    "AND st_distance_sphere(ndvi.location, "\
                    "ST_geomfromtext('point(-53 -31)', 4326)) "\
                    "<= " + str(dist))
        res = c.fetchall()
        fim = datetime.datetime.now()
        arq.writerow([dist] + [(fim - ini).total_seconds()])
        dist += 500


def consulta_distancia_neo4j(pasta):
    db = functions_neo4j.conecta_neo4j()

    #1var
    arq = csv.writer(open('teste3/' + pasta + "/neodistancia_1var.csv",
    "w", newline=''), delimiter=',')
    dist = 0

    while dist < 100001:
        d = str(dist / 1000)
```

```
202         query = "CALL spatial.withinDistance" \
                    "('layer', {latitude: -53, longitude: -31}, "+d+")"\
204                 "yield node AS pontos " \
                    "MATCH (pontos)-[:date]->(val:k) " \
206                 "RETURN pontos.latitude, pontos.longitude, val.value"

208         ini = datetime.datetime.now()
            resul = db.query(query, returns=(str, str, str))
210         fim = datetime.datetime.now()

212         arq.writerow([dist] + [(fim-ini).total_seconds()])
            dist += 500
214
    #2var
216 arq = csv.writer(open('teste3/' + pasta + "/neodistancia_2var.csv",
    "w", newline='\n'), delimiter=',')
218 dist = 0

220 while dist < 100001:
        d = str(dist / 1000)
222
        query = "CALL spatial.withinDistance" \
224                 "('layer', {latitude: -53, longitude: -31}, "+d+")"\
                    "yield node AS pontos " \
226                 "OPTIONAL MATCH (pontos)-[:date]->(k:k)" \
                    "OPTIONAL MATCH (pontos)-[:date]->(p:p)" \
228                 "RETURN pontos.latitude, pontos.longitude, k.value, "\
                    "p.value"
230
            ini = datetime.datetime.now()
232         resul = db.query(query, returns=(str, str, str, str))
            fim = datetime.datetime.now()
234
            arq.writerow([dist] + [(fim-ini).total_seconds()])
236         dist += 500

238 #3var
    arq = csv.writer(open('teste3/' + pasta + "/neodistancia_3var.csv",
240 "w", newline='\n'), delimiter=',')
    dist = 0
242
```

```
while dist < 100001:
    d = str(dist / 1000)

    query = "CALL spatial.withinDistance" \
            "('layer', {latitude: -53, longitude: -31}, "+d+")"\
            "yield node AS pontos " \
            "OPTIONAL MATCH (pontos)-[:date]->(k:k)" \
            "OPTIONAL MATCH (pontos)-[:date]->(p:p)" \
            "OPTIONAL MATCH (pontos)-[:date]->(om:om)" \
            "RETURN pontos.latitude, pontos.longitude, k.value, "\
            "p.value, om.value"

    ini = datetime.datetime.now()
    resul = db.query(query, returns=(str, str, str, str, str))
    fim = datetime.datetime.now()

    arq.writerow([dist] + [(fim-ini).total_seconds()])
    dist += 500

#4var
arq = csv.writer(open('teste3/' + pasta + "/neodistancia_4var.csv",
"w", newline=''), delimiter=',')
dist = 0

while dist < 100001:
    d = str(dist / 1000)

    query = "CALL spatial.withinDistance" \
            "('layer', {latitude: -53, longitude: -31}, "+d+")"\
            "yield node AS pontos " \
            "OPTIONAL MATCH (pontos)-[:date]->(k:k)" \
            "OPTIONAL MATCH (pontos)-[:date]->(p:p)" \
            "OPTIONAL MATCH (pontos)-[:date]->(om:om)" \
            "OPTIONAL MATCH (pontos)-[:date]->(ph:ph)" \
            "RETURN pontos.latitude, pontos.longitude, k.value,"\
            "p.value, om.value, ph.value"

    ini = datetime.datetime.now()
    resul = db.query(query, returns=(str, str, str, str, str, str))
    fim = datetime.datetime.now()
```

```
284         arq.writerow([ dist ] + [(fim-ini). total_seconds () ])
            dist += 500
286
    #5var
288 arq = csv. writer (open(' teste3 /' + pasta + "/ neodistancia_5var .csv",
    "w", newline='' ), delimiter =', ')
290 dist = 0

292 while dist < 100001:

294         d = str ( dist / 1000)

296         query = "CALL spatial. withinDistance " \
                    " (' layer ', { latitude : -53, longitude : -31}, "+ d + ")"\
298                 " yield node AS pontos " \
                    "OPTIONAL MATCH (pontos)-[:date]->(k:k)" \
300                 "OPTIONAL MATCH (pontos)-[:date]->(p:p)" \
                    "OPTIONAL MATCH (pontos)-[:date]->(om:om)" \
302                 "OPTIONAL MATCH (pontos)-[:date]->(ph:ph)" \
                    "OPTIONAL MATCH (pontos)-[:date]->(ndvi:ndvi)" \
304                 "RETURN pontos.latitude, pontos. longitude , k. value , "\
                    "p. value , om.value, ph. value , ndvi . value"
306
            ini = datetime . datetime . now()
308         resul = db.query(query, returns =
            ( str , str , str , str , str , str , str ))
310         fim = datetime . datetime . now()

312         arq.writerow([ dist ] + [(fim-ini). total_seconds () ])
            dist += 500
```

APÊNDICE E – FUNÇÕES EM PYTHON PARA CONSULTA DE VARIÁVEIS EM UM PONTO ESPECÍFICO

```
def consulta_ponto_mysql(pasta):

    arq = csv.writer(open('teste3/' + pasta+ "/consultaponto.csv",
    "w", newline=''), delimiter=',')

    db = cons_mysql.conect_database('bd3')
    c = db.cursor()

    #1 var
    i=0
    while i<10:
        i+=1

        ini = datetime.datetime.now()

        c.execute("Select * from K where location = "\
        "ST_geomfromtext('point(-53 -31)', 4326)")
        res = c.fetchall()

        fim = datetime.datetime.now()

        arq.writerow(['1VAR'] + [(fim - ini).total_seconds()])

    #2 var
    i=0
    while i<10:
        i+=1

        ini = datetime.datetime.now()

        c.execute("select * from k "\
                  "left join p on k.location = p.location " \
                  "where k.location = "\
                  "ST_geomfromtext('point(-53 -31)', 4326) "\
                  "union all " \
                  "select * from k "\
                  "right join p on k.location = p.location " \
```

```
                "where k.value  IS  NULL AND p.location = "\
                "ST_geomfromtext('point(  -53   -31)',  4326)")
        res  = c.  fetchall  ()

        fim  =  datetime.datetime.now()

        arq.writerow(['2VAR'] + [(fim  -  ini).  total_seconds ()])

    #3  var
    i=0
    while  i<10:
        i+=1

        ini  =  datetime.datetime.now()

        c.execute("SELECT * "\
                "FROM K LEFT JOIN P on k.location = p.location  "\
                    "LEFT JOIN OM on p.location = OM.location "\
                "where k. location  = "\
                "ST_geomfromtext('point(  -53   -31)',  4326)"
                "union  all  "\
                "SELECT *  "\
                "FROM K RIGHT JOIN P on k.location = p.location  "\
                    "LEFT JOIN OM on p.location = OM.location "\
                "where k.value  IS  NULL AND p.location = "\
                "ST_geomfromtext('point(  -53   -31)',  4326)"\
                "union  all  "\
                "SELECT * "\
                "FROM K RIGHT JOIN P on k.location = p.location  "\
                    "RIGHT JOIN OM on K.location = OM.location "\
                "where k.value  is  null  and p.value  is  null  "\
                "AND om.location = "\
                "ST_geomfromtext('point(  -53   -31)',  4326)")
        res  = c.  fetchall  ()

        fim  =  datetime.datetime.now()

        arq.writerow(['3VAR'] + [(fim  -  ini).  total_seconds ()])


    #4  var
```

```
i=0
while i<10:
    i+=1

    ini = datetime.datetime.now()

    c.execute("SELECT * "\
                "FROM K LEFT JOIN P on k.location = p.location "\
                    "LEFT JOIN OM on p.location = OM.location "\
                        "LEFT JOIN PH on OM.location = ph.location "\
                "where k.location = "\
                "ST_geomfromtext('point(-53 -31)', 4326) "\
                "union all "\
                "SELECT * "\
                "FROM K RIGHT JOIN P on k.location = p.location "\
                    "LEFT JOIN OM on p.location = OM.location "\
                        "LEFT JOIN PH on OM.location = ph.location "\
                "WHERE K.value IS NULL AND p.location = "\
                "ST_geomfromtext('point(-53 -31)', 4326)"\
                "union all "\
                "SELECT * "\
                "FROM K RIGHT JOIN P on k.location = p.location "\
                    "RIGHT JOIN OM on p.location = OM.location "\
                        "LEFT JOIN PH on OM.location = ph.location "\
                "WHERE K.value IS NULL AND P.value IS NULL "\
                "AND om.location = "\
                "ST_geomfromtext('point(-53 -31)', 4326)"\
                "union all "\
                "SELECT * "\
                "FROM K RIGHT JOIN P on k.location = p.location "\
                    "RIGHT JOIN OM on p.location = OM.location "\
                        "RIGHT JOIN PH on OM.location = ph.location "\
                "WHERE K.value IS NULL AND P.value IS NULL "\
                "AND om.value is null AND ph.location = "\
                "ST_geomfromtext('point(-53 -31)', 4326)")
    res = c.fetchall()

    fim = datetime.datetime.now()

    arq.writerow(['4VAR'] + [(fim - ini).total_seconds()])
```

```
120     #5 var
        i=0
122     while i<10:
            i+=1
124
            ini = datetime.datetime.now()
126
            c.execute("SELECT * "\
128                       "FROM K LEFT JOIN P on k.location = p.location "\
                      "LEFT JOIN OM on p.location = OM.location "\
130                       "LEFT JOIN PH on OM.location = ph.location "\
                          "LEFT JOIN NDVI on ph.location = ndvi.location "\
132                       "where k.location = "\
                          "ST_geomfromtext('point(-53 -31)', 4326) "\
134                       "UNION ALL "\
                          "SELECT * "\
136                       "FROM K RIGHT JOIN P on k.location = p.location "\
                      "LEFT JOIN OM on p.location = OM.location "\
138                       "LEFT JOIN PH on OM.location = ph.location "\
                          "LEFT JOIN NDVI on ph.location = ndvi.location "\
140                       "WHERE K.value IS NULL AND p.location = "\
                          "ST_geomfromtext('point(-53 -31)', 4326)"\
142                       "UNION ALL "\
                          "SELECT * "\
144                       "FROM K RIGHT JOIN P on k.location = p.location "\
                      "RIGHT JOIN OM on p.location = OM.location "\
146                       "LEFT JOIN PH on OM.location = ph.location "\
                          "LEFT JOIN NDVI on ph.location = ndvi.location "\
148                       "WHERE K.value IS NULL AND P.value IS NULL "\
                          "AND om.location = "\
150                       "ST_geomfromtext('point(-53 -31)', 4326)"\
                          "UNION ALL "\
152                       "SELECT * "\
                          "FROM K RIGHT JOIN P on k.location = p.location "\
154                   "RIGHT JOIN OM on p.location = OM.location "\
                          "RIGHT JOIN PH on OM.location = ph.location "\
156                       "LEFT JOIN NDVI on ph.location = ndvi.location "\
                          "WHERE K.value IS NULL AND P.value IS NULL "\
158                       "AND om.value is null AND ph.location = "\
                          "ST_geomfromtext('point(-53 -31)', 4326)"\
160                       "UNION ALL "\
```

```
                    "SELECT * "\
162                 "FROM K RIGHT JOIN P on k.location = p.location "\
                  "RIGHT JOIN OM on p.location = OM.location "\
164                 "RIGHT JOIN PH on OM.location = ph.location "\
                    "RIGHT JOIN NDVI on ph.location = ndvi.location "\
166                 "WHERE K.value IS NULL AND P.value IS NULL "\
                    "AND om.value is null AND ph.value is null "\
168                 "AND ndvi.location = "\
                    "ST_geomfromtext('point(-53 -31)', 4326)")
170         res = c.fetchall()

172         fim = datetime.datetime.now()

174         arq.writerow(['5VAR'] + [(fim - ini).total_seconds()])

176 def consulta_ponto_neo4j(pasta):

178     arq = csv.writer(open('teste3/' + pasta+ "/consultapontoneo.csv",
    "w", newline='\n'), delimiter=',')
180
    db = functions_neo4j.conecta_neo4j()
182
    # 1 var
184 i = 0
    while i < 10:
186         i += 1

188         query = "MATCH (pontos:location) " \
                "MATCH (pontos)-[:date]->(val:k) " \
190             "WHERE pontos.latitude = -53 AND pontos.longitude = -31 " \
                "RETURN pontos.latitude, pontos.longitude, val.value"
192
            ini = datetime.datetime.now()
194         resul = db.query(query, returns=(str, str, str))
            fim = datetime.datetime.now()
196
            arq.writerow(['1VAR'] + [(fim - ini).total_seconds()])
198
    # 2 var
200 i = 0
    while i < 10:
```

```
202         i += 1

204         query = "MATCH (pontos:location) " \
                "OPTIONAL MATCH (pontos)-[:date]->(k:k) " \
206             "OPTIONAL MATCH (pontos)-[:date]->(p:p) " \
                "WHERE pontos.latitude = -53 AND pontos.longitude = -31 " \
208             "RETURN pontos.latitude, pontos.longitude, k.value,p.value"

210         ini = datetime.datetime.now()
            resul = db.query(query, returns=(str, str, str, str))
212         fim = datetime.datetime.now()

214         arq.writerow(['2VAR'] + [(fim - ini).total_seconds()])

216     # 3 var
        i = 0
218     while i < 10:
            i += 1
220
            query = "MATCH (pontos:location) " \
222             "OPTIONAL MATCH (pontos)-[:date]->(k:k) " \
                "OPTIONAL MATCH (pontos)-[:date]->(p:p) " \
224             "OPTIONAL MATCH (pontos)-[:date]->(om:om) " \
                "WHERE pontos.latitude = -53 AND pontos.longitude = -31 " \
226             "RETURN pontos.latitude, pontos.longitude, k.value, "\
                "p.value, om.value"
228
            ini = datetime.datetime.now()
230         resul = db.query(query, returns=(str, str, str, str, str))
            fim = datetime.datetime.now()
232
            arq.writerow(['3VAR'] + [(fim - ini).total_seconds()])
234
        # 4 var
236     i = 0
        while i < 10:
238         i += 1

240         query = "MATCH (pontos:location) " \
                "OPTIONAL MATCH (pontos)-[:date]->(k:k) " \
242             "OPTIONAL MATCH (pontos)-[:date]->(p:p) " \
```

```
            "OPTIONAL MATCH (pontos)-[:date]->(om:om) " \
            "OPTIONAL MATCH (pontos)-[:date]->(ph:ph) " \
            "WHERE pontos.latitude = -53 AND pontos.longitude = -31 " \
            "RETURN pontos.latitude, pontos.longitude, k.value, "\
            "p.value, om.value, ph.value"

        ini = datetime.datetime.now()
        resul = db.query(query, returns=(str, str, str, str, str, str))
        fim = datetime.datetime.now()

        arq.writerow(['4VAR'] + [(fim - ini).total_seconds()])

    # 5 var
    i = 0
    while i < 10:
        i += 1

        query = "MATCH (pontos:location) " \
            "OPTIONAL MATCH (pontos)-[:date]->(k:k) " \
            "OPTIONAL MATCH (pontos)-[:date]->(p:p) " \
            "OPTIONAL MATCH (pontos)-[:date]->(om:om) " \
            "OPTIONAL MATCH (pontos)-[:date]->(ph:ph) " \
            "OPTIONAL MATCH (pontos)-[:date]->(ndvi:ndvi) " \
            "WHERE pontos.latitude = -53 AND pontos.longitude = -31 " \
            "RETURN pontos.latitude, pontos.longitude, k.value, "\
            "p.value, om.value, ph.value, ndvi.value"

        ini = datetime.datetime.now()
        resul = db.query(query, returns=
        (str, str, str, str, str, str, str))
        fim = datetime.datetime.now()

        arq.writerow(['5VAR'] + [(fim - ini).total_seconds()])
```