UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Câmpus de São José do Rio Preto

INSTITUTO DE BIOCIÊNCIAS, LETRAS E CIÊNCIAS EXATAS
IBILCE / UNESP - CÂMPUS DE SÃO JOSÉ DO RIO PRETO

JOÃO ROBERTO MARQUES GUBOLIN

Ambiente baseado em contexto para identificação de tuplas duplicadas com recursos de paralelização e meta-heurísticas

São José do Rio Preto
2021

JOÃO ROBERTO MARQUES GUBOLIN

Ambiente baseado em contexto para identificação de tuplas duplicadas com recursos de paralelização e meta-heurísticas

Trabalho de Conclusão de Curso apresentado ao Departamento de Ciências de Computação e Estatística do Instituto de Biociências Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Financiador: Conselho Nacional de Desenvolvimento Científico e Tecnológico – CNPq.

Orientador: Prof. Dr. Carlos Roberto Valêncio

São José do Rio Preto
2021

G921a

Gubolin, João Roberto Marques

Ambiente baseado em contexto para identificação de tuplas duplicadas com recursos de paralelização e meta-heurísticas / João Roberto Marques Gubolin. -- São José do Rio Preto, 2022

43 f. : il., tabs.

Trabalho de conclusão de curso (Bacharelado - Ciência da Computação) - Universidade Estadual Paulista (Unesp), Instituto de Biociências Letras e Ciências Exatas, São José do Rio Preto

Orientador: Carlos Roberto Valêncio

1. Banco de dados. 2. Mineração de dados (Computação).. 3. Algoritmos paralelos. 4. Big Data. 5. Detecção de duplicações. I. Título.

Sistema de geração automática de fichas catalográficas da Unesp. Biblioteca do Instituto de Biociências Letras e Ciências Exatas, São José do Rio Preto. Dados fornecidos pelo autor(a).

JOÃO ROBERTO MARQUES GUBOLIN

Ambiente baseado em contexto para identificação de tuplas duplicadas com recursos de paralelização e meta-heurísticas

Trabalho de Conclusão de Curso apresentado ao Departamento de Ciências de Computação e Estatística do Instituto de Biociências Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Financiador: Conselho Nacional de Desenvolvimento Científico e Tecnológico – CNPq.

Banca examinadora:

Prof. Dr. Carlos Roberto Valêncio
UNESP – São José do Rio Preto
Orientador

Prof. Dr. Valeriano Antunes de Oliveira
UNESP – São José do Rio Preto

Prof. Dr. Adriano Mauro Cansian
UNESP – São José do Rio Preto

São José do Rio Preto
2021

Dedico este trabalho aos meus pais e aos meus avós.

Agradecimentos

Agradeço ao meu pai João e a minha mãe Nilza que sempre me apoiaram.

Agradeço ao meu irmão João Ricardo, aos meus avós João, Anita, João e Maria que sempre foram muito presentes.

Agradeço aos meus familiares, aos meus amigos que a vida proporcionou.

Agradeço ao grupo GBD e a todos seus membros e ex-membros pelas oportunidades e convívio, em especial ao professor Valêncio.

Agradeço a todos os professores do curso de Ciência da Computação que fizeram parte de minha graduação.

“Quando você tem a mente aberta e coração quente, é muito mais fácil explicar o que você quer.”

Abel Ferreira

Resumo

O problema da ocorrência de tuplas duplicadas não idênticas em base de dados consiste na existência de registros que se referem a um mesmo objeto, mas que são encontrados sob distintas representações. Por causarem inconsistências nas bases de dados e afetarem a qualidade dos resultados de processos de extração de conhecimento, os registros duplicados precisam ser identificados e removidos. Entretanto, ainda que existam algoritmos na literatura científica que realizam os procedimentos de detecção de duplicações, se fazem necessárias contribuições no sentido de sanar problemas atuais, como a necessidade de profundo conhecimento do conjunto de dados a fim de que seja escolhida uma função de similaridade eficaz, a complexidade NP-Hard do problema e as características do volumosos repositórios de dados do cenário Big Data. As estratégias como a utilização de meta-heurísticas tem se mostrado promissoras para lidar com a alta complexidade do problema, contudo, o volume de processamento adicional dispendido para que o contexto dos dados seja considerado e para que funções de similaridade sejam implementadas, acarreta em aumento do custo computacional, o que tende a inviabilizar a aplicação em grandes conjuntos de dados. Este trabalho tem por objetivo a apresentação de um ambiente eficiente que permita a identificação de tuplas duplicadas não idênticas, com utilização de mais de uma única função de similaridade. Como contribuição científica, tem-se um ambiente capaz de considerar o contexto de cada base de dados em todas as etapas de pré-processamento, com apoio de Algoritmos Genéticos na identificação de duplicações, além de apresentar uma abordagem paralela e em memória. Tais ingredientes considerados, em relação aos correlatos da literatura, oferecem a possibilidade de maximizar a quantidade de tuplas duplicadas não idênticas encontradas e consequente redução do tempo de processamento aplicados a grandes bancos de dados.

Palavras-chave: Banco de dados. Mineração de dados (Computação). Algoritmos paralelos. Big Data. Detecção de duplicações.

Abstract

The problem with the occurrence of non-identical duplicate tuples in a database consists of the existence of records that refer to the same object, but are found under different representations. Duplicate records need to be identified and removed, because they can cause inconsistencies in databases and affect the quality of the knowledge extraction process. However, even though there are algorithms in the scientific literature that can detect and remove duplications, they have to be improved in order to solve current problems. The problems are not considering the data context, the need for in-depth knowledge of the data set in order to use the best similarity function in that case, the NP-Hard complexity of the problem and the characteristics of Big Data repositories. The use of meta-heuristics has shown to be promising to deal with the high complexity of the problem. However, the amount of additional processing for considering the data context and choosing the similarity function automatically leads to an increase in computational cost, which can make the application in large data sets unfeasible. In this scenario, this work aims to create an efficient environment that allows the identification and removal of deduplication in data sets, with automatic similarity function selection according to the context. As a scientific contribution, the environment will allow context consideration of the database in all pre-processing steps, in addition to presenting a parallel approach and in memory, which, in relation to the literature correlates, will allow to maximize the number of deduplication found correctly, reducing processing time, making the application feasible to be used in the Big Data scenario.

Keywords: Database. Data mining. Parallel algorithms. Big Data. Duplication detection.

Lista de Ilustrações

Figura 1— Exemplo de Pré-processamento 20
Figura 2— Pseudocódigo do algoritmo de Damerau-Levenshtein 23
Figura 3— Pseudocódigo de um Algoritmo Genético 26
Figura 4— Diagrama do ambiente 29
Figura 5— Modelo de Entifade Relacionamento de Configuração do Ambiente 30
Figura 6— Exemplo de configuração inicial 31
Figura 7— Ilustração do Pré-processamento do Ambiente 31

Lista de Tabelas

Tabela 1— Caso de ocorrência de tuplas duplicadas..... 19
Tabela 2— Teste de comparação de qualidade na base de dados SIVAT..... 34
Tabela 3— Teste de comparação de qualidade na base de dados GEP..... 35
Tabela 4— Comparação de tempo de execução entre o ambiente sequencial e paralelizado.. 36

Lista de Abreviaturas e Siglas

AG	Algoritmos Genéticos
FPGA	*Field Programmable Gate Arrays*
GBD	Grupo de Banco de Dados
GEP	Gestão e Evolução de Prontuários
GPU	*Graphics Processing Unit*
HPC	*High-Performance Computing Clusters*
RDD	*Resilient Distributed Datasets*
SIVAT	Sistemas de Informação e Vigilância de Acidentes de Trabalho

Sumário

1 Introdução 14
1.1 Motivação e escopo 15
1.2 Objetivos 16
1.3 Metodologia 17
1.4 Organização da monografia 17
2 Fundamentação teórica 18
2.1 Limpeza de dados 18
2.2 Tuplas duplicadas 18
2.3 Pré-processamento 19
2.3.1 Tokenização 20
2.3.2 Remoção de palavras de parada 20
2.3.3 Redução de palavras flexionadas 21
2.4 Cálculo de similaridade 21
2.4.1 Funções de similaridade 21
2.4.2 Algoritmo de Damerau-Levenshtein 22
2.4.3 Algoritmo de Jaro-Winkler 23
2.5 Métricas de qualidade para detecção de tuplas duplicadas 24
2.6 Algoritmos Genéticos 25
2.7 Complexidade das bases de dados 26
2.8 Paralelização e processamento em memória 26
3 Desenvolvimento do trabalho 28
3.1 Ambiente proposto 28
3.1.1 Configuração inicial 29
3.1.2 Pré-processamento 30
3.1.3 Identificação de duplicações 31
3.2 Tecnologias utilizadas 32
4 Avaliação Experimental 33
4.1 Metodologia de experimentação 33
4.2 Teste de comparação de qualidade das funções de similaridade 34

4.3 Testes de desempenho do ambiente 35
4.4 Considerações finais 36
5 Conclusão 38
5.1 Contribuição científica 38
5.2 Atividades futuras 39

Introdução

Nas últimas décadas, a produção e armazenamento de dados tem sido cada vez mais recorrente. Este grande volume de dados se apresenta como um recurso valioso, uma vez que sua análise pode facilitar a tomada de decisões em diferentes contextos. Assim, é essencial para o avanço da tecnologia que novas técnicas de manipulação, extração, armazenamento, recuperação e análise de dados sejam desenvolvidas. Como os dados podem ser encontrados sob diversos tipos e formatos, em diferentes fontes de dados, referentes a diversos assuntos, a capacidade humana de análise, interpretação e tratamento é excedida, de modo que se faz necessário um auxílio computacional automatizado para que seja obtido conhecimento útil. Desta forma, surgiu o processo de extração de conhecimento em bases de dados (KDD, do inglês Knowledge Discovery in Databases), cujo objetivo é obter conhecimento útil de grandes bases de dados, mesmo que de forma não trivial (RISTOSKI; PAULHEIM, 2016).

Dentre todos as características encontradas em base de dados, uma das que mais afeta diretamente a qualidade dos resultados de processos de análises é a presença de informações duplicadas (DHIVYABHARATHI; KUMARESAN, 2015), uma vez que tornam estes processos demorados e custosos computacionalmente ao se considerar grandes volumes de dados. Na literatura científica esse problema é conhecido também como resolução de entidades, ligação de registro, correspondência de entidades, reconciliação de referência, detecção de duplicações, ou ainda, deduplicação (MESTRE, 2017; DHIVYABHARATHI; KUMARESAN, 2015). Entre os agentes causadores estão a ocorrência de sinônimos, abreviações, entrada de valores semelhantes ou não esperados, erros de digitação ou falta de padronização no formato

dos dados (DHIVYABHARATHI; KUMARESAN, 2015; BHARATHI; REDDY; SUDARSANA, 2017).

1.1 Motivação e escopo

A fim de permitir o correto tratamento dos dados duplicados, diversos algoritmos foram propostos. Todavia, esses algoritmos atuam de forma genérica, isto é, não consideram o contexto idiomático, linguístico ou vocabular de cada base de dados analisada. Como consequência, a saída dos métodos utilizados torna-se também genérica (LI et al., 2015, JANSSEN; VAN DER VOORT; WAHYUDI, 2017). Isso faz com que o resultado obtido possa não ser condizente ao esperado e fazer com que dados e informações importantes não sejam descobertos. Ademais, a não consideração do contexto pode causar inconsistências e incoerências nas demais fases da extração de conhecimento em que esses resultados sejam aplicados (JANSSEN; VAN DER VOORT; WAHYUDI, 2017).

A fim de garantir qualidade e bons resultados no processo de detecção de duplicações, tem-se como objetivo atingir boa precisão, boa eficiência e, ainda, reduzir taxas de valores falso positivos (WANDHEKAR; MOHANPURKAR, 2015). As funções, ou métricas, de similaridade são utilizadas para comparar e identificar duplicações dentre os dados armazenados em um banco de dados. Inicialmente, foram utilizadas funções de similaridade genéricas. Porém observou-se que, com o aumento da utilização de sistemas computacionais em diferentes contextos, os resultados não foram satisfatórios, uma vez que variam de acordo com características dos dados no qual foram empregados, como tamanho dos registros, contexto ou presença de caracteres especiais. Estratégias foram propostas na literatura científica para adaptar estas funções à um domínio específico, como a proposta por Chaudhuri et al. (2003). Ainda que tenham sido importantes, estas estratégias são muito dependentes de um controle humano, com conhecimento prévio do banco de dados. Desta forma, há necessidade de desenvolvimento de estratégias capazes de realizarem uma escolha criteriosa destas funções e que permitam livrar-se da dependência humana.

Outro aspecto a se considerar na execução de um processo de deduplicação de dados, é o alto custo computacional. Este problema, tal como apresentado por Ganti e Sarma (2013), tem complexidade NP-hard. Diante disso, foram propostas abordagens com base na aplicação de metaheurísticas para sua resolução, como Algoritmos Genéticos (AG), Otimização por Colônia de Formigas, entre outras, o que permite que a busca por soluções sem que seja necessário percorrer todo o espaço de busca. Dentre essas técnicas, os Algoritmos Genéticos se

destacam, superando-as pelas suas boas propriedades de convergência (DAS; MULLICK; SUGANTHAN, 2016).

Diante da necessidade de consideração do contexto dos dados, escolha eficaz e automatizada de funções de similaridade que apresentem bons resultados em relação aos dados a serem tratados, assim como necessidade de utilização de meta-heurísticas para lidar com complexidade do problema, percebe-se a dificuldade de realização de processos de deduplicação em grandes conjuntos de dados. Mesmo para as técnicas de otimização que fazem uso de meta-heurísticas, a análise de uma grande quantidade de soluções, aliada ao processamento adicional dos métodos de consideração do contexto e escolha de funções de similaridade, indicam o aumento no tempo de processamento. A computação paralela e o processamento de dados em memória têm se tornado alternativas viáveis, pois permitem a resolução de problemas complexos de maneira mais rápida. Assim há necessidade de ferramentas e técnicas que venham a contribuir com a detecção e remoção de duplicações de maneira eficaz e com boa qualidade, ao mesmo tempo que sejam viáveis computacionalmente para serem executadas em grandes conjuntos de dados presentes no âmbito Big Data (RAMYA et al., 2017).

1.2 Objetivos

Tendo em vista a importância da remoção de duplicações a fim de garantir qualidade à descoberta de conhecimento em bases de dados e dos desafios presentes na execução desta etapa, este trabalho tem por objetivo a elaboração de um ambiente eficiente que permita a identificação e remoção de tuplas duplicadas não idênticas, no cenário Big Data, de modo que seja possível identificar a melhor função de similaridade, sem que seja necessário conhecimento profundo da natureza dos dados.

Além disso, como contribuição científica, o ambiente permitirá que o contexto de cada base de dados seja considerado em todas as etapas de pré-processamento, além de apresentar uma abordagem paralela e em memória por meio do framework Apache Spark™, o que, em relação aos correlatos da literatura, permitirá maximizar a quantidade de tuplas duplicadas não idênticas encontradas corretamente e reduzirá o tempo de processamento. Esse framework tem se mostrado eficiente em trabalhos semelhantes de manipulação de tuplas e também superior a outros métodos de paralelização (HILDEBRANDT, 2017; PINTO, 2015).

1.3 Metodologia

Inicialmente foi realizado o levantamento bibliográfico a respeito de conceitos necessários para a relização do trabalho, em especial tópicos sobre Big Data, limpeza de dados, deduplicação, pré-processamento, funções de similaridade, Algoritmos Genéticos e paralelização e processamento em memória.

Em seguida, foi elaborado um ambiente capaz de realizar as etapas de pré-processamento de uma base de dados seguido pela identificação das tuplas duplicadas, com auxílio de algoritmos genéticos e paralelização, a fim de otimizar o processo.

Após realização do estudo sobre os conceitos teóricos e elaboração conceitual do ambiente foi realizada a implementação do sistema e condução de testes, buscando a validação dos resultados obtidos.

1.4 Organização da monografia

O presente trabalho apresenta nesta primeira seção o contexto atual de identificação de tuplas duplicadas sob o cenário atual Big Data e suas dificuldades. Foi exposto a motivação e objetivo do trabalho, bem como os recursos disponíveis para sua execução.

As próximas seções estão organizados da seguinte forma:

a) Seção 2 – Fundamentações teóricas referentes aos assuntos pertinentes ao escopo do trabalho incluindo Big Data, limpeza de dados, deduplicação, pré-processamento, funções de similaridade, Algoritmos Genéticos e paralelização e processamento em memória.
b) Seção 3 – Indtrodução e explicação do ambiente desenvolvido no trabalho em questão, de modo que é mostrado seu funcionamento, partindo da configuração inicial do usuário, que leva em conta o contexto dos dados de entrada, passando pelo pré-processamento e chegando na parte de identificação de duplicações, feita de forma que um algoritmo, dentre dois implementados, é escolhido com base em sua aptidão ao contexto dos dados, atuando de forma paralelizada.
c) Seção 4 – Avaliação experimental do ambiente, de modo que é mostrada a metodologia de experimentação feita, buscando a validação do trabalho e mostrando os resultados obtidos.
d) Seção 5 – Apresentação das conclusões, mostrando as contribuições científicas do trabalho e as atividades futuras a serem feitas.

2 Fundamentação teórica

Esta seção tem como propósito apresentar a fundamentação teórica e conceitos essenciais a respeito de limpeza de dados, tuplas duplicadas, pré-processamento dos dados, funções de similaridades usadas, meta heurísticas e técnicas de paralelização.

2.1 Limpeza de dados

O processo de extração de conhecimento a partir de dados pode trazer muitos benefícios, como por exemplo melhorar a tomada de decisões, tornar análises mais completas e providenciar cada vez mais dados para serem utilizados no aprendizado de máquinas. No entanto, a qualidade dos dados cada vez mais se torna um problema a ser solucionado. Dados que ainda não passaram por um processo de limpeza podem conter valores faltantes, erros de digitação, formatos misturados e outros problemas, que podem acarretar em tomadas de decisões erradas e análises incorretas (CHU et al., 2016).

Nesse aspecto, o processo de limpeza de dados atua para melhorar a qualidade desses, ao lidar com esses problemas citados que estão presentes em dados ainda não tratados. O processo usualmente consiste em duas etapas: detecção de erros e reparação de erros (CHU et al., 2016).

Ao considerar o cenário do Big Data, essa limpeza dos dados se torna necessára, devido à dificuldade em se processar volumes ainda maiores de dados em uma velocidade também considerável (WANG et al., 2014). Assim, surgiram muitas formas de se realizar a limpeza dos dados, porém ela ainda é vista como um desafio. Assim, pode-se buscar aprimorar cada vez mais as soluções existentes para o problema.

2.2 Tuplas duplicadas

Um tipo de problema que é resolvido na limpeza de dados é a ocorrência de tuplas duplicadas, que podem surgir por inúmeras razões. Esse problema é resolvido por meio da deduplicação dos dados, processo em que ocorre a identificação das tuplas, em uma ou mais

relações, que se referem a uma mesma entidade. O processo é geralmente seguido por uma consolidação de entidades ou fusão, que resulta em uma representação unificada para entradas que representam uma mesma entidade (ILYAS; CHU, 2019).

A solução para esse problema pode se tornar de alta complexidade. Porém, um exemplo simples de um caso está ilustrado na Tabela 1, em que produtos foram cadastrados mais de uma vez em categorias diferentes. Na tabela são apresentados os nomes de produtos duplicados com os identificadores (ID) 1 e 2, o mesmo ocorre com os nomes de produtos 3 e 4.

Tabela 1 – Caso de ocorrência de tuplas duplicadas

ID	categoria_produto	nome_produto
1	cozinha	detergente
2	limpeza	detergente
3	cozinha	bucha
4	limpeza	bucha

Fonte: Elaborada pelo autor.

Algoritmos de identificação de duplicações podem identificar as tuplas duplicadas e removê-las com a unificação das entidades iguais, o que reduz a quantidade de entidades do exemplo apresentado pela metade.

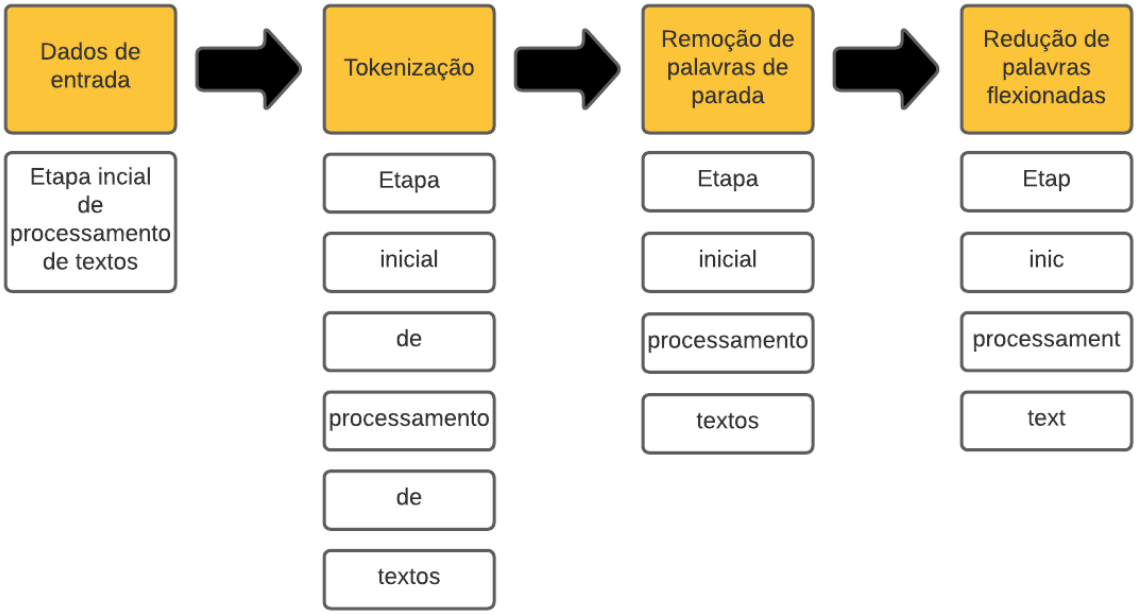
2.3 Pré-processamento

Para que ocorra a etapa de deduplicação dos dados, existem etapas prévias no processo de limpeza de dados que devem ser executados para se facilitar essas etapas posteriores (DHIVYABHARATHI; KUMARESAN, 2015). O processo começa com a separação dos dados a serem analisados, seguido por um pré-processamento desses. Após isso, ocorre a etapa de deduplicação, onde são utilizados algoritmos para se detectar se existem as duplicações e depois faz-se uma interpretação dos resultados obtidos, verificando se são realmente coerentes.

Na etapa específica do pré-processamento, os dados brutos são preparados para as etapas seguintes, sendo transformados em um formato mais simplificado (DHIVYABHARATHI; KUMARESAN, 2015). Entre os principais passos do pré-processamento estão a tokenização, a remoção de palavras de parada e a redução de palavras flexionadas (VIJAYARANI; ILAMATHI; NITHYA, 2015). Na Figura 1 é ilustrado esse processo. Pode-se observar os dados brutos de uma entrada, seguido pelo processo de tokenização. Na etapa de remoção de palavras de parada, observa-se que foi removida a palavra “de”, uma preposição. Na última etapa, foi feita a redução das palavras flexionadas restantes, removendo as terminações das palavras. Esse

processo acaba por diminuir o tamanho e a quantidade dos dados de entrada, sem que se altere o sentido deles, ou seja, sem que seja feita uma diminuição da qualidade dos dados.

Figura 1 – Exemplo de pré-processamento



Fonte: Elaborada pelo autor.

2.3.1 Tokenização

O processo de tokenização consiste em dividir o texto recebido em palavras ou tokens, ou seja, transformar um conjunto de palavras, que seria complexo de analisar conjuntamente, em tokens singulares, mais fáceis de serem analisados (REZAEIAN; MONTAZERI; LOONEN, 2017).

Durante esse processo, podem ser removidos caracteres simbólicos, sinais de pontuação e outras palavras que não fazem contribuição semântica (ALAM; YAO, 2018). Feito esse processo, obtém-se um conjunto de tokens, mais fácil de ser analisado, reduzindo-se também o tamanho de caracteres que não contribuem com a identificação do sentido das palavras.

2.3.2 Remoção de palavras de parada

A remoção de palavras de parada se dá pela retirada de palavras que não são cruciais para o contexto de um texto, ou seja, palavras como artigos, preposições e pronomes. Essas palavras geralmente fazem o texto ficar pesado e pouco importante para uma análise,

aumentando a dimensão do texto analisado e contribuindo pouco para o contexto (VIJAYARANI; ILAMATHI; NITHYA, 2015).

2.3.3 Redução de palavras flexionadas

A redução de palavras flexionadas pode ser definida como uma radicalização de palavras, no qual ocorre a remoção de afixos, plurais e terminações verbais, contudo o sentido das palavras é mantido. Isso acarreta numa diminuição do tamanho dos dados e também da complexidade do texto analisado, sem ocorrer a perda do contexto desse (JIVANI, 2011).

Vários algoritmos foram propostos para a finalidade de radicalizar as palavras, sendo que eles podem ser baseados em análises probabilísticas ou em regras. O primeiro caso se baseia em análises estatísticas para se definir quais são os radicais que devem ser retirados. Já o segundo tipo se baseia em uma lista pré definida de prefixos e sufixos de acordo com o idioma dos dados analisados. O segundo método, por considerar as regras de uma determinada linguagem específica, se mostra mais eficaz. (MORAL et al., 2014).

2.4 Cálculo de similaridade

Após ocorrido o pré-processamento dos dados, reduzindo-se o tamanho do conjunto de busca, utiliza-se algoritmos para identificar a similaridade das palavras na base de dados, identificando as duplicações existentes (YU, 2016). Essas funções de similaridade entre strings podem ser classificadas em três categorias, sendo elas: baseadas em caracteres, baseadas em tokens e baseadas em fonética (ILYAS; CHU, 2019).

Ao se considerar o âmbito da detecção de duplicações em dados textuais, acaba se utilizando de algoritmos baseados em caracteres. Esses algoritmos atuam verificando a grafia das palavras analisadas, sem se preocupar com o significado delas (ALENAZI; AHMAD, 2016).

2.4.1 Funções de similaridade

As funções de similaridade baseadas em caracteres se baseiam na comparação da grafia das strings, levando em consideração a distância entre uma string 1 de uma string 2. Para o cálculo dessa distância, são executadas operações para converter uma string em outra, sendo a quantidade total de operações realizadas determinada como a distância entre as strings. Esse tipo de função é muito utilizada para identificar erros ortográficos e também na área da bioinformática, para a comparação de cadeias genômicas, visto que compara os caracteres

individualmente. O tamanho da string comparada pode afetar o desempenho significativamente, porém a comparação feita é muito precisa devido à comparação de todos os caracteres feita (ALENAZI; AHMAD, 2016).

O segundo tipo das funções de similaridade, baseadas em tokens, acaba verificando os tokens formados nas tuplas, ao invés de analisá-las de uma forma geral. Um exemplo é se comparando as tuplas “Marcos Silva” com “Silva Marcos ”, seria dado o caso como uma possível duplicação, devido a não se considerar a ordem (ILYAS; CHU, 2019).

Já o último tipo de função de similaridade de strings é o baseado em fonética. Neles são buscadas semelhanças entre as strings, levando-se em conta a pronúncia das palavras no devido idioma analisado. Um exemplo na língua portuguesa seria comparar “Posso pegar esse lápis?” com “Poço pegar ece lápiz”. As duas tuplas seriam vistas como possíveis duplicações, devido à consideração do som das pronúncias das palavras (ILYAS; CHU, 2019).

2.4.2 Algoritmo de Damerau-Levenshtein

Um algoritmo bastante utilizado na literatura para medição de similaridade de strings baseado em caracteres é o de Damerau-Leveshtein (BALHAF et al., 2017). Esse algoritmo atua comparando a distância entre duas strings, contando o número mínimo de operações para uma string ficar igual outra, sendo ideal para identificar erros de digitação na comparação de duas palavras (PRASETYA; WIBAWA; HIRASHIMA, 2018). As operações consideradas para fazer essa comparação são 4, sendo elas: inserção, deleção, substituição e transposição (BARD, 2007). Um exemplo é que a distância de edição das strings “Olimpíadas 2020” e “Olimpíadas 2016” é 1, devido ao fato de se precisar fazer apenas uma operação para as strings ficarem iguais.

O algoritmo em questão apresenta como entrada duas *strings* a serem comparadas. O algoritmo então percorre os caracteres das palavras, identificando a distância entre as duas *strings* de entrada. Após percorrer as entradas, realizando as operações de inserção, deleção, substituição e transposição, o algoritmo vai retornar a distância de edição entre as duas *strings*. Na Figura 2 é mostrado o pseudocódigo do algoritmo.

Figura 2 – Pseudocódigo do algoritmo de Damerau-Levenshtein

```
1   Entrada: s, t: strings de caracteres.  Saída: distância de edição: inteiro; m, n: inteiros
2   Começar:
3       //Passo 1: Verificar se as strings são vazias
4       m = |s|
5       n = |t|
6       if m=0: return n end-if
7       if n=0: return m end-if
8       //Passo 2: Criar a matriz de distância d
9       for i=0 to m: d[i][0] = i end-for
10      for j=1 to n: d[0][j] = j end-for
11      //Passo 3: Calcular a matriz de distância d
12      for i=1 to m
13          for j=1 to n
14              if s[i-1] = t[j-1]
15                  d[i][j] = d[i-1][j-1]
16              else
17                  d[i][j] = min(d[i-1][j], d[i][j-1], d[i-1][j-1]) + 1
18                  if i>1 and j>1
19                      if s[i-1] = t[j-2] and s[i-2] = t[j-1]
20                          d[i][j] = min(d[i][j], d[i-2][j-2]+1)
21                      end-if
22                  end-if
23              end-if
24          end-for
25      end-for
26      return d[m][n]
27  end
```

Fonte: Adaptada de (ALKANHAL, 2012).

Apesar de muito utilizado, o algoritmo de Damerau-Levenshtein apresenta o problema de possuir uma dependência no desempenho de acordo com o tamanho das strings a serem comparadas, possuindo complexidade O(*mn*) para os pares de strings comparadas, sendo *m* e *n* o tamanho das strings. Isso fica ainda mais significativo quando utilizado em grandes bancos de dados, tornando o tempo de processamento inviável (BALHAF et al., 2017).

2.4.3 Algoritmo de Jaro-Winkler

Outro algoritmo baseado em caracteres muito utilizado na literatura para comparação de strings é o de Jaro-Winkler. Esse algoritmo funciona calculando a similaridade entre as strings com apenas operações de transposição, podendo obter valores entre 0 e 1, sendo 0 strings sem nenhuma similaridade e 1 strings iguais (LEONARDO; HANSUN, 2017).

O algoritmo funciona primeiro recebendo duas strings de entrada, calculando-se o tamanho das entradas. Em seguida é feita uma busca para encontrar a quantidade de caracteres iguais nas duas strings. Por fim, é encontrado a quantidade de transposições necessárias para se igualar as strings. A distância então é calculada por meio da seguinte fórmula:

$$d = \frac{1}{3} * \frac{m}{|s1|} * \frac{m}{|s2|} + \frac{m-t}{m}$$

(1).

Para a função (1), tem-se que *d* é a distância obtida, *m* é a quantidade de caracteres iguais entre as strings, *s1* é o tamanho da string 1, *s2* é o tamanho da string 2 e *t* é a quantidade de transposições para se igualar as strings (LEONARDO; HANSUN, 2017). Após encontrada a distância *d*, o algoritmo ainda leva em consideração se as strings possuem um prefixo igual, calculando uma distância ajustada. Esse segundo cálculo atua de modo a facilitar a identificação de palavras com o mesmo prefixo.

Esse algoritmo é muito eficiente para obter semelhanças em nomes pessoais e de entidades, sendo assim, possui uma vantagem em relação a bases de dados com alta concentração desse tipo de dados. No entando, conforme o tamanho do banco de dados analisado aumenta, esse algoritmo se torna mais custoso em termos de processamento (WANG; QIN; WANG, 2017).

2.5 Métricas de qualidade para detecção de tuplas duplicadas

Para se verificar se as duplicações de uma base de dados foram identificadas corretamente, faz-se a utilização de métricas para a avaliação dos resultados obtidos após o processo de deduplicação (KÖPCKE; RAHM, 2010).

Uma das métricas é o cálculo da eficácia, que faz a medida de quanto todas as duplicações foram encontradas sem que nos resultados existam saídas que não sejam de fato dados duplicados. Assim sendo, se diz respeito à revocação obtida e à precisão dos resultados (DRAISBACH, U. et al., 2012).

A revocação, identificada como *r,* se refere à relação do número de pares duplicados identificados corretamente, com número de pares duplicados existentes na base de dados analisada.

$$r = \frac{pares\ duplicados\ identificados\ corretamente}{pares\ duplicados\ existentes}$$

(2).

Já a precisão, identificada por *p,* se refere à relação do número de pares duplicados identificados corretamente, com o número de pares duplicados identificados no processo de detecção de duplicações (DRAISBACH, U. et al, 2012).

$$p = \frac{pares\ duplicados\ identificados\ corretamente}{pares\ duplicados\ identificados} \quad (3).$$

Outra métrica capaz de medir a eficácia da detecção das tuplas duplicadas relaciona as duas outras métricas anteriores. Essa métrica é chamada de F-measure e tem por base o cálculo da média harmônica entre as métricas anteriores, simbolizando portanto, o equilíbrio entre as mesmas (CHAN; CHEN; TOWEY, 2002). O cálculo é feito da seguinte forma:

$$F - measure = \frac{2pr}{p + r} \quad (4).$$

O valor obtido pela última métrica varia entre 0 e 1, identificando o equilíbrio alcançado entre os valores das outras duas métricas usadas.

2.6 Algoritmos Genéticos

A utilização de Algoritmos Genéticos (AG) é uma solução adequada para resolver problemas de busca e otimização (GASTONI; GRANELLI; MONTAGNA, 2013). Um AG pode ser definido como um algoritmo de busca inspirado na seleção natural encontrada na natureza, em que os indivíduos mais aptos são selecionados (SHARMA; SABHARWAL; SIBAL, 2014).

O funcionamento dos AG contém as etapas de seleção, recombinação e por fim mutação. Primeiro, ocorre uma seleção de possíveis soluções em espaços de busca do problema, cada espaço é chamado de cromossomo. Em seguida, os cromossos são submetidos a um processo evolucionário em que seus indivíduos são avaliados, selecionados e recombinados, buscando assim uma maior variabilidade. Após a repetição desse processo, busca-se encontrar os indivíduos mais aptos, ou seja, a solução para o problema inicial em questão (SHARMA; SABHARWAL; SIBAL, 2014). Na Figura 3 ilustra-se o pseudocódigo de um AG.

Figura 3 – Pseudocódigo de um Algoritmo Genético

```
1   Inicialização (população)
2   Avaliação (população)
3   Enquanto (condição de parada não satisfeita)
4   {
5           Seleção (população)
6           Crossover (população)
7           Mutação (população)
8           Avaliação (população)
9   }
```

Fonte: Adaptada de (SHARMA; SABHARWAL; SIBAL, 2014).

2.7 Complexidade das bases de dados

Visto o cenário atual do Big Data, tem-se cada vez mais uma necessidade de maior complexidade computacional para processar os dados, sendo possível o escalonamento de sistemas. Esse escalonamento pode ser obtido tanto por escalabilidade horizontal quanto vertical (SINGH; REDDY, 2015).

A escalabilidade horizontal se refere à capacidade de se distribuir a carga de trabalho para processamento entre diferentes máquinas. O processo envolve realizar a coordenação das múltiplas máquinas para promover um aumento da capacidade computacional obtida em conjunto (ALI, 2019).

Já a escalabilidade vertical se refere ao aumento da capacidade computacional de uma mesma máquina por meio da instalação de processadores adicionais, melhoria de hardware e instalação de mais memórias para realizar o processamento, sendo alguns exemplos as GPUs (*Graphics Processing Unit*), HPCs (*High-Performance Computing Clusters*) e FPGAs (*Field Programmable Gate Arrays*) (ALI, 2019).

De maneira geral, a escalabilidade horizontal acaba sendo a mais eficiente para lidar com o Big Data, devido a permitir a utilização de escalabilidade vertical em cada máquina singularmente junto com a escalabilidade horizontal em que novos servidores podem ser adicionados para realizar o aumento da capacidade computacional (HILDEBRANDT, 2017).

2.8 Paralelização e processamento em memória

Várias técnicas e ferramentas foram surgindo para a realização do processamento paralelo e em memória, muito útil na melhoria do processamento de dados, necessário cada vez mais quando lidando com o cenário Big Data. Uma das *frameworks* mais utilizadas recentemente para possibilitar esse processamento é o Apache Spark (ALI, 2019).

A ferramenta foi desenvolvida para superar os obstáculos apresentados na performance de operações de entrada e saída da ferramenta Hadoop, também muito utilizada para processamento de grandes quantidades de dados (KATIB, 2019). A *framework* possibilita o processamento em memória por meio da utilização de RDDs (*Resilient Distributed Datasets*) que são abstrações geradas capazes de armazenar dados com tolerância a falhas e armazenadas de maneira paralela (ALI, 2019).

Uma outra vantagem é que se comparada com a *framework* Hadoop, o Apache Spark se mostrou ser mais eficiente tanto no processamento em memória quanto em disco, sendo assim uma ótima solução para o uso em processamento de grandes quantidades de dados (POL, 2016).

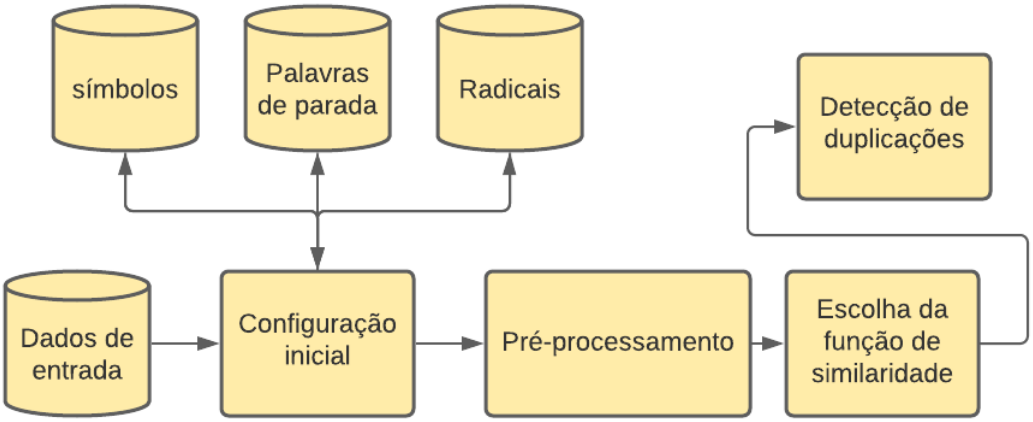
3 Desenvolvimento do trabalho

A atual seção tem por objetivo mostrar as tecnologias que foram utilizadas no desenvolvimento deste trabalho. Foi desenvolvido um ambiente capaz de realizar o pré-processamento dos dados textuais e a devida identificação de tuplas duplicadas. O ambiente foi desenvolvido se utilizando-se a linguagem de programação Java. Foi utilizado também o *framework* Apache Spark, o qual possibilita a parelelização dos algoritmos e processamento em memória.

3.1 Ambiente proposto

O ambiente proposto é dividido em três partes principais: configuração inicial, pré-processamento e identificação de duplicações. Na parte de configuração inicial, o usuário é capaz de realizar as configurações para os dados de entrada, levando em consideração o contexto da base de dados que será analisada. Na segunda parte do ambiente, os dados escolhidos para análise são pré-processados, visando um melhor funcionamento do ambiente na etapa seguinte. Já na terceira e última etapa, os dados pré-processados passam por uma meta heurística baseada em algoritmos genéticos, no qual será avaliado a eficiência da base analisada para os dois algoritmos implementados. Assim, o de melhor desempenho será escolhido e utilizado na identificação das duplicações. Recursos de paralelização e processamento em memória foram utilizados a fim de reduzir o tempo de processamento. Na Figura 4 apresenta-se a visão geral do ambiente.

Figura 4 – Diagrama do Ambiente



Fonte: elaborada pelo autor.

3.1.1 Configuração inicial

A primeira parte do ambiente tem como função inicial a entrada dos dados por parte do usuário. Isso é feito com a inserção e configuração das propriedades da base de dados desejada, sendo escolhidas a tabela e coluna desta que passará pelo processo de detecção de duplicações. Os dados escolhidos, são então armazenados em uma estrutura RDD do Apache Spark, sendo elementos do tipo <chave, valor>, contendo respectivamente a chave primária e o dado da tabela e coluna escolhidas.

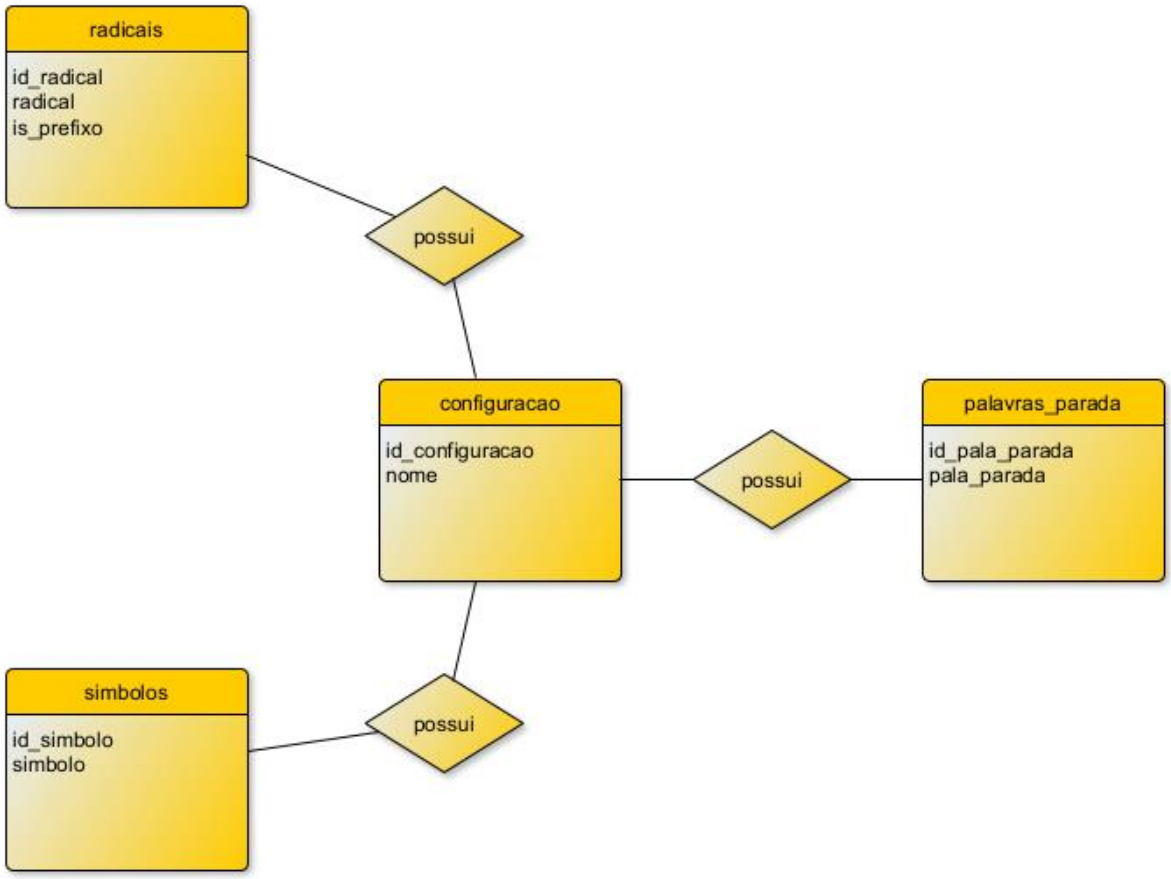
Após realizada a configuração da entrada de dados desejada, o usuário também deve configurar o parâmetro de distância que será utilizado pelo algoritmo de detecção de duplicações. Esse parâmetro irá determinar qual deve ser a distância máxima de procura, ou seja, quanto menor o parâmetro passado pelo usuário, maior será a similaridade entre os pares de dados encontrados.

Ainda nessa parte, o usuário deve adicionar os símbolos e sinais de pontuação que devem ser removidos pelo ambiente, seguido pelas palavras de parada desejadas do contexto e por fim, devem ser configurados e inseridos os radicais a serem considerados pelo ambiente. Nesse último caso, o usuário deve indicar ao ambiente se eles são prefixos ou sufixos, por meio de uma variável booleana adicionada. Feitas essas definições, o ambiente terá a configuração da manipulação que deve ser realizada nos dados durante o pré-processamento.

A configuração dessa primeira parte é feita pelo usuário por meio da inserção das regras de contexto desejadas e inseridas em tabelas do ambiente. Deve ser feita a escolha de inserir dados o banco de dados criado, o qual é mostrado na Figura 5, que ilustra o modelo entidade

relacionamento do banco de dados do ambiente. Assim, podem ser incluídos radicais, símbolos e palavras de paradas a serem removidas durante a etapa de pré-processamento.

Figura 5 – Modelo de Entidade Relacionamento de Configuração do Ambiente



Fonte: elaborada pelo autor.

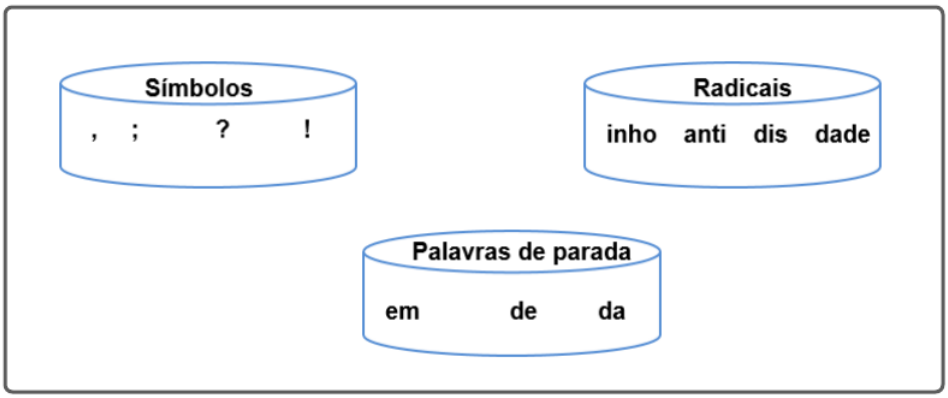
3.1.2 Pré-processamento

A segunda parte do ambiente tem como função realizar a padronização dos dados, conforme especificado na configuração inicial do usuário, de modo a se adequar no contexto da base de dados desejada. Também é feita uma padronização dos dados, facilitando o processo de identificação das duplicações.

Assim, o ambiente primeiro realiza uma padronização dos dados, colocando as entradas em letras minúsculas. Em seguida, os símbolos e sinais de pontuação são removidos, seguido pela tokenização das palavras, transformando frases em listas de palavras. Sequencialmente, são removidos os acentos das palavras, percorrendo as listas geradas anteriormente. Por fim, as palavras de parada encontradas são também removidas, seguido pela remoção das palavras flexionadas, identificadas pelos afixos e sufixos adicionados à configuração, de modo que ficam

restando apenas os radicais das palavras. Na Figura 6 é ilustrado um exemplo de possíveis entradas cadastradas por um usuário na configuração inicial do ambiente.

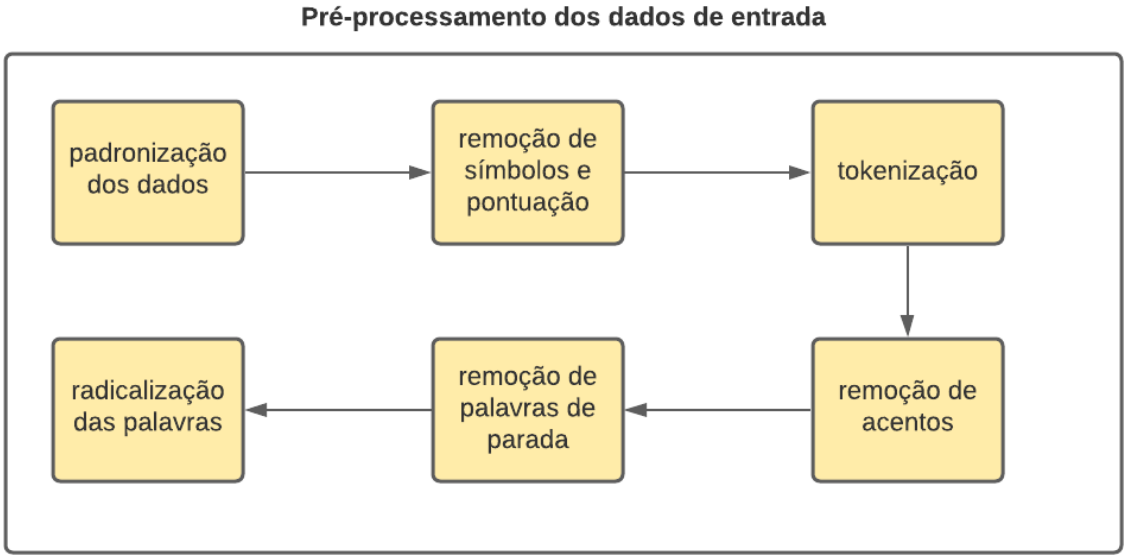
Figura 6 – Exemplo de configuração inicial



Fonte: elaborada pelo autor.

O processo de pré-processamento é realizado de forma paralela, utilizando-se da função map do Apache Spark, que se aplica ao RDD dos dados de entrada. Na Figura 7 é ilustrado o processo dessa segunda parte do ambiente.

Figura 7 – Ilustração do Pré-processamento do Ambiente



Fonte: elaborada pelo autor.

3.1.3 Identificação de duplicações

Na terceira e última parte do ambiente, é realizada a identificação das duplicações dos dados de entrada. Nesta etapa é feita a seleção da função de similaridade mais adequada aos dados analisados. Isso é feito por meio de um algoritmo genético que funciona de forma que uma amostra inicial randômica dos dados é selecionada e dividida em partições usando RDD. Cada uma das partições fazem operações independentes das outras. Assim, o ambiente é capaz de avaliar os algoritmos implementados para realizar a identificação das duplicações, sendo eles o de Damerau-Levenshtein e o de Jaro-Winkler para o contexto dos dados atuais se utilizando de diferentes partições ao mesmo tempo.

Como critério de avaliação do algoritmo de similaridade, é levado em consideração o número de amostras duplicadas encontradas e o tempo que esse processo leva. Essa amostra de dados passa pelo processo de operações de aptidão independente, de modo que, como critério de seleção, é utilizado uma troca randômica de alguns dados da amostra para gerar a próxima geração de dados a serem testados. Assim, quando o parâmetro de parada, que deve ser informado baseado no tempo e número de amostras duplicadas encontradas, for atingido, o processo de seleção do algoritmo de similaridade mais adequado se encerra, de modo que o primeiro a atingir o critério de parada é selecionado para o contexto dos dados em questão.

3.2 Tecnologias utilizadas

O ambiente foi desenvolvido na linguagem Java, implementada por meio da interface de desenvolvimento Netbeans. Essas tecnologias foram escolhidas pois a linguagem escolhida possui fácil implementação em conjunto com o Framework Apache Spark, de modo que facilita a implementação da paralelização dos algoritmos, otimizando o tempo de processamento total.

Para o armazenamento de regras de configuração iniciais foi utilizado o Sistema Gerenciador de Banco de Dados PostgreSQL, o qual possui fácil implementação e é open source, possibilitando sua utilização de forma simples, rápida e sem custos.

4 Avaliação Experimental

Neste capítulo são apresentados os testes e resultados obtidos com a execução do ambiente proposto neste trabalho, bem como a metodologia de experimentação, de modo a aferir a qualidade dos resultados obtidos.

4.1 Metodologia de experimentação

A experimentação foi realizada com a execução de testes a fim de se analisar a qualidade e o desempenho do ambiente. Os testes de qualidade buscaram verificar se a utilização de dois algoritmos de identificação de duplicações resultaria em uma diferença nos valores de tuplas duplicadas encontradas, se utilizando de bases de dados diferentes. Já os testes de desempenho buscaram comparar os desempenhos do algoritmo sequencial e paralelizado, mostrando a viabilidade e adaptação do ambiente ao cenário Big Data.

A análise de qualidade busca mostrar e comparar a utilização de apenas uma função de similaridade para a realização das buscas, mantendo-se o mesmo limite de parâmetro de distância máxima. Essa comparação permite mostrar que dependendo do contexto da base de dados analisada, um algoritmo de identificação de duplicações pode ser mais eficiente que o outro. A comparação foi feita com a extração de dados da base de dados do Sistemas de Informação e Vigilância de Acidentes de Trabalho (SIVAT) e da base de dados do sistema de Gestão e Evolução de Prontuários (GEP), ambas fornecidas pelo GBD para fins acadêmicos.

Ambas as bases de dados analisadas contém registros preenchidos manualmente, o que pode ocasionar registros com erros de digitação e ortografia, podendo ocasionar o problema dos registros duplicados.

A configuração inicial do ambiente para cada base de dados foi feita com a inclusão de 5 regras de palavras de parada, 3 regras de símbolos, e 15 regras de radicais (tanto sufixos como prefixos).

Para os testes de desempenho, foram feitas execuções do ambiente com a utilização de diferentes valores de entrada, comparando-se a execução dos algoritmos sequenciais e paralelos.

Todos os testes foram realizados na IDE Netbeans versão 11.2 em uma máquina com sistema operacional Windows 10 Home de arquitetura 64 bits, com um processador AMD Ryzen 5 2400G 3.60 Ghz, 16 Gb de memória RAM e uma GPU Nvidia Geforce GTX 1050 Ti.

4.2 Teste de comparação de qualidade das funções de similaridade

Primeiro foi feita a análise de qualidade por meio do teste com a comparação das duas funções de similaridade para as duas bases de dados. Nesse primeiro teste, foram utilizadas as regras para cada contexto, utilizando a mesma quantidade de regras de configuração inicial. Nesse teste, o foco era analisar a diferença da atuação das funções de similaridade nas bases de dados, por isso a meta heurística de seleção do algoritmo mais adequado não foi utilizada.

A configuração inicial foi feita utilizando-se o parâmetro de distância máxima igual a 3, seguido pela configuração de 23 regras inseridas para cada contexto. O primeiro caso de teste foi feito utilizando-se de uma amostra de dados do SIVAT, a qual verificou-se manualmente que possuía 44 registros duplicados. O ambiente foi executado utilizando-se primeiro apenas o algoritmo de Damerau-Levenshtein, seguido pela execução do algoritmo de Jaro-Winkler. Após a obtenção das duplicações, foi feito um cálculo de precisão, revocação e F-measure dos algoritmos, utilizando-se das fórmulas mostradas na seção 2.5. Na Tabela 2 mostra-se os resultados obtidos nesse primeiro teste.

Tabela 2 – Teste de comparação de qualidade na base de dados SIVAT

Algoritmos	Pares encontrados	Duplicações encontradas	Precisão	Revocação	F-measure
Damerau-Levenshtein	49	43	0.877	0.977	0.924
Jaro-Winkler	58	40	0.690	0.909	0.785

Fonte: elaborada pelo autor.

Após essa primeira execução, foi feita uma segunda execução de maneira similar no contexto de dados do segundo banco de dados, o qual foi o banco de dados do GEP. Para esse caso, verificou-se manualmente que a amostra escolhida possuía 32 registros duplicados. Foram

calculadas novamente as métricas mostradas na seção 2.5. Os resultados podem ser observados na Tabela 3.

Tabela 3 – Teste de comparação de qualidade na base de dados GEP

Algoritmos	Pares encontrados	Duplicações encontradas	Precisão	Revocação	F-measure
Damerau-Levenshtein	46	31	0.674	0.969	0.795
Jaro-Winkler	40	31	0.775	0.969	0.861

Fonte: elaborada pelo autor

Conforme os resultados observados nas Tabelas 2 e 3 pode-se concluir que os algoritmos, quando empregados para um mesmo contexto, apresentam algumas diferenças nos resultados, isso indica uma certa vantagem na utilização de um algoritmo mais apropriado para um contexto específico. Na amostra de dados vindas da base do SIVAT, o algoritmo mais eficiente foi o de Damerau-Levenshtein, com os melhores valores encontrados de precisão, revocação e F-measure, enquanto que na amostra de dados do GEP, o algorítmo mais eficiente foi o de Jaro-Winkler, caso em que houve a mesma precisão dos dois algoritmos. No entanto, o algoritmo em questão teve melhores precisão e F-measure. Isso indica uma certa vantagem na utilização desse algoritmo em um certo contexto de dados. No caso, a amostra de dados retirada do GEP, possuía dados com nomes de pacientes, característica de dados em que esse algoritmo é mais eficiente em encontrar, falhando menos vezes, o que ocasionou uma maior precisão.

4.3 Testes de desempenho do ambiente

O teste seguinte busca medir o desempenho do ambiente conforme se aumenta o número de amostras iniciais de dados analisadas, comparando-se a implementação sequencial com a paralelizada. Assim, foram testadas situações com mil, 2 mil, 4 mil, 8 mil e 16 mil dados iniciais retirados da base de dados do SIVAT. A execução dos testes foi repetida por cinco vezes para cada situação, de forma a calcular a média dos tempos de execução do ambiente paralelizado e

compará-la com a execução do ambiente sequencial. O parâmetro de distância máxima utilizado foi novamente o igual a 3.

Para a parte da seleção do algoritmo de similaridade, foram utilizados parâmetro de parada baseados primeiramente no algoritmo que primeiro encontrasse uma quantidade de 50 duplicações durante a execução da meta heurística, seguido pela segunda condição de parada que levou em consideração a escolha do algoritmo que tivesse encontrado o maior número de duplicações ao se atingir o limite de tempo de 1.5 segundos. Os resultados obtidos são mostrados na Tabela 4.

Tabela 4 – Comparação de tempo de execução entre o ambiente sequencial e paralelizado

	Tempo de execução (s)	
Tuplas analisadas	**Ambiente sequencial**	**Ambiente paralelizado**
1000	2.493	0.093
2000	10.985	0.136
4000	56.687	0.156
8000	170.776	0.188
16000	611.650	0.213

Fonte: elaborada pelo autor

Analisando os resultados obtidos, pode-se concluir que o ambiente paralelizado apresenta desempenho muito superior em relação ao ambiente sequencial, em especial quando a quantidade de tuplas analisadas aumenta. O tempo de execução do ambiente sequencial passa de 2.493 segundos para 611.65 segundos da menor para a maior amostra, representando um aumento de mais de 245 vezes no tempo, enquanto que na versão paralelizada o tempo não passa de 0.213 segundos para a maior amostra. Também nota-se que o tempo de execução na ocasião com maior quantidade de tuplas analisadas no caso do ambiente sequencial é mais de 2850 vezes maior que o tempo de execução obtido utilizando-se o ambiente paralelizado.

4.4 Considerações finais

Neste capítulo foram apresentados os testes realizados a fim de validar experimentalmente o ambiente desenvolvido neste trabalho. Foram feitos testes de comparação de qualidade das funções de similaridade e também testes de desempenho.

Por meio dos testes de qualidade, foi possível mostrar que dados de contextos diferentes podem se adequar a certas funções de similaridades. Assim, o ambiente é capaz de se adequar a certo contexto, com a utilização de mais de uma função de similaridade.

Os testes de desempenho foram capazes de mostrar que o ambiente executado de forma paralelizada apresenta um desempenho superior ao ambiente usando algoritmos sequenciais, tornando assim possível o uso para o cenário Big Data.

5 Conclusão

O ambiente desenvolvido neste trabalho foi capaz de lidar com dados de diferentes contextos. Isso foi feito por meio de todo um processo de configuração inicial e pré-processamento, de modo que facilitasse a identificação das tuplas duplicadas na etapa devida.

A etapa de identificação de tuplas duplicadas foi implementada com a utilização de duas funções de similaridades distintas. Essas funções são capazes de se adequar ao contexto dos dados. O ambiente faz uma avalização dos dados de entrada, testando as funções por meio de um algoritmo genético, assim, a função mais adequada a lidar com os dados em questão é utilizada na deduplicação.

O ambiente desenvolvido também mostrou-se eficaz ao lidar com maiores quantidades de dados devido a sua implementação de forma paralelizada por meio da ferramenta Apache Spark. A implementação de forma paralela mostrou-se muito eficiente em termos de tempo de processamento.

Pode-se concluir, após o desenvolvimento deste trabalho que a utilização de múltiplas funções de similaridade ajuda na identificação de forma correta de tuplas duplicadas em uma base de dados. Isso é feito de forma automática e com possibilidade de se realizar um pré-processamento anterior, facilitando o processo e garantindo melhores resultados. Também pode-se concluir que recursos de paralelização acabam por tornar o desempenho do ambiente muito mais viável, de forma que ao se lidar com grandes quantidades de dados o tempo de processamento ocorre em um tempo viável, tornando a implementação possível no cenário Big Data.

5.1 Contribuição científica

Este trabalho possui como contribuição científica a implementação de um ambiente capaz de realizar a configuração e pré-processamento de dados, levando em consideração o contexto, para buscar a identificação de tuplas duplicadas em determinada amostra de dados de entrada.

A parte de identificação de duplicações ainda conta com uma seleção entre dois algoritmos de cálculo de similaridade. Essa seleção é feita por meio de um algoritmo genético que avalia qual das duas funções é mais adequada para o contexto dos dados de entrada atuais.

Além disso, a implementação do ambiente foi realizada de forma paralelizada, de modo que torne possível sua implementação no contexto Big Data, podendo lidar com grandes volumes de dados. A implementação paralelizada se mostrou muito superior em questão de tempo de execução se comparada à abordagem sequencial.

5.2 Atividades futuras

O ambiente se mostrou eficaz em questão de identificar duplicações e considerar o contexto dos dados de entrada, no entanto a implementação da combinação de mais funções de similaridade se mostra um possível aperfeiçoamento no ambiente, de modo a considerar a adequação em mais contextos de dados.

Assim, trabalhos futuros consistem no estudo e viabilidade de implementação de mais funções de similaridade em conjunto. Isso pode ser feito para ajudar a identificar as duplicações nas bases de dados de forma mais eficiente e correta. Além disso, pode ser possível considerar outros métodos de configuração do ambiente, de modo a incrementar o pré-processamento e possibilitar que a identificação das duplicações encontre menos erros.

Referências bibliográficas

ALAM, S.; YAO, N. Big data analytics, text mining and modern english language. **Journal of Grid Computing**, Dordrecht, v. 17, n. 2, p. 357-366, 04 ago. 2018.

ALENAZI, S. R.; AHMAD, K. Record duplication detection in database: a review. **International Journal on Advanced Science, Engineering and Information Technology**, Padang, v. 6, n. 6, p. 838-845, 2016.

ALI, A. H. A survey on vertical and horizontal scaling platforms for big data analytics. **International Journal of Integrated Engineering**, v. 11, n. 6, p. 138-150, 2019.

ALKANHAL, M. I.; AL-BADRASHINY, M.A.; ALGHAMDI, M.M.; AL-QABBANY, A. O. Automatic stochastic arabic spelling correction with emphasis on space insertions and deletions. **IEEE Transactions on Audio, Speech, and Language Processing**, v. 20, n. 7, p. 2111-2122, 2012.

ARFAT, Y.; USMAN, S.; MEHMOOD, R.; KATIB, I. Big data tools, technologies, and applications: A survey. In: **Smart Infrastructure and Applications**. Springer, Cham, 2020. p. 453-490.

BALHAF, K.; ALSMIRAT, M. A.; AL-AYYOUB, M.; JARARWEH, Y.; SHEHAB, M. A. Accelerating Levenshtein and Damerau edit distance algorithms using GPU with unified memory. In: **2017 8th international conference on information and communication systems (ICICS)**. IEEE, 2017. p. 7-11.

BARD, G. V. Spelling-error tolerant, order-independent pass-phrases via the Damerau-Levenshtein string-edit distance metric. In: **Proceedings of the fifth Australasian symposium on ACSW frontiers-Volume 68**. 2007. p. 117-124.

BHARATHI, B.; REDDY, C.; SUDARSANA. R. Duplicate record deletion in relational database management systems. **International Journal of Scientific & Engineering Research**, v. 8, n. 5, p. 266-271, maio 2017.

CHAN, K. P.; CHEN, T. Y.; TOWEY, D. Restricted random testing. In: **European Conference on Software Quality**. Springer, Berlin, Heidelberg, 2002. p. 321-330.

CHAUDHURI, S.; GANJAM, K.; GANTI, V.; MOTWANI, R. Robust and efficient fuzzy match for online data cleaning. In: **Proceedings of the 2003 ACM SIGMOD international conference on Management of data**. 2003. p. 313-324.

CHU, X.; ILYAS, I. F.; KRISHNAN, S.; WANG, J. Data cleaning: overview and emerging challenges. In: **Proceedings of the 2016 International Conference on Management of Data.** ACM, San Francisco, p. 2201-2206, 1 jul. 2016.

DAMERAU, F. J. A technique for computer detection and correction of spelling errors. **Communications of the ACM**, v. 7, n. 3, p. 171-176, 1964.

DAS, S.; MULLICK, S.S.; SUGANTHAN, P.N. Recent advances in differential evolution–an updated survey. **Swarm and Evolutionary Computation**, v. 27, p. 1-30, 2016.

DHIVYABHARATHI, G. V.; KUMARESAN, S. A. Survey on duplicate record detection in real world data. *In*: **2016 3rd International Conference on Advanced Computing and Communication Systems.** IEEE, Coimbatore, p. 1-5, 22 jan. 2015.

DRAISBACH, U.; NAUMANN, F.; SZOTT, S.; WONNEBERG, O. Adaptative windows for duplicate detection. In: **2012 IEEE 28th International Conference on Data Engineering**. IEEE, Washington, p. 1073-1083, 1 abr. 2012.

GANTI, V.; SARMA, A. D. Data cleaning: A practical perspective. **Synthesis Lectures on Data Management**, v. 5, n. 3, p. 1-85, 2013.

GASTONI, S.; GRANELLI, G. P.; MONTAGNA, M.. Multiple bad data processing by genetic algorithms. In: **2003 IEEE Bologna Power Tech Conference Proceedings,**. IEEE, 2003. p. 6 pp. Vol. 1.

HILDEBRANDT, K.; PANSE, F.; WILCKE, N.; RITTER, N. Large-scale data pollution with Apache Spark. **IEEE Transactions on Big Data**, 2017.

ILYAS, I. F.; CHU, X. **Data cleaning**. ACM, 2019.

JIVANI, A. G. A comparative study of stemming algorithms. **Int. J. Comp. Tech. Appl**, v. 2, n. 6, p. 1930-1938, 2011.

JANSSEN, M.; VAN DER VOORT, H.; WAHYUDI, A. Factors influencing big data decision-making quality. **Journal of Business Research**. Elsevier Inc., v. 70, p. 338-345, 9 ago. 2017.

KÖPCKE, H.; RAHM, E. Frameworks for entity matching: A comparison. **Data & Knowledge Engineering**, v. 69, n. 2, p. 197-210, 2010.

LEONARDO, B.; HANSUN, S. Text documents plagiarism detection using Rabin-Karp and Jaro-Winkler distance algorithms. **Indonesian Journal of Electrical Engineering and Computer Science**, v. 5, n. 2, p. 462-471, 2017.

LI, Y.; ALBATHAN, M.; SHEN, Y.; BIJAKSANA, M. A. Relevance feature discovery for text mining. **IEEE Transactions on Knowledge and Data Engineering**. IEEE, v. 27, n. 6, p. 1656-1669, 1 jun. 2015.

MESTRE, D. G.; PIRES, C. E. S.; NASCIMENTO, D. C.; DE QUEIROZ, R. M. An efficient spark-based adaptive windowing for entity matching. **Journal of Systems and Software**. Elsevier, v. 128, p. 1-10, 15 jul. 2017.

MORAL, C.; DE ANTONIO, A.; IMBERT, R.; RAMÍREZ, J. A survey of stemming algorithms in information retrieval. **Information Research: An International Electronic Journal**, v. 19, n. 1, p. 1-16, mar. 2014.

POL, U. R. Big data analysis: comparison of hadoop MapReduce and apache spark. **International Journal of Engineering Science**. IJESC, Kolhapur, v. 6, n. 6, p. 6389- 6391, jun. 2016.

PRASETYA, D. D.; WIBAWA, A. P.; HIRASHIMA, T. The performance of text similarity algorithms. **International Journal of Advances in Intelligent Informatics**, v. 4, n. 1, p. 63-69, 2018.

RAMYA, R. S.; VENUGOPAL, K. R.; IYENGAR S. S.; PATNAIK, L. M. Feature extraction and duplicate detection for text mining: a survey. **Global Journal of Computer Science and Technology**, v.16, n. 5, p. 1-21, Jan. 2017.

REZAEIAN, M.; MONTAZERI, H.; LOONEN, R. C. G. M. Science foresight using life-cycle analysis, text mining and clustering: A case study on natural ventilation. **Technological Forecasting and Social Change**. Elsevier, v. 118, p. 270-280, 21 fev. 2017.

RISTOSKI, P.; PAULHEIM, H; Semantic Web in data mining and knowledge discovery: A comprehensive survey. **Web semantics: science, services and agents on the World Wide Web**, v. 36, p. 1-22, 2016.

SHARMA, C.; SABHARWAL, S.; SIBAL, Ritu. A survey on software testing techniques using genetic algorithm. **arXiv preprint arXiv:1411.1154**, 2014.

SINGH, D.; REDDY, C. K. A survey on platforms for big data analytics. **Journal of big data**, v. 2, n. 1, p. 8, 2015.

STEORTS, R. C.; HALL, R.; FIENBERG, S. E. A Bayesian approach to graphical record linkage and deduplication. **Journal of the American Statistical Association**, v. 111, n. 516, p. 1660-1672, 12 jan 2017.

VIJAYARANI, S.; ILAMATHI, M. J.; NITHYA, M. Preprocessing techniques for text mining-an overview. **International Journal of Computer Science & Communication Networks**, v. 5, n. 1, p. 7-16, 2015.

WANDHEKAR, V.; MOHANPURKAR, A. Validation of Deduplication in Data using Similarity Measure. **International Journal of Computer Applications**, v. 116, n. 21, 2015.

WANG, J.; KRISHNAN, S.; FRANKLIN, M. J.; GOLDBERG, K.; KRASKA, T.; MILO, T. A sample-and-clean framework for fast and accurate query processing on dirty data. In: **Proceedings of the 2014 ACM SIGMOD international conference on Management of data**. 2014. p. 469-480.

WANG, Y.; QIN, J.; WANG, W. Efficient approximate entity matching using jaro-winkler distance. In: **International Conference on Web Information Systems Engineering**. Springer, Cham, 2017. p. 231-239.

WAYKOLE, J. R.; SHINDE, S. M. A Survey Paper on Deduplication by using Genetic Algorithm Alongwith Hash-Based Algorithm. **International Journal of Engineering Research and Applications**, v. 4, n. 1, 2014.

YU, M.; LI, G.; DENG, D.; FENG, J. String similarity search and join: a survey. **Frontiers of Computer Science**, v. 10, n. 3, p. 399-417, 24 nov. 2016.