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

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

MARIANA FRANCELINO

COMBINAÇÃO DE SUCESSIVAS FUNÇÕES DE SIMILARIDADE PARA OTIMIZAÇÃO DE AMBIENTE PARA IDENTIFICAÇÃO DE TUPLAS DUPLICADAS COM RECURSOS DE PARALELIZAÇÃO

São José do Rio Preto
2023

MARIANA FRANCELINO

COMBINAÇÃO DE SUCESSIVAS FUNÇÕES DE SIMILARIDADE PARA OTIMIZAÇÃO DE AMBIENTE PARA IDENTIFICAÇÃO DE TUPLAS DUPLICADAS COM RECURSOS DE PARALELIZAÇÃO

Trabalho de Conclusão de Curso (TCC) apresentado como parte dos requisitos para obtenção do título de Bacharel em Ciência da Computação, junto ao Conselho de Curso de Bacharelado em Ciência da Computação, do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", Campus de São José do Rio Preto.

Orientador: Prof. Dr. Carlos Roberto Valêncio

São José do Rio Preto
25 de janeiro de 2023

F815c

Francelino, Mariana

Combinação de sucessivas funções de similaridade para otimização de ambiente para identificação de tuplas duplicadas com recursos de paralelização / Mariana Francelino. -- São José do Rio Preto, 2022

53 p. : 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. Mineração de dados (Computação). 2. Algoritmos paralelos. 3. Big Data. 4. Processamento de textos (Computação). 5. Banco de dados. 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).

Essa ficha não pode ser modificada.

MARIANA FRANCELINO

COMBINAÇÃO DE SUCESSIVAS FUNÇÕES DE SIMILARIDADE PARA OTIMIZAÇÃO DE AMBIENTE PARA IDENTIFICAÇÃO DE TUPLAS DUPLICADAS COM RECURSOS DE PARALELIZAÇÃO

Trabalho de Conclusão de Curso (TCC) apresentado como parte dos requisitos para obtenção do título de Bacharel em Ciência da Computação, junto ao Conselho de Curso de Bacharelado em Ciência da Computação, do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", Campus de São José do Rio Preto.

Comissão Examinadora:

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

Profª. Drª. Carina Alexandra Rondini
UNESP – São José do Rio Preto

Prof. Dr. Leandro Alves Neves
UNESP – São José do Rio Preto

São José do Rio Preto
25 de janeiro de 2023

Resumo

Nos últimos anos o problema de identificação de tuplas duplicadas tem se tornado cada vez mais expressivo com o crescimento do cenário Big Data. Empresas, governos, meio acadêmico e indústria geram cada vez mais informações, e tem-se como consequência a dificuldade de extrair conhecimentos valiosos dessas bases, nas quais muitas das vezes são desorganizadas e heterogêneas. Quando usuários, independente da organização, inserem dados em uma base de dados, é comum que eles contenham erros ou inconsistências, onde tais cenários impedem que a etapa de análise de dados gere resultados satisfatórios. Sendo assim, é imprescindível que estes dados sejam limpos, normalizados, e posteriormente tratados. Desta forma, pretendeu-se implementar um ambiente de identificação de tuplas duplicadas com recursos de paralelização e processamento em memória com Apache Spark. Para isso, inicialmente foi realizada uma etapa de limpeza de dados e, posteriormente, foram executados 3 algoritmos de similaridade, Damerau-Levenshtein, Jaro-Winkler e Needleman-Wunsch, com auxílio da paralelização. O intuito foi identificar tuplas duplicadas, aumentar o desempenho dessas técnicas e torná-las funcionais, uma vez que elas possuíam um alto nível de custo computacional. Ao mesmo tempo, foram testadas diversas combinações dessas funções para otimizar os cálculos de similaridade. A análise dos resultados obtidos após a execução do ambiente proposto mostrou que é possível aumentar a precisão dos resultados ao utilizar mais de uma função de similaridade. Além disso, as técnicas de paralelização e processamento em memória melhoraram o desempenho do algoritmo, e o tornam mais eficiente e rápido. A análise de qualidade dos dados também mostrou resultados positivos, sendo possível identificar uma quantidade significativa de tuplas duplicadas e as funções de similaridade utilizadas foram capazes de tratar contextos diferentes e detectar duplicações. Deste modo, propôs-se como contribuição científica a aplicação de diversas funções de similaridade a fim de otimizar os ambientes de identificação de tuplas duplicadas, com recursos de paralelização.

Palavras-chave: Detecção de duplicatas. Big Data. Pré-processamento de dados. Contexto dos dados. Processamento paralelo.

Abstract

In recent years, the problem of identifying duplicate tuples has become increasingly expressive with the growth of the Big Data scenario. Companies, governments, the academic community, and industry generate more and more information, leading to difficulties in extracting valuable knowledge from these databases, which are often disorganized and heterogeneous. When users, regardless of the organization, insert data into a database, it is common for them to contain errors or inconsistencies, which prevent the data analysis stage from generating satisfactory results. Therefore, it is essential that these data be cleaned, normalized, and subsequently treated. With this in mind, we aimed to implement an environment for identifying duplicate tuples with parallelization and in-memory processing capabilities using Apache Spark. To this end, a data cleaning step was initially performed, and then three similarity algorithms, Damerau-Levenshtein, Jaro-Winkler, and Needleman-Wunsch, were executed with the help of parallelization. The goal was to identify duplicate tuples, increase the performance of these techniques, and make them functional, since they had a high level of computational cost. At the same time, various combinations of these functions were tested to optimize the similarity calculations. The analysis of the results obtained after the execution of the proposed environment showed that it is possible to increase the accuracy of the results by using more than one similarity function. In addition, the parallelization and in-memory processing techniques improved the performance of the algorithm, making it more efficient and faster. The data quality analysis also showed positive results, it was possible to identify a significant amount of duplicate tuples, and the similarity functions used were able to handle different contexts and detect duplicates. Thus, it was proposed as a scientific contribution the application of various similarity functions in order to optimize the environments for identifying duplicate tuples with parallelization capabilities.

Keywords: Duplicate detection. Big data. Data preprocessing. Data context. Parallel processing.

Lista de Ilustrações

Lista de Tabelas

Lista de Abreviaturas e Siglas

GBD	Grupo de Banco de Dados
KDD	*Knowledge Discovery in Databases*
RDD	*Resilient Distributed Dataset*
SIVAT	Sistemas de Informação e Vigilância de Acidentes de Trabalho

Sumário

1 Introdução

Nas últimas décadas, o cenário de Big Data ganhou destaque na economia mundial. Muitos setores da indústria possuem sistemas que dependem da precisão dos dados para realizar as tarefas do cotidiano com eficiência. Logo, a qualidade dos dados armazenados interfere diretamente no custo e na tomada de decisão dos negócios (ELMAGARMID; IPEIROTIS; VERYKIOS, 2007). É comum encontrar problemas nas bases de dados relacionados à inconsistência de dados, como erros de digitação, tuplas duplicadas ou valores nulos. Para solucionar esses problemas, o processo de extração de conhecimento em bases de dados (KDD, do inglês Knowledge Discovery in Databases) realiza a etapa de limpeza de dados, mais conhecida pelo termo em inglês Data Cleaning, no qual prepara os dados de forma que garanta a integridade destes para as próximas etapas do KDD (MITRA; ACHARYA, 2003).

De acordo com Andrade et al. (2011), a limpeza de dados é uma das etapas do pré-processamento do KDD que tem como objetivo utilizar técnicas computacionais para tratar erros, inconsistências e duplicidades presentes nos dados. Existem diversas soluções para tentar resolver dados duplicados e dados inconsistentes, entretanto, não é possível eleger a melhor metodologia ou ferramenta, uma vez que os resultados gerados após o pré-processamento nem sempre são totalmente tratados, e o custo do processamento dessas aplicações também costuma ser bastante elevado. (ELMAGARMID; IPEIROTIS; VERYLIOS, 2007).

Sendo assim, é importante investir em soluções computacionais que tratam diversos cenários ainda não explorados, porém relevantes para contribuir para uma boa consistência dos dados armazenados. Desse modo, o investimento em tecnologias para limpeza de dados em Big Data tem aumentado cada vez mais (SILVEIRA; MARCOLIN; FREITAS, 2015) e tem um impacto positivo nas demais etapas do processo de Descoberta de Conhecimento em Dados . Como apontado por ELMAGARMID; IPEIROTIS; VERYKIOS (2007) em sua revisão sobre detecção de registros duplicados, a limpeza de dados é uma das etapas

fundamentais no processo de KDD. Outros autores também destacam a importância da limpeza de dados no processo de mineração de dados, como por exemplo MITRA; ACHARYA (2003) e CHU; ILYAS; KRISHNAN; WANG (2016).

Para realizar essa limpeza, existem diversos algoritmos e técnicas disponíveis, como a remoção de duplicatas baseada em algoritmos de similaridade fonética (ANDRADE; SOUZA; BABINI; VALENCIO, 2011), janelas adaptativas para detecção de duplicatas (DRAISBACH; NAUMANN; SZOTT; WONNEBERG, 2012), entre outros. Alguns trabalhos também estudam limpeza de dados em contextos específicos, como limpeza de dados em bancos de dados relacionais (BHARATHI; REDDY; SUDARSANA, 2017) e limpeza de dados em dados web (AYE, 2011) e textuais (SETTE; MARTINS, 2016). Diversas pesquisas também foram realizadas para comparar a performance de diferentes algoritmos de limpeza de dados, tais como (PRASETYA; PRASETYA WIBAWA; HIRASHIMA, 2018) e (VIJAYARANI; ILAMATHI; NITHYA, 2015).

1.1 Motivação e escopo

Em muitas aplicações os relacionamentos nos bancos de dados são associados a dados duplicados. Localizar e, em seguida, fazer a remoção das tuplas duplicadas é um dos processos mais importantes do KDD. Erros de digitação, inserção errada de dados e inúmeras representações da mesma entidade no mundo real geram duplicações na maneira de representar tais dados. (BHARATHI et al., 2017). O desafio deste problema é identificar de maneira eficiente as tuplas duplicadas que no mundo real se trata do mesmo objeto e com isso extrair as informações úteis dos dados. (DRAISBACH et al., 2011). Existem vários problemas que envolvem registros duplicados. Por exemplo, em uma base de dados de um supermercado, há a possibilidade de os níveis de estoque estarem incorretos, as classificações de crédito podem conter erros de cálculo, o número de clientes pode ser contado inúmeras vezes, entre outros.

Dessa forma, é preciso otimizar a maneira que é feita a detecção dessas tuplas, senão o tempo de busca pode ser muito elevado, e como resultado, alguns registros duplicados não são identificados e, no pior dos casos, até perdidos. Este processo é custoso, pois as representações podem divergir, portanto uma medida de similaridade deve ser definida para comparar esses valores. Também tem o cenário de que os dados podem ter um volume alto, isso torna a comparação em pares de todos os dados do banco de dados inviável (DRAISBACH et al., 2011).

1.2 Objetivos

Este trabalho teve como objetivo principal garantir a qualidade dos dados extraídos de uma base de dados por meio da remoção de tuplas duplicadas. Para alcançar esse objetivo, foi implementado um ambiente capaz de identificar a duplicação de tuplas de forma precisa e eficiente, leva-se em conta o contexto inserido pelo usuário e utiliza-se técnicas de pré-processamento e paralelização para lidar com grandes volumes de dados. Além disso, foram aplicadas as funções de similaridade Damerau-Levenshtein, Jaro-Winkler e Needleman-Wunsch em sequência para otimizar o algoritmo de Data Clean e encontrar tuplas de forma mais precisa e confiável, para reduzir o tempo de processamento.

Desta forma, a contribuição científica deste trabalho foi a utilização de recursos de paralelização em conjunto com as funções de similaridade Damerau-Levenshtein, Jaro-Winkler e Needleman-Wunsch em sequência para otimizar o ambiente de identificação de tuplas duplicadas a fim de aperfeiçoar as técnicas de similaridade, aumentar a precisão dos dados gerados e reduzir o tempo de processamento para cálculo de similaridade entre os pares de dados.

.

1.3 Metodologia

Esse trabalho adotou uma metodologia baseada nas seguintes etapas: revisão bibliográfica, desenvolvimento, testes e conclusão. A revisão bibliográfica consistiu em realizar um levantamento dos trabalhos existentes sobre os temas relacionados ao projeto, como Big Data, Limpeza de Dados, identificação de tuplas duplicadas, funções de similaridade e paralelização e processamento em memória. O objetivo desta etapa foi entender o estado da arte dos trabalhos existentes sobre esses assuntos.

Após o levantamento bibliográfico, foi elaborado um ambiente capaz de realizar a identificação das tuplas duplicadas com a combinação de sucessivas funções de similaridade, onde o objetivo foi otimizar tal ambiente. Isso foi feito baseado nas informações teóricas adquiridas na revisão bibliográfica.

Após a implementação do ambiente, foram realizados testes com a base de dados SIVAT, extraída do sistema Kaggle. Foram testadas as funções de similaridade, Damerau-Levenshtein, Jaro-Winkler e Needleman-Wunsch individualmente e em sequência, para

avaliar a qualidade e desempenho do sistema. Além disso, o ambiente foi configurado para processamento em paralelo e em memória.

Em seguida, foram comparados os resultados obtidos com os trabalhos anteriores, e foi possível avaliar a validade dos resultados obtidos. Finalmente, foi elaborada a conclusão, onde observou-se que, apesar de existir um aumento no tempo de processamento computacional ao utilizar a combinação sucessiva de funções de similaridade, é possível aumentar a precisão na identificação de tuplas duplicadas.

1.4 Organização da monografia

Este trabalho apresentou na primeira seção o contexto atual de identificação de tuplas duplicadas no contexto do cenário atual Big Data. Foi exposto a motivação e objetivo do trabalho, bem como as ferramentas disponíveis para sua execução. As demais seções estão dispostas da seguinte maneira:

1. Seção 2 – Fundamentações teóricas referentes ao estado da arte do escopo do trabalho, inclui Big Data, limpeza de dados, tuplas duplicadas, pré-processamento, combinação sucessiva de funções de similaridade, paralelização e processamento em memória.
2. Seção 3 – Desenvolvimento do projeto que realizou o pré-processamento dos dados, em seguida foi desenvolvido o ambiente para identificação de tuplas duplicadas com a combinação de sucessivas funções de similaridade e recursos de paralelização
3. Seção 4 – Análise dos resultados obtidos a partir dos testes realizados no ambiente construído. Essa seção apresentou as métricas utilizadas para avaliar a precisão da identificação de tuplas duplicadas, bem como a comparação entre os resultados obtidos com o uso de uma função de similaridade e com o uso de múltiplas funções de similaridade em sequência. Além disso, foi feita uma análise do tempo de processamento computacional e do uso de recursos de paralelização.
4. Seção 5 – Conclusão e Contribuição científica, onde foram apresentadas as principais conclusões obtidas com o trabalho, que incluem as contribuições

científicas e técnicas. Essa seção discute as atividades futuras de pesquisa e desenvolvimento para o aperfeiçoamento do ambiente construído.

2 Fundamentação teórica

Nesta seção é apresentada a etapa de fundamentação teórica e trabalhos correlatos que foram encontrados na literatura científica. Dessa forma, são abordados temas sobre identificação de tuplas duplicadas não idênticas, limpeza de dados, cálculo de similaridade, técnicas de escalabilidade em um contexto de Big Data e recursos de paralelização com Apache Spark.

2.1 Limpeza de dados

Corrigir dados inconsistentes é um dos desafios para uma análise de dados eficiente e não detectar esses erros resulta em análises imprecisas e prejudica as decisões de negócios. Na última década houve interesse da indústria e da área acadêmica em solucionar problemas que envolvem limpeza de dados, sejam eles técnicas estatísticas, escalabilidade, novas interfaces e novas abstrações (CHU et al., 2016).

O volume de dados coletados no mundo cresce exponencialmente, o que torna a extração de dados não adequada para o cenário Big Data. O processo de limpeza de dados envolve principalmente fazer a identificação de erros (como tuplas duplicadas) e corrigi-los. Tal método tem uma complexidade alta e demora para garantir dados que tenham uma boa qualidade (RIDZUAN; WAN ZAINON, 2019), por este motivo é necessário desenvolver

novas técnicas computacionais para limpar esses registros e explorar cenários que ainda não possuem soluções eficientes.

2.2 Tuplas duplicadas

Ao inserir dados do mundo real em uma base de dados, os registros podem ter mais de uma representação. Os erros de correspondência podem ser descritos como resultado de erros de digitação, falta de padronização nos registros, informações incompletas ou maneiras diferentes de descrever a mesma entidade do mundo real. Em uma base de dados que não contém erros, e em termos relacionais, é possível unir tabelas por meio de seus campos-chave. Entretanto, os dados não possuem um único identificador global que permita essa operação (ELMAGARMID; IPEIROTIS; VERYLIOS, 2007).

Além deste cenário, não existe um controle para as entradas de dados, ou seja, é permitido qualquer tipo de entrada em diferentes fontes de dados. Dessa forma, a qualidade dos dados é comprometida (por exemplo, ao inserir a string Microsft no lugar de de Microsoft), restrições de integridade podem ocorrer (por exemplo, permite EmployeeAge ¼ 567) e várias convenções para registrar informações acontecem (por exemplo, 44 W. 4th St. versus 44 West Fourth Street). É possível também que em bancos de dados a estrutura, semântica e suposições subjacentes nos registros também podem diferir (ELMAGARMID; IPEIROTIS; VERYLIOS, 2007).

Solucionar este impasse pode gerar um problema de alta complexidade. Porém, há a possibilidade de solucionar este caso com algoritmos de identificação de registros duplicados para que posteriormente seja possível remover as entidades que representam o mesmo objeto no mundo real, o que reduz a quantidade de tuplas duplicadas e aumenta a qualidade da análise desses dados. Um exemplo de um caso é mostrado na Tabela 1, onde todos os cenários representam o mesmo dado, que se trata do remédio popularmente conhecido como Aspirina. Este tem diversas formas de ser nomeado, e também podem ser cadastrados em categorias diferentes.

Dessa forma, as etapas mais importantes são:

1. Tokenização.
2. Remoção de palavras de parada.

3. Redução de palavras flexionadas.

Estas etapas garantem uma maior qualidade na análise dos dados, pois eliminam palavras e cadeias de caracteres que não são relevantes para o contexto. Portanto, os dados devem ser tratados na etapa de pré-processamento antes que seja possível aplicar técnicas de data mining. (AYE, 2011; VIJAYARANI; ILAMATHI; NITHYA, 2015).

Tabela 1— Exemplo de Registros Duplicados não idênticos

Identificador	nome_remedio	tipo_remedio
1	aspirina	comprimido
2	ácido acetilsalicílico	anti-inflamatório
3	AAS (C9H8O4)	analgésico
4	aspirina	analgésico

Fonte: Elaborado pelo autor.

2.3 Pré-processamento em dados textuais

2.3.1 Tokenização

A tokenização é a etapa que realiza uma análise textual que divide a sequência de caracteres em palavras, frases, palavras-chave ou outros elementos de interesse. O objetivo principal dela é a identificação de palavras significativas. Para realizar esse processo, é feita a conversão da frase em uma árvore, e assim, é possível remover palavras genéricas que não são interessantes para gerar informações significativas, como verbos, advérbios, caracteres especiais como ‘@’, ‘&’, ‘%’ e palavras irrelevantes para o contexto. Existem diversas funções de processamento de linguagem, e algumas fases desse processo são: tradução, verificação ortográfica e detecção de limites de frases, recuperação de informações e extração de informações, entre outros. (REZAEIAN; MONTAZERI; LOONEN, 2017).

Ao processarmos dados textuais, é importante identificar as palavras presentes em cada registro. Isso permite representar cada registro por uma lista de palavras, e assim facilita o processamento. Além disso, remover caracteres inválidos e espaços em branco ajuda a

reduzir consideravelmente a quantidade de informações a serem tratadas nas etapas subsequentes (ALENAZI; AHMAD, 2016.)

2.3.2 Remoção de palavras de parada

As palavras que não são interessantes para gerar uma análise de qualidade são frequentes em todos os tipos de fonte de dados, sendo assim necessário filtrar esses dados irrelevantes e ruidosos para melhorar a qualidade de dados extraída. Palavras sem significado contribuirão para a falha nos resultados, então, filtrá-las é processo utilizado para a remoção de “palavras de parada”, no qual é um assunto contido na linguagem natural (ALAM; YAO, 2018). A razão pela qual as palavras de parada precisam ser removidas é que a existência delas faz com que o texto fique mais pesado e se torne mais difícil de ser analisado por analistas. O mais recorrente são as preposições, artigos e pronomes, o que não contribuem para uma extração com significado. Alguns exemplos para palavras de parada são: “o”, “na”, “a”, “com”, etc. Tais informações são removidas dos documentos porque elas não são medidas como palavras-chave em aplicações de mineração de texto (VIJAYARANI; ILAMATHI; NITHYA, 2015).

2.3.3 Radicalização de palavras

Reduzir palavras flexionadas significa remover afixos, plurais e verbos, porém, ao remover essas partes, o sentido da palavra é mantido. Isso implica em uma diminuição do volume de dados, e diminui a complexidade do texto analisado sem ocorrer a perda do contexto deste (JIVANI, 2011). Vários algoritmos foram propostos para a finalidade de radicalizar as sentenças, e eles se baseiam em regras ou probabilidades estatísticas. O primeiro caso se baseia em uma lista definida pelo usuário de sufixos e prefixos de acordo com o idioma do texto analisado. E o segundo tem como suporte as análises estatísticas para a retirada dos radicais irrelevantes para o contexto. A primeira abordagem que envolve o usuário inserir as regras do contexto do idioma é mais eficiente. (MORAL et al., 2014).

2.4 Similaridade entre dados textuais

2.4.1 Funções de similaridade baseadas em strings

A abordagem de funções baseadas em strings é a maneira de identificar registros duplicados mais simples. Existem duas principais subdivisões de funções de similaridade baseadas em strings: caracteres e tokens. A função de similaridade baseada em caracteres utiliza duas sequências de strings como entrada e, após isso, quantifica a semelhança dos caracteres, que como resultado apresenta o mínimo de operações que uma string precisa para se transformar na outra. Portanto, duas strings são parecidas se o número de operações é menor do que o limite definido (GOMAA; FAHMY, 2013). Podemos citar como exemplo alguns exemplos: distância de Levenshtein, Damerau-Levenshtein, Needleman-Wunsch, Smith-Waterman, JaroWinkler.

A abordagem de edição baseada em strings é muito utilizada para lidar com erros e dados inconsistentes. O método baseado em tokens é conhecido dessa forma pois divide a string em um conjunto de tokens. A similaridade nesse caso é feita com a quebra das strings em substrings e, em seguida, manipula este conjunto de tokens como palavras. A principal ideia dessa abordagem é comparar duas strings com base em seus tokens e identificar se é duplicada ou não. Pode-se citar como exemplo dessa abordagem a Semelhança de Jaccard, coeficiente de Dice, Similaridade de cosseno, à distância de Manhattan e à distância euclidiana (PRASETYA; PRASETYA WIBAWA; HIRASHIMA, 2018). O resumo dos temas que foram abordados acima está representado na Figura 1. As funções de similaridade baseadas em strings se dividem em baseadas em caracteres e baseadas em tokens e cada uma dessas tem seus algoritmos de aplicação.

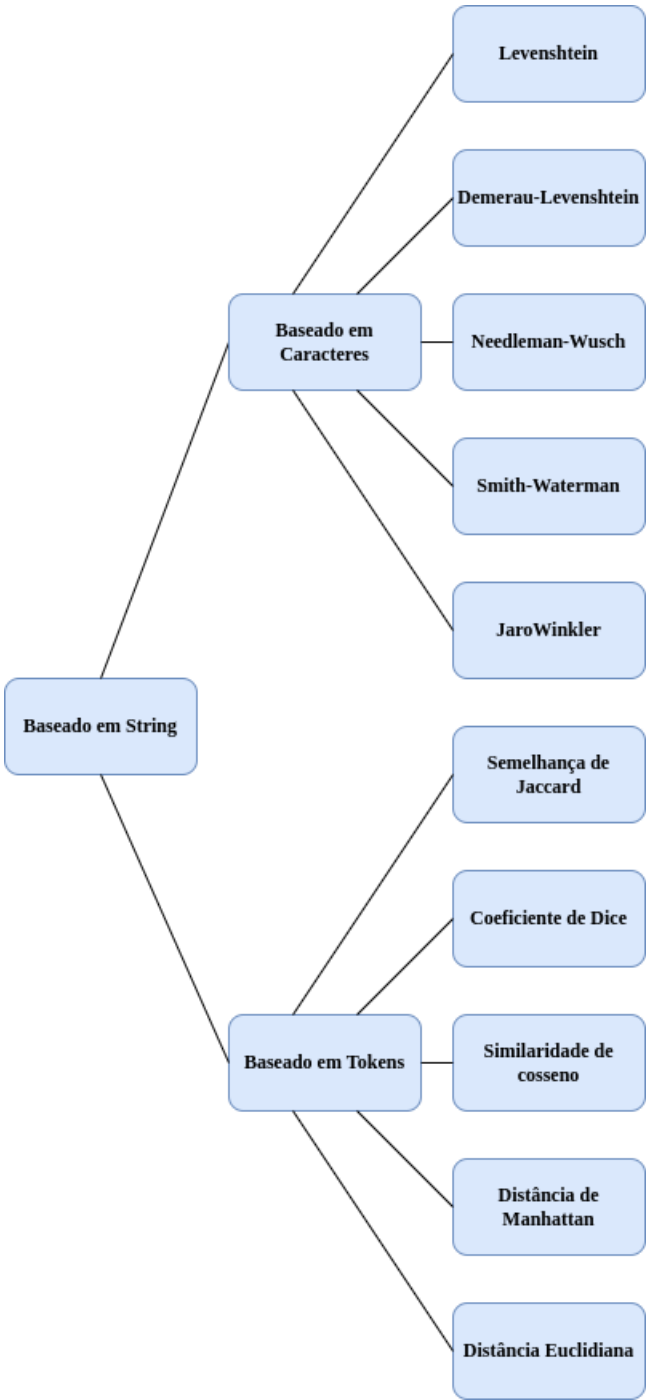


Figura 1 – Divisão de tipos de similaridade baseado em strings

Fonte: Adaptada de (PRASETYA; PRASETYA WIBAWA; HIRASHIMA, 2018)

2.4.2 Funções de similaridade baseadas em *corpus*

Ao comparar dados textuais, é importante levar em conta o significado das palavras e não apenas da string em si. Uma forma de fazer isso é utilizar funções de similaridade semântica, que podem ser classificadas em dois tipos: baseadas em corpus ou baseadas em conhecimento (ELMAGARMID; IPEIROTIS; VERYKIOS, 2007). As funções baseadas em corpus comparam a semelhança entre strings com base em sua presença em um conjunto de dados, enquanto as funções baseadas em conhecimento utilizam ontologias ou outras representações do conhecimento para comparar a semelhança entre os dados. (MITRA; ACHARYA, 2003) acentua que essa abordagem possui vantagens e desvantagens, e os desenvolvedores devem escolher as funções de similaridade de acordo com as necessidades do projeto.

A abordagem de similaridade baseada em corpus utiliza uma técnica semântica para comparar dados textuais. Ela extrai informações de grandes corpora e determina a semelhança entre as strings. Como resultado, fornece a tradução. Alguns exemplos de funções de similaridade baseadas em corpus incluem o Hyperspace Analogue to Language, Latent Semantic Analysis, Explicit Semantic Analysis, Pointwise Mutual Information, Google's normalized distance e DISCO (DRAISBACH; NAUMANN; SZOTT; WONNEBERG, 2012). Essas técnicas podem ser aplicadas para a limpeza de dados e detecção de tuplas duplicadas.

.

2.4.3 Funções de similaridade baseadas em conhecimento

A similaridade baseada em conhecimento é uma medida de similaridade semântica que serve para comparar palavras baseadas em uma grande corpora. Funções de similaridade baseadas em conhecimento, ou com base na semântica, utilizam uma rede semântica para verificar o grau de similaridade entre as strings comparadas, essa medida é chamada de medida de similaridade baseada em conhecimento. Tem-se uma divisão na similaridade baseada em conhecimento, que é a semelhança com base em parentesco e a semelhança com base na semântica. A similaridade semântica usa uma representação clara de conhecimento, por exemplo, a interconexão de fatos, significados das palavras. Alguns exemplos são WordNet, SENSUS1, Cyc2, UMLS3, SNOMED4 (GOMAA; FAHMY, 2013). Uma ilustração das funções de similaridade semânticas, corpus e conhecimento, estão exibidas na Figura 2. As funções de similaridade baseadas em semântica se dividem em baseadas em corpus e conhecimento, onde cada uma destas tem seus algoritmos de aplicação.

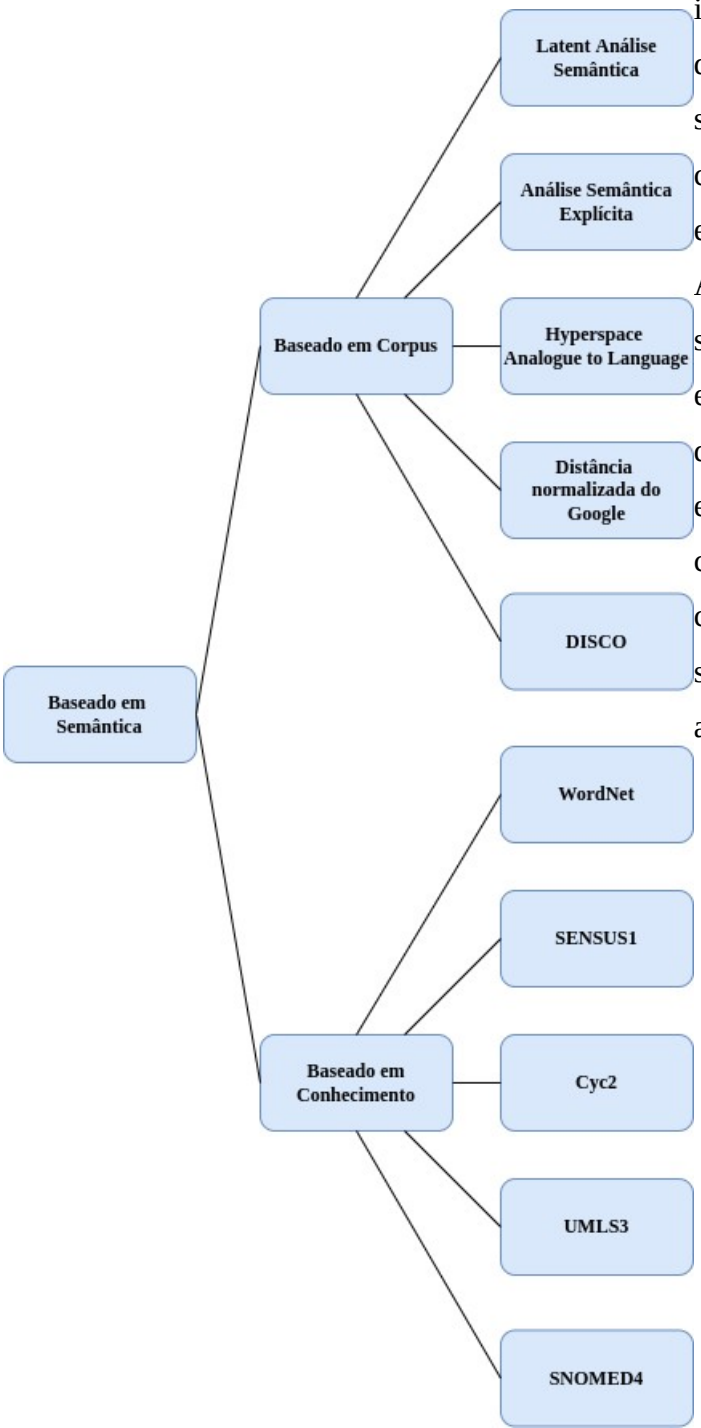


Figura 2 – Divisão de tipos de similaridade baseados em semântica.

Fonte: Adaptada de (PRASETYA; PRASETYA WIBAWA; HIRASHIMA, 2018)

2.4.4 Funções de similaridade híbridas

As categorias descritas nas seções anteriores são as principais contidas na literatura, entretanto, no presente há várias outras medidas de similaridade que não se encaixam em nenhuma das categorias anteriores. As funções de similaridade híbridas combinam conceitos das similaridades baseadas em strings, corpus e conhecimento para atingir um melhor resultado, onde explora-se então as vantagens de cada uma das diferentes abordagens. Exemplos das métricas híbridas são o método Level2, SoftTFIDF e similaridade de edição generalizada (GOMAA; FAHMY, 2013).

2.4.5 Algoritmo de Damerau-Levenshtein

O Algoritmo de Damerau-Levenshtein utiliza a distância entre duas sequências de caracteres para calcular a semelhança entre tuplas. É amplamente utilizado em aplicações de biologia molecular, onde é importante medir a similaridade entre sequências de DNA ou proteínas. A função do algoritmo é calcular a distância entre duas sequências de caracteres, baseado no algoritmo de Levenshtein e adicionando a capacidade de lidar com transposições de caracteres adjacentes. A implementação pode ser feita utilizando uma tabela de distância e atualizando as distâncias em cada iteração. Além de sua aplicação em biologia molecular, o algoritmo também é utilizado em aplicações de processamento de texto, como correção automática de palavras e reconhecimento de voz (Zhao; Sahni; 2017).

A implementação do Algoritmo de Damerau-Levenshtein é baseada na construção de uma tabela de distância entre as duas sequências a serem comparadas. Cada célula da tabela representa a distância entre uma subsequência da primeira sequência e uma subsequência da segunda sequência. A primeira linha e coluna da tabela representam subsequências vazias, e as células subsequentes são preenchidas com base nas regras de edição: inserção, exclusão, substituição e transposição. A transposição é a adição importante que o Algoritmo de Damerau-Levenshtein traz em relação ao algoritmo de Levenshtein. Ela permite que duas letras adjacentes sejam trocadas sem aumentar a distância entre as duas sequências (Zhao; Sahni; 2017).

2.4.6 Algoritmo de Jaro-Winkler

O algoritmo de Jaro-Winkler é uma medida de similaridade entre duas strings, utilizada para a identificação de correspondência e correção de erros de digitação. É uma combinação do algoritmo Jaro com a adição de uma penalização para a distância entre as strings iniciais. O algoritmo é amplamente utilizado em aplicações de processamento de linguagem natural, como correção de ortografia e agrupamento de nomes. (Wang, Y., Qin, J., Wang, W., 2017).

A implementação do algoritmo de Jaro-Winkler começa comparando as strings letra por letra e contando o número de caracteres iguais e a distância entre eles. Em seguida, é calculada a distância média entre as strings e é aplicado o fator de penalização à distância inicial das strings. O resultado é uma medida de similaridade entre 0 e 1, onde valores mais próximos de 1 indicam maior similaridade entre as strings. É importante destacar que o algoritmo é sensível à ordem dos caracteres e aos espaços entre eles, o que o torna útil para

aplicações que requerem precisão na identificação de correspondências (Wang, Y., Qin, J., Wang, W., 2017).

2.4.7 Algoritmo de Needleman-Wunsch

O algoritmo de Needleman-Wunsch é um algoritmo de alinhamento global de sequências biológicas. Ele foi desenvolvido para comparar duas sequências de DNA ou proteínas e determinar as áreas de semelhança e diferença entre elas. O algoritmo funciona comparando as sequências letra a letra e usando uma matriz de pontuação para determinar a similaridade entre os caracteres. A matriz de pontuação pode ser personalizada para refletir a biologia subjacente e as propriedades específicas das sequências sendo comparadas. O resultado final é um alinhamento completo das duas sequências, que pode ser usado para identificar relações evolutivas e funcionais entre proteínas ou para estudar a evolução molecular. O algoritmo de Needleman-Wunsch é amplamente utilizado em biologia computacional e é uma ferramenta valiosa para a comparação de sequências de DNA e proteínas. (Rashed et. al, 2021).

A implementação do algoritmo de Needleman-Wunsch envolve a construção de uma matriz de similaridade e o preenchimento desta matriz através da comparação de cada caractere das duas sequências. A matriz é preenchida usando uma fórmula que leva em conta a pontuação da similaridade entre os caracteres comparados, bem como o custo de inserção ou deleção de caracteres nas sequências. Uma vez preenchida a matriz, o algoritmo segue o caminho de maior similaridade através da matriz para produzir o alinhamento final das duas sequências. O alinhamento pode ser visualizado como uma série de caracteres iguais, substituições ou espaços representando inserções ou deleções. A implementação do algoritmo de Needleman-Wunsch é simples e eficiente, e pode ser facilmente integrada a vários sistemas de biologia computacional. (Rashed et. al, 2021).

2.5 Escalabilidade em aplicações do cenário *Big Data*

Escolher a plataforma ideal para Big Data exige conhecimento aprofundado sobre as capacidades das plataformas disponíveis. Porém, uma característica relevante em todas elas é

que a plataforma precisa ter a capacidade de se adaptar às crescentes demandas de processamento de dados. Para conseguir concluir se o processamento é suficiente, é necessário analisar o dimensionamento do sistema. O dimensionamento é o quanto um sistema é capaz de se adaptar à crescente expansão de processamento, e ele é separado em duas categorias, vertical e horizontal (SINGH; REDDY, 2015).

A escalabilidade horizontal se trata da possibilidade de distribuição de carga de trabalho entre várias máquinas diferentes. Já a escalabilidade vertical é sobre aumentar a capacidade computacional por meios físicos, como processadores adicionais ou mais memória RAM para o processamento (ALI, 2019). Segundo Hildebrandt a maneira mais eficiente de fazer o dimensionamento é a abordagem de escalabilidade horizontal, pois permite combinar a escalabilidade vertical em cada máquina individualmente e a escalabilidade horizontal que atua em paralelo (HILDEBRANDT, 2017).

2.6 Métricas de qualidade para detecção de tuplas duplicadas

Para identificarmos se os registros duplicados são confiáveis, ou seja, se foram identificados de maneira correta, são utilizados métricas para uma avaliação dos resultados (KÖPCKE; RAHM, 2010). As métricas utilizadas são revocação, precisão e F-measure.

A revocação identifica como a variável r se comporta comparado ao número de pares identificados corretamente

$$r = \frac{pares\ duplicados\ identificados\ corretamente}{pares\ duplicados\ existentes} \quad (2).$$

A precisão *p,* é número de tuplas duplicados sobre o número de tuplas duplicadas identificadas no processo de detecção de registro duplicados (DRAISBACH, U. et al, 2012).

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

E por fim, tem-se uma medida que relaciona p e q calculado anteriormente. O F-measure baseia-se no cálculo da média harmônica entre p e q, ou seja, mostra um equilíbrio entre as variáveis (CHAN; CHEN; TOWEY, 2002). O cálculo é feito da seguinte forma:

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

2.7 Trabalhos correlatos

Na era Big Data, as quantidades de dados geradas e armazenadas são enormes e crescentes, o que torna a tarefa de limpeza e remoção de registros duplicados desafiadora. Algoritmos tradicionais não são sempre eficientes o suficiente para lidar com essas grandes quantidades de dados. Na literatura, existem vários tópicos sobre limpeza de dados e remoção de registros duplicados, uma vez que os algoritmos existentes em si não são uma estratégia eficiente para atender às exigências da era Big Data. (Alenazi; Ahmad, 2016). Uma solução proposta para limpeza de dados e remoção de registros duplicados é o uso de funções de similaridade. (Xu; Wu, 2016). Uma estratégia eficaz para limpar os dados e remover os registros duplicados é a aplicação sucessiva de funções de similaridade, que envolve aplicar várias funções de similaridade ao mesmo conjunto de dados, a fim de obter resultados mais precisos e confiáveis. (Hu; Lian, 2018). Esta técnica também pode melhorar a precisão na identificação de registros duplicados, visto que ela considera mais atributos e aplica múltiplas funções de similaridade. (Zhang; Wang, 2017). Desta forma, são apresentados a seguir trabalhos da literatura que abordam conceitos de funções de similaridade aplicados em um contexto de Big Data.

O artigo de Pepe (2019) aborda a importância do pré-processamento e limpeza de dados no contexto de Big Data e apresenta um ambiente para a identificação de tuplas duplicadas com recursos de paralelização. Tal trabalho é semelhante ao atual, pois também aborda a questão da identificação de tuplas duplicadas em grandes volumes de dados e utiliza recursos de paralelização. Inicialmente, o artigo aborda o problema da limpeza de dados, que é um dos principais desafios para uma análise de dados eficiente e como os erros de correspondência podem ser descritos como resultado de erros de digitação, falta de padronização nos registros, informações incompletas ou maneiras diferentes de descrever a mesma entidade do mundo real. Em seguida, o autor apresenta um histórico sobre a identificação de tuplas duplicadas, e mostra as principais abordagens adotadas até o momento. O trabalho apresenta a proposta do sistema desenvolvido, que é baseado na utilização de uma única função de similaridade, a Damerau-Levenshtein, para comparar as strings e identificar as tuplas duplicadas. Além disso, o sistema usa recursos de paralelização, com o auxílio do Apache Spark, para melhorar a eficiência do processo. É importante notar que o artigo de Pepe se concentra na identificação de tuplas duplicadas e no pré-processamento de dados, enquanto o trabalho atual se concentra em explorar a eficácia das funções de similaridade

baseadas em strings e caracteres, e a adição de novas funções de similaridade para aumentar a precisão e eficiência na identificação de tuplas duplicadas.

O artigo de (MAHDI; AHMAD; ISMAIL, 2018) é importante para o trabalho atual pois fornece uma visão geral das técnicas de busca de similaridade utilizadas em cenários de pesquisa exploratória. Este trabalho destaca a importância da identificação e correção de dados duplicados como uma etapa crucial no processo de limpeza de dados. Além disso, o artigo destaca as dificuldades enfrentadas na identificação de dados duplicados, que incluem a variação de escrita e erros de digitação. Também foi apresentada uma revisão das técnicas de busca de similaridade utilizadas em pesquisa exploratória, que incluem métodos baseados em strings, corpus e conhecimento semântico. Ele também destaca a importância de técnicas de escalabilidade para lidar com grandes conjuntos de dados e discute várias abordagens para aumentar a eficiência do processo de busca de similaridade. Em resumo, o artigo (MAHDI; AHMAD; ISMAIL, 2018) fornece uma visão geral das técnicas de busca de similaridade utilizadas em pesquisa exploratória e destaca desafios semelhantes aos encontrados no trabalho atual. Isso ajuda a estabelecer uma base teórica para o desenvolvimento do sistema de identificação de tuplas duplicadas com recursos de paralelização.

O artigo de Bayrak et al. (2018) apresenta um estudo sobre técnicas de detecção de tuplas duplicadas não-idênticas em bancos de dados relacionais. Os autores propõem uma abordagem baseada na comparação de cadeias de caracteres que utilizam diferentes métodos, como o método de Jaccard, o método de Levenshtein e o método de Jaro-Winkler. O método de Jaccard é baseado na comparação de conjuntos de caracteres, que calcula a similaridade entre as cadeias como a proporção entre o número de caracteres comuns nas cadeias comparadas e o número total de caracteres diferentes. O método de Levenshtein, por sua vez, é baseado na contagem de operações (inserção, deleção ou substituição) necessárias para transformar uma cadeia em outra e é utilizado para medir a distância entre duas cadeias. E o método de Jaro-Winkler é utilizado para medir a similaridade entre duas strings, onde é baseado no número de caracteres iguais e no número de caracteres diferentes que as precedem dentro de uma das cadeias comparadas. Os autores apresentam uma revisão das técnicas de detecção de duplicatas próximas em bancos de dados relacionais, com enfoque nas técnicas de comparação de cadeias de caracteres. Eles discutem as principais abordagens existentes, como o uso de técnicas de normalização, o cálculo de similaridade baseado em distância de edição e o uso de algoritmos de hash. O artigo também apresenta uma análise comparativa das técnicas mencionadas, e mostra seus prós e contras.

Os artigos de Mahdi et al. (2018), Bayrak et al. (2018) e Pepe (2019) discutem diferentes abordagens para a limpeza e remoção de registros duplicados em contextos de Big Data. Enquanto Mahdi et al. (2018) destacam a importância das técnicas de busca de similaridade em buscas exploratórias, Bayrak et al. (2018) apresentam uma metodologia para detectar registros próximos duplicados em bancos de dados relacionais. Por sua vez, Pepe (2019) apresenta uma proposta de sistema baseado na utilização de uma única função de similaridade e recursos de paralelização. Além disso, é importante destacar que além da limpeza e remoção de registros duplicados, existem outras tarefas de pré-processamento de dados que podem ser realizadas antes da análise, como o uso de ferramentas e tecnologias avançadas que possibilitem o armazenamento, o gerenciamento e o processamento de grandes volumes de dados. As sugestões propostas, como similaridade baseada em hash, similaridade baseada em mineração de dados e similaridade baseada em aprendizado de máquina, são alternativas que podem ser avaliadas de acordo com o cenário e necessidade específica. Na Tabela 2 é existe uma comparação a respeito dos três artigos relacionados com o trabalho atual, que evidencia a importância das funções de similaridade para a limpeza e remoção de dados duplicados.

Tabela 2- – Comparativo entre os trabalhos correlatos a este trabalho

	(PEPE, 2019)	**(BAYRAK, 2018)**	**(MAHDI, M. AHMAD, A. R. ISMAIL (2018)**
Estratégias propostas ao cenário *Big Data*			

Discussão sobre técnicas de similaridade textual			
Realizam a combinação de sucessivas funções de similaridade para otimizar o cálculo de similaridade			

Fonte: Elaborado pelo autor

2.8 Considerações Finais

Esta seção abordou as temáticas de limpeza de dados, identificação de tuplas duplicadas não idênticas, cálculo de similaridade, técnicas de escalabilidade e paralelização com Apache Spark. As funções de similaridade podem ser usadas para quantificar a semelhança entre dois objetos. O processo de limpeza de dados é crucial para a descoberta de padrões, pois converte dados puros em abstrações. Uma solução proposta para limpeza de dados e remoção de registros duplicados é o uso de funções de similaridade, que podem ser aplicadas de forma sucessiva para obter resultados mais precisos e confiáveis. A escalabilidade é essencial para aplicações de Big Data, e a escalabilidade horizontal é a mais eficiente. A métrica de qualidade, revocação, precisão e F-measure pode ser usada para determinar se os registros duplicados são confiáveis. Com base nas discussões apresentadas neste trabalho, conclui-se que o uso de funções de similaridade, escalabilidade e métricas de qualidade permite a limpeza de dados eficiente e a identificação de registros duplicados confiáveis.

3 Desenvolvimento do projeto

Esta seção apresentou uma descrição detalhada do ambiente de apoio à identificação de tuplas duplicadas não idênticas, bem como descreveu as tecnologias utilizadas. De início foram definidas as padronizações ortográficas, palavras de parada e afixos, a fim de configurar o ambiente para preparar para o pré-processamento. Após a configuração do ambiente foi desenvolvida a etapa de pré-processamento, essas duas etapas garantiram dados mais precisos e consistentes. Com os dados preparados iniciou a fase de identificação de registros duplicados utilizou-se sucessivas funções de similaridade com finalidade de aumentar a precisão da identificação de tuplas duplicadas.

3.1 Ambiente para detecção de tuplas duplicadas não idênticas

O desenvolvimento do ambiente tem início com a inserção de dados pelo usuário, estes que simulam dados inseridos manualmente e passíveis a erro. O usuário também insere dados de configuração para calibrar o ambiente de acordo com o contexto desejado. Após a configuração, é feita a etapa de pré-processamento dos dados, etapa que ajuda a limpar os dados para minimizar erros e identificar com maior precisão as tuplas duplicadas.

Após a realização da etapa de pré-processamento, foram utilizados três algoritmos de funções de similaridade para identificação de registros duplicados. Segundo (PEPE, 2019), é possível observar que cada algoritmo trabalha melhor em determinados contextos, então neste trabalho foram utilizados três funções em sequência para que fosse possível verificar se ao usar mais de uma função de similaridade é viável tratar contextos mais diversos e com uma maior precisão, revocação, F-measure e desempenho.

Nessa seção então são testadas diversas combinações de funções de similaridade sucessivamente para otimizar os cálculos de similaridade. As funções escolhidas foram o algoritmo de Damerau-Levenshtein, Jaro-Winkler E Needleman-Wunsch, nos quaissão algoritmos de similaridade de strings que calculam a distância entre duas strings.

O algoritmo Damerau-Levenshtein usa a operação de inserção, deleção, substituição e transposição para encontrar a distância entre duas strings. É utilizado principalmente para encontrar strings que se aproximam. Por exemplo, a palavra "carro" e "carrro" teriam uma distância de um, pois apenas a letra "r" é diferente (Damerau, 1964).

O algoritmo Jaro-Winkler é semelhante ao Damerau-Levenshtein, mas também leva em consideração as letras adjacentes às letras diferentes e dá mais peso à string inicial (Winkler, 1990). Por exemplo, a palavra "carro" e "carrro" teriam uma distância zero, pois, as letras adjacentes são iguais.

O algoritmo Needleman-Wunsch é baseado na programação dinâmica (Needleman, 1970). Ele compara strings e calcula a distância entre elas por meio de uma matriz. A distância calculada faz uso da soma dos valores da matriz, e a melhor string é a que produz a menor distância entre as strings. Por exemplo, a palavra "carro" e "carrro" teriam uma distância zero, pois, a matriz é preenchida com os mesmos valores. Portanto, os três algoritmos de similaridade de strings são diferentes em sua abordagem, mas todos permitem que você encontre a distância entre duas strings.

Para encontrar tuplas duplicadas que usam os algoritmos Damerau-Levenshtein, Jaro-Winkler e Needleman-Wunsch, cria-se um código que utiliza esses algoritmos em sequência para processar as strings. O código começa lendo as strings a serem comparadas e, em seguida, cada algoritmo é aplicado para calcular a distância entre as strings. Se a distância encontrada for menor que o limite definido, as strings são consideradas similares e, portanto, tuplas duplicadas.

Para fazer isso, o código precisa passar os dados de uma função para outra. Cada algoritmo processa o input de uma forma ligeiramente diferente e devolve um resultado. O código deve garantir que esses resultados sejam passados para a próxima função, de forma que cada algoritmo possa usar os resultados do algoritmo anterior.

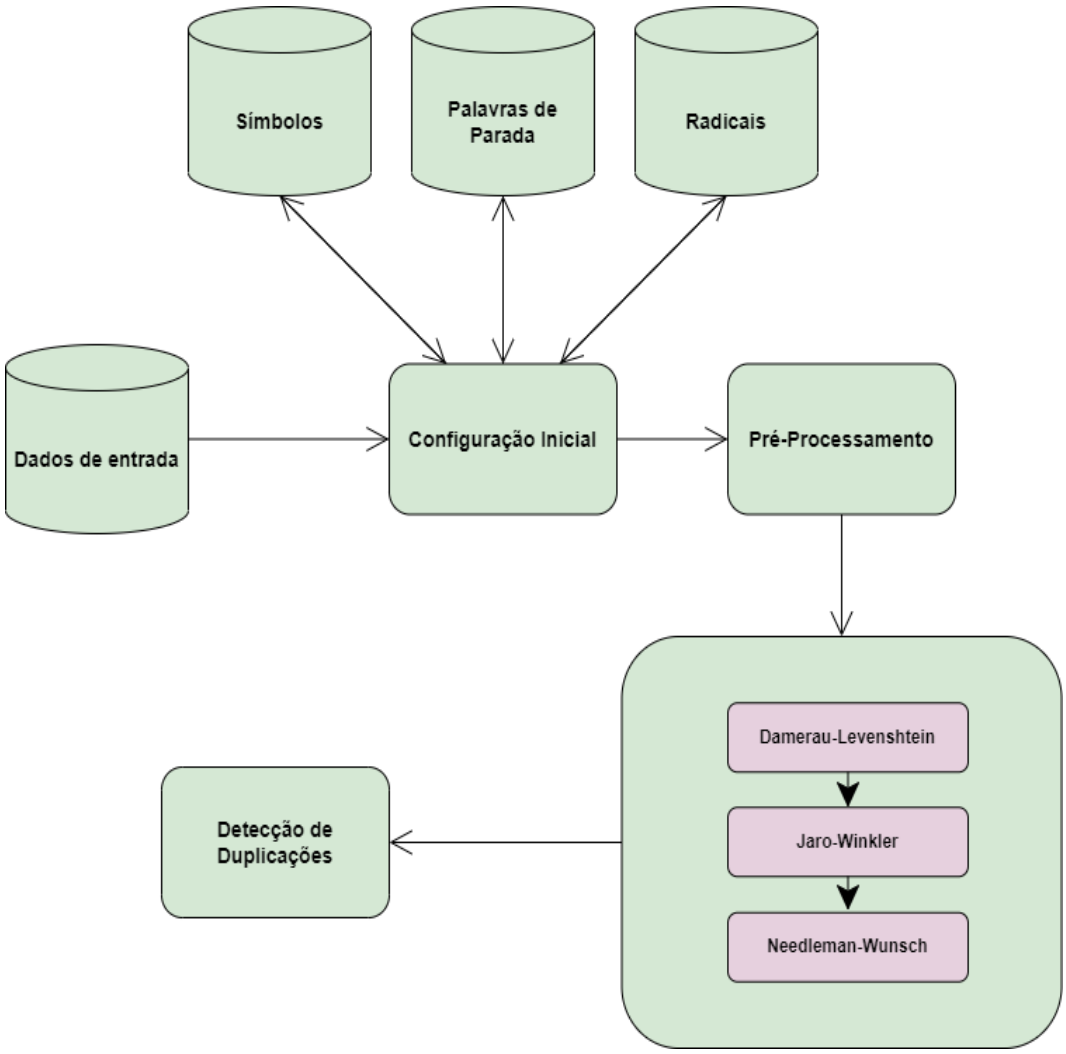
Ao procurar por tuplas duplicadas, utiliza-se uma sequência de três algoritmos distintos. Inicialmente, realiza-se a leitura das strings que serão comparadas. Em seguida, aplicamos o algoritmo Damerau-Levenshtein para armazenar o resultado obtido. Esse resultado é então utilizado pelo algoritmo Jaro-Winkler para calcular a distância entre as strings, o qual também se armazena e passa para o próximo algoritmo, o Needleman-Wunsch. Este último utiliza o resultado anterior para calcular a distância entre as strings e o resultado é comparado com um limite estabelecido para determinar se as strings são consideradas similares. Portanto, a combinação dos algoritmos Damerau-Levenshtein, Jaro-Winkler e

Needleman-Wunsch possibilita encontrar tuplas duplicadas de forma eficiente. (Damerau, 1964; Winkler, 1990; Needleman, 1970).

O ambiente foi implementado com recursos de paralelização e processamento em memória por meio da ferramenta Apache Spark. O ambiente construído pode ser descrito na Figura 3.

.

Figura 3 – Ambiente de Identificação de Tuplas Duplicadas



Fonte: elaborada pelo autor

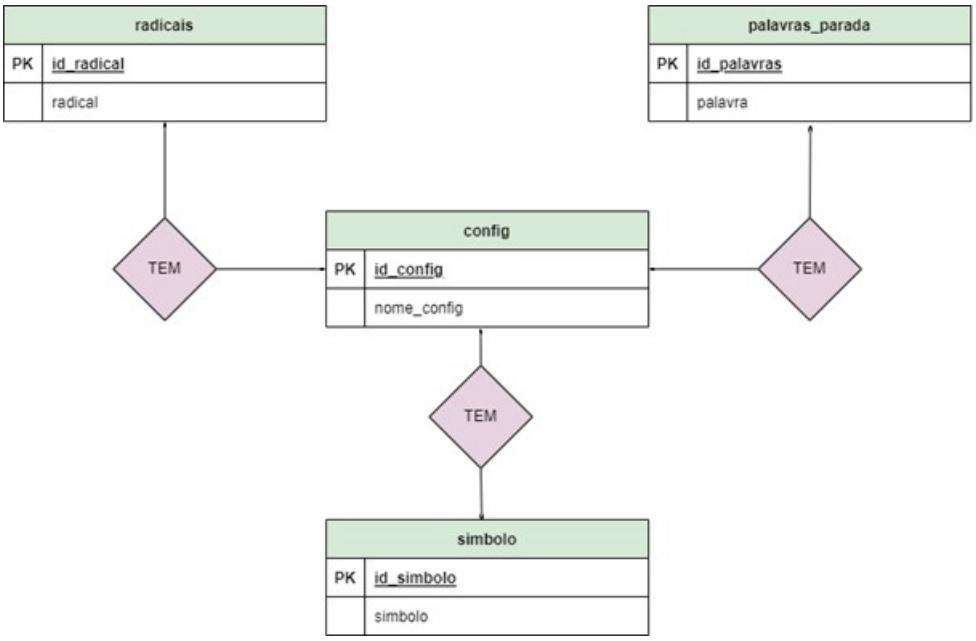
3.1.1 Configuração inicial

. A primeira parte do diagrama da Figura 3 é a respeito da Configuração Inicial, onde essa etapa tem por objetivo configurar as propriedades da base de dados desejada por meio da entrada de dados do usuário. Os dados são inseridos em uma estrutura chamada Resilient Distributed Dataset (RDD) do Apache Spark. RDD que traduzido significa Conjunto de Dados Distribuído Resiliente, é uma estrutura de dados distribuída de alto desempenho que permite o processamento de grandes quantidades de dados. Esta estrutura possui a capacidade de aproveitar o processamento paralelo e distribuir o processamento de dados em várias máquinas. Além disso, a RDD possui uma estrutura de dados genérica, o que permite que sejam armazenadas várias estruturas de dados específicas, como matrizes, vetores, listas, tabelas e outras, de forma a melhorar o desempenho de processamento. A RDD do Apache Spark fornece uma interface de programação de alto nível para a criação de aplicações de big data. Essa interface permite aos usuários criar, manipular e processar dados de forma eficiente. Além do mais, o RDD possui uma arquitetura com camadas que permite a execução de aplicações em clusters de computadores (CHAWLA; SHARMA; 2020).

Depois de inseridas as informações pelo usuário, é necessário ajustar o limite de proximidade que o algoritmo de identificação de conjuntos duplicados requer. Esta informação ajusta o alcance máximo de busca, ou seja, quanto menor for o valor fornecido, maior será a similaridade entre os conjuntos.

Ainda nessa etapa, é necessário configurar o contexto que será trabalhado, então são inseridos os símbolos e sinais de pontuação que devem ser ignorados pelo sistema, as de palavras de parada importantes para o contexto, e por último, é preciso configurar os sufixos e prefixos a serem levados em conta pelo ambiente. Portanto, foram utilizadas 3 bases de dados para alimentar o ambiente, Radicais, Símbolos e Palavras de Parada. Os relacionamentos entre tais dados podem ser observados na Figura 4.

Figura 4 – Modelo Entidade de Relacionamento

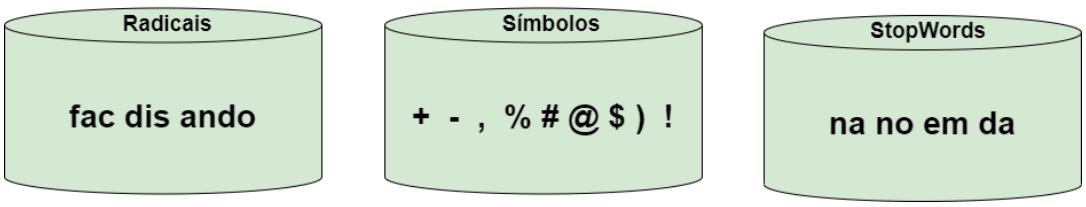


Fonte: Adaptado de Pepe, 2019

3.1.2 Pré-processamento

A segunda parte do sistema elaborado realiza o pré-processamento dos dados, ou seja, padroniza os dados de acordo com o contexto configurado pelo usuário. Além dos dados desejados inseridos, são feitas padronizações como colocar todas as letras em letras minúsculas e são retirados pontuações, acentos e caracteres especiais. Em seguida, é realizada a tokenização de palavras para que seja possível transformar em uma lista útil de palavras. Por último, são removidas as palavras de parada, os sufixos e os prefixos das palavras, até restar apenas os radicais. Um exemplo de possíveis palavras cadastradas pelo usuário está na Figura 5.

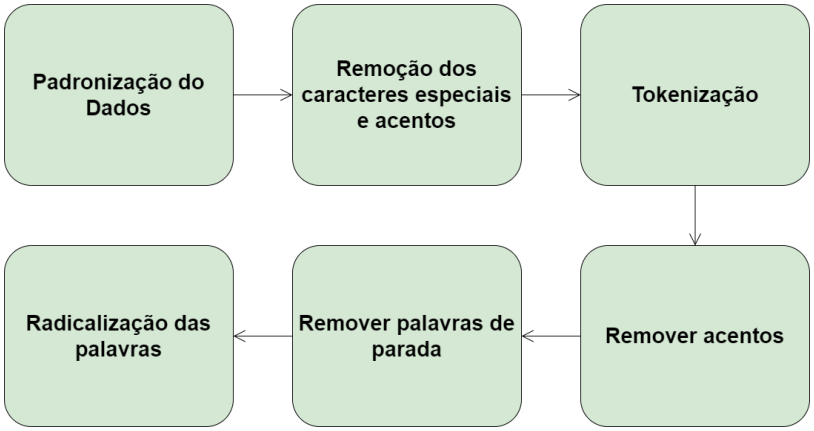
Figura 5 – Entradas de Configuração inseridas pelo usuário



Fonte: Elaborado Pelo Autor

Portanto, o módulo de Pré-processamento tem como objetivo fazer a correção de dados incorretos e irrelevantes para a identificação de tuplas duplicadas. No final deste módulo resta apenas palavras relevantes para entrar na etapa de combinação de diversas funções de similaridade. O pré-processamento usa a função map do Spark, isso possibilita trabalhar de forma paralela e se aplica ao RDD nos dados de entrada. Na Figura 6 é exibida como é feita a segunda parte do sistema.

Figura 6 – Pré-processamento



Fonte: Adaptado de Pepe, 2019

3.1.3 Identificação de duplicações

A terceira etapa do sistema é onde são realizadas as identificações de tuplas duplicadas. É nesta etapa que está a contribuição científica deste trabalho. Nos trabalhos correlatos é escolhida uma única função de similaridade para comparar se os dados são iguais, onde neste foram combinadas várias funções de similaridade em sequência, a fim de observar melhores resultados. Para avaliar a qualidade das combinações de similaridade, são observadas a quantidade de tuplas duplicadas e o tempo de processamento para achar essas duplicações.

3.2 Tecnologias utilizadas

A linguagem escolhida foi a linguagem Java e o framework foi o Apache Spark que possibilita a paralelização dos recursos que otimiza o tempo de processamento total do ambiente. A base de dados escolhida foi o PostgreSQL e a IDE para execução do projeto foi o NetBeans.

3.3 Considerações Finais

Esta pesquisa teve como objetivo avaliar a eficácia de três algoritmos de similaridade de strings para identificar tuplas duplicadas. Para isso, foram usados os algoritmos Damerau-Levenshtein, Jaro-Winkler e Needleman-Wunsch para calcular a distância entre duas strings e verificar se eram consideradas similares. Primeiramente, foi necessário configurar o ambiente, que usa o Resilient Distributed Dataset (RDD) e três bases de dados (Radicais, Símbolos e Palavras de Parada). O código foi desenvolvido em Java e o framework escolhido foi o Apache Spark, com a IDE NetBeans para execução do projeto. A seguir, foi realizado o pré-processamento dos dados, que incluiu a padronização, tokenização de palavras, remoção de palavras de parada, sufixos e prefixos, que usa a função map do Spark. Por fim, foram combinadas diversas funções de similaridade para encontrar as duplicações. O objetivo deste ambiente foi proporcionar um meio eficaz para identificar tuplas duplicadas.

4 Avaliação Experimental

Neste trabalho foi feita a construção de um ambiente para a identificação de tuplas duplicadas em um ambiente de paralelização e processamento em memória, neste capítulo serão exibidos os resultados obtidos após executar o ambiente proposto, bem como os métodos de experimentação, a fim de verificar a qualidade dos resultados obtidos e as possíveis contribuições científicas.

4.1 Metodologia de experimentação

Para verificar os resultados foram feitos um experimento para avaliar o desempenho do ambiente, ou seja, o quão rápido o algoritmo é executado, e um experimento para avaliar a qualidade dos dados, ou seja, quantas tuplas duplicadas o sistema consegue identificar e se são válidos e confiáveis. O diferencial deste trabalho em relação aos trabalhos correlatos é que os dados passam por mais de uma função de similaridade para identificar tuplas idênticas que busca maximizar e melhorar os resultados gerados na busca por duplicações. Os algoritmos escolhidos foram Damerau-Levenshtein, Jaro-Winkler e Needleman-Wunsch, estes se encaixam na categoria de funções de similaridade baseadas em strings e na subcategoria baseada em caracteres.

O experimento de análise de qualidade visa testar as funções individualmente para a realização das buscas, e testar se com a combinação delas gera resultados mais precisos. É possível ver que por meio do contexto uma combinação dessas funções de similaridade é mais eficiente do que outras, e que adicionar mais funções de similaridade diferentes apesar de gerar resultados mais precisos, aumentam o tempo de processamento, porém para um contexto de Big Data ainda é uma decisão relevante. A extração de dados foi feita a partir do sistema Kaggle, que é um dos principais sites de Ciência de Dados no mundo para testes, estudos e mineração. A base de dados do Kaggle escolhida foi a Sistemas de Informação e Vigilância de Acidentes de Trabalho (SIVAT).

Para verificar o desempenho dos algoritmos foram utilizadas amostras da base de dados SIVAT, primeiro cada algoritmo escolhido foi testado individualmente e depois em

sequência. A base escolhida possui dados que foram inseridos por usuários, portanto, podem existir erros de digitação que torna possível gerar erros na identificação de tuplas duplicadas.

Os experimentos foram rodados em um hardware com as seguintes especificações: processador Intel® Core™ i7-1185G7 (3.000 GHz), memória RAM de 16GB e o sistema operacional Windows 11. O software utilizado para realizar a paralelização para que fosse possível paralelizar os recursos foi o Apache Spark, disponível em: https://spark.apache.org/downloads.html. O compilador utilizado foi o Apache NetBeans IDE 16 disponível em: https://netbeans.apache.org/download/index.html. O banco de dados foi o PostgreSQL e pode ser encontrado em: https://www.postgresql.org/download/.

Foram adicionadas 173 linhas originais de código no total no ambiente CLER baseado no trabalho desenvolvido por (Pepe, 2019). Foram adicionadas 86 linhas no arquivo NwSpark.java, arquivo cujo qual o objetivo foi implementar o algoritmo Needleman-Wunsch paralelizado. Foram adicionadas 57 linhas para o arquivo JwSpark.java, arquivo adicionado com o objetivo de paralelizar o algoritmo de Jaro-Winkler como é mostrado nas Figuras 7 e Figura 8. E 9 linhas na função Main.java para chamar os algoritmos de similaridade em sequência como é mostrado na Figura 7.

Figura 7 – Código para colocar funções de similaridade em sequência

```
1  //calcular tuplas em sequência
2  JavaPairRDD<Integer, Tuple2<Row, Row>> res1 =
3  NeedlemanWunschSpark.sparkNeedlemanWunsch(spark, dados, dados, threshold);
4
5  JavaPairRDD<Integer, Tuple2<Row, Row>> res2 =
6  sparkDamerauLevenshtein.sparkDamerauLevenshtein(spark, res1.values(), res1.values(), max_d);
7
8  JavaPairRDD<Integer, Tuple2<Row, Row>> res3 =
9  NeedlemanWunschSpark.sparkNeedlemanWunsch(spark, res2.values(), res2.values(), threshold);
```

Fonte: Elaborado pelo autor

Figura 8 – Parte 1 – Código de paralelização do algoritmo Jaro-Winkler em java

```java
package controller;
import java.io.File;
import java.io.FileWriter;
import java.io.PrintWriter;
import org.apache.spark.api.java.JavaPairRDD;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.broadcast.Broadcast;
import org.apache.spark.sql.Row;
import org.apache.spark.sql.SparkSession;
import scala.Tuple2;

public class NeedlemanWunschSpark {
    public static JavaPairRDD<Integer, Tuple2<Row, Row>> sparkNeedlemanWunsch(
        SparkSession spark, JavaRDD<Row> dados, JavaRDD<Row> dados2, Integer max_d,
        String filePath) {
        Broadcast<Integer> max_d_broad = spark.sparkContext().broadcast(max_d,
        scala.reflect.ClassTag$.MODULE$.apply(Integer.class));
        Long tempoInicial, tempoFinal;
        JavaRDD<Row> rdd1 = dados;
        JavaRDD<Row> rdd2 = dados2;
        File file = new File(filePath);
        file.delete(); //deleta o arquivo caso exista.
        tempoInicial = System.currentTimeMillis();
        JavaPairRDD<Row, Row> cart = rdd1.cartesian(rdd2); // pair (conj1, conj2)

        JavaPairRDD<Integer, Tuple2<Row, Row>> conj = cart.mapToPair((t1) -> {
            Row r1 = t1._1;
            Row r2 = t1._2;
            String colR1 = r1.getString(1);
            String colR2 = r2.getString(1);

            if (colR1 == null || colR2 == null) {
                return new Tuple2(-1, t1);
            }
            int tam1 = colR1.length();
            int tam2 = colR2.length();
            if (Math.abs(tam1 - tam2) > max_d_broad.value()) {
                return new Tuple2(-1, t1);
            } else {
                if (r1.getString(0).equals(r2.getString(0))) {
                    return new Tuple2(-1, t1);
                }
                int dist = getDistance(colR1, colR2);
```

Fonte: Elaborado pelo autor

Figura 9 – Parte 2 – Código de paralelização do algoritmo Jaro-Winkler em java

```
                if (dist <= max_d_broad.value()) {
                    try {
                        FileWriter arq = new FileWriter(file, true);
                        PrintWriter gravarArq = new PrintWriter(arq);
                        gravarArq.printf("%s,%s,%d\n", r1.getString(0),
                        r2.getString(0), dist);
                        arq.close();
                    } catch (Exception e) {
                        e

                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                    return new Tuple2(dist, t1);
                }
                else {
                    return new Tuple2(-1, t1);
                }
            }
        });
        tempoFinal = System.currentTimeMillis();
        System.out.println("Tempo de execução: " + (tempoFinal - tempoInicial) + "s");
        return conj;
    }

    public static int getDistance(String s1, String s2) {
        int[][] distanceMatrix = new int[s1.length() + 1][s2.length() + 1];

        for (int i = 0; i <= s1.length(); i++) {
            distanceMatrix[i][0] = i;
        }

        for (int j = 0; j <= s2.length(); j++) {
            distanceMatrix[0][j] = j;
        }

        for (int i = 1; i <= s1.length(); i++) {
            for (int j = 1; j <= s2.length(); j++) {
                int cost = (s1.charAt(i - 1) == s2.charAt(j - 1) ? 0 : 1);
                distanceMatrix[i][j] = Math.min(Math.min(distanceMatrix[i - 1][j] + 1,
                distanceMatrix[i][j - 1] + 1), distanceMatrix[i - 1][j - 1] + cost);
            }
        }
```

Fonte: Elaborado pelo autor

Figura 10 – Código de paralelização do algoritmo Needleman-Wunsch em java

```java
package controller;
import java.io.File;
import java.io.FileWriter;
import java.io.PrintWriter;
import org.apache.commons.text.similarity.JaroWinklerDistance;
import org.apache.spark.api.java.JavaPairRDD;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.broadcast.Broadcast;
import org.apache.spark.sql.Row;
import org.apache.spark.sql.SparkSession;
import scala.Tuple2;

public class JaroWinklerSpark {
    public static JavaPairRDD<Double, Tuple2<Row, Row>> sparkJaroWinkler(SparkSession spark,
    JavaRDD<Row> dados, JavaRDD<Row> dados2, Double threshold, String filePath) {
        Broadcast<Double> thresholdBroadcast = spark.sparkContext().broadcast(threshold,
        scala.reflect.ClassTag$.MODULE$.apply(Double.class));
        Long tempoInicial, tempoFinal;

        JavaRDD<Row> rdd1 = dados;
        JavaRDD<Row> rdd2 = dados2;
        File file = new File(filePath);
        file.delete(); //deleta o arquivo caso exista.
        tempoInicial = System.currentTimeMillis();
        JavaPairRDD<Row, Row> cart = rdd1.cartesian(rdd2); // pair (conj1, conj2)

        JavaPairRDD<Double, Tuple2<Row, Row>> conj = cart.mapToPair((t1) -> {
            Row r1 = t1._1;
            Row r2 = t1._2;
            String colR1 = r1.getString(1);
            String colR2 = r2.getString(1);

            if (colR1 == null || colR2 == null) { return new Tuple2(-1.0, t1); }

            if (r1.getString(0).equals(r2.getString(0))) { return new Tuple2(-1.0, t1); }

            JaroWinklerDistance jaroWinkler = new JaroWinklerDistance();
            double similarity = jaroWinkler.apply(colR1, colR2);

            if (similarity >= thresholdBroadcast.value()) {
                try {
                    FileWriter arq = new FileWriter(file, true);
                    PrintWriter gravarArq = new PrintWriter(arq);
                    gravarArq.printf("%s,%s,%.4f\n", r1.getString(0), r2.getString(0), similarity);
                    arq.close();
                } catch (Exception e) {
                    e.printStackTrace();
                }
                return new Tuple2(similarity, t1);
            } else {
                return new Tuple2(-1.0, t1);
            }
        });
        tempoFinal = System.currentTimeMillis();
        System.out.println("Tempo de execução: " + (tempoFinal - tempoInicial) + "ms");
        return conj;
    }
```

Fonte: Elaborado pelo autor

4.2 Análise da Qualidade das funções de similaridade

A análise de qualidade foi feita a partir da execução dos algoritmos Damerau-Levenshtein, Jaro-Winkler e Needleman-Wunsch individualmente, e depois foram colocados em sequência para verificar se são gerados mais registros duplicados do que as funções executadas individualmente.

A configuração inicial para todas as etapas é a mesma para verificar a confiabilidade e acurácia. A configuração inicial utilizada foi a distância máxima 3 e 23 regras inseridas para cada de contexto, e a base SIVAT foi utilizada em todos os casos. Depois do resultado das duplicações de cada algoritmo individualmente e após dos algoritmos combinados, foi realizado cálculos de precisão, revocação e F-measure conforme mostrados na seção 2.5. Na análise da qualidade das funções de similaridade, foram aplicadas as métricas de precisão, revocação e F-measure para cada algoritmo individualmente e, em seguida, para a combinação de algoritmos. Essas métricas fornecem uma visão abrangente do desempenho dos algoritmos na detecção de registros duplicados no conjunto de dados.

A precisão mede a proporção de resultados positivos verdadeiros (registros duplicados corretamente identificados) em relação ao número total de resultados positivos (tanto identificações corretas quanto incorretas). Uma alta precisão significa que o algoritmo gera poucos falsos positivos (identificou poucos registros não duplicados como duplicados) e, portanto, é mais preciso na identificação de verdadeiros duplicados.

A revocação, por outro lado, mede a proporção de resultados positivos verdadeiros (registros duplicados corretamente identificados) em relação ao número total de registros duplicados reais no conjunto de dados. Uma alta revocação significa que o algoritmo é capaz de detectar a maioria dos registros duplicados no conjunto de dados.

Por fim, a F-measure é uma média harmônica de precisão e revocação, que fornece uma única métrica que equilibra a precisão e a revocação. Uma alta F-medida indica que o algoritmo tem alta precisão e alta revocação e, portanto, é capaz de detectar a maioria dos registros duplicados reais no conjunto de dados, enquanto gera poucos falsos positivos.

É exibido na tabela 3 os resultados obtidos para cada algoritmo e a combinação de algoritmos em termos de precisão, revocação e F-measure. Como pode-se observar, o algoritmo Damerau-Levenshtein tem a maior precisão, revocação e F-measure entre os algoritmos individuais, com valores de 0,877, 0,977 e 0,924, respectivamente. Isso indica que o algoritmo Damerau-Levenshtein é capaz de identificar corretamente a maioria dos registros

duplicados no conjunto de dados (alta revocação) enquanto produz um baixo número de falsos positivos (alta precisão).

Ao combinar o algoritmo Damerau-Levenshtein com Jaro-Winkler, a F-measure diminui de 0,924 para 0,900, entretanto o número de tuplas identificadas é maior, e comparado com a função Jaro-Winkler isolada o algoritmo Damerau-Levenshtein melhorou o desempenho dela consideravelmente. Isso indica que a combinação desses dois algoritmos é capaz de melhorar o equilíbrio entre precisão e revocação e, portanto, fornecer resultados mais precisos na detecção de registros duplicados.

Quando combinado o Damerau-Levenshtein com Needleman-Wunsch, a F-medida também diminui de 0,924 para 0,919, entretanto o número de tuplas identificadas também é maior. O que indica que essa combinação também é capaz de identificar corretamente a maioria dos registros duplicados no conjunto de dados (alta revocação) enquanto produz um baixo número de falsos positivos (alta precisão).

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

Algoritmos	Pares encontrados	Duplicações encontradas	Precisão	Revocação	F-measure
Damerau-Levenshtein	58	55	0,877	0,977	0,924
Jaro-Winkler	53	41	0,690	0,909	0,785
Needleman-Wunsch	47	40	0,822	0,902	0,808
Damerau-Levenshtein + Jaro-Winkler	65	63	0,841	0,969	0,900
Jaro-Winkler + Needleman-Wunsch	70	61	0,847	0,872	0,859
Damerau-Levenshtein + Needleman-Wunsch	66	61	0,916	0,924	0,919
Damerau-Levenshtein + Jaro-Winkler + Needleman-Wunsch	74	70	0,921	0,945	0,932

Fonte: elaborado pelo autor

Em geral, os resultados mostram que a combinação do algoritmo Damerau-Levenshtein com Needleman-Wunsch obteve os melhores resultados, com a maior F-medida. No entanto, a combinação de Damerau-Levenshtein com Jaro-Winkler também proporcionou resultados melhores do que o algoritmo individual. Esses resultados sugerem que a combinação de múltiplos algoritmos fornece uma abordagem mais abrangente e melhora o desempenho na identificação de registros duplicados no conjunto de dados.

De acordo com os resultados apresentados na Tabela 3, pode-se concluir que o uso de múltiplas funções de similaridade em sequência melhora significativamente a precisão, revocação e F-medida na detecção de registros duplicados. Isso sugere que, quando o objetivo é maximizar a qualidade dos resultados e o número de registros duplicados encontrados, é vantajoso utilizar mais de uma função de similaridade, pois isso permite cobrir contextos mais amplos e abrangentes.

Além disso, é importante mencionar que a ordem dos algoritmos é fundamental nesse processo. Se fosse utilizada uma ordem diferente para processar as tuplas, os resultados poderiam ser diferentes. Isso ocorre porque cada algoritmo tem suas próprias vantagens e desvantagens, e a ordem em que eles são usados pode afetar o desempenho final. Portanto, é importante avaliar e selecionar cuidadosamente a ordem dos algoritmos para garantir resultados ótimos. Em suma, o uso de múltiplas funções de similaridade em sequência é uma abordagem promissora na detecção de registros duplicados, mas a ordem dos algoritmos também desempenha um papel importante na qualidade dos resultados.

4.3 Testes de desempenho do ambiente

O segundo teste mede o desempenho do ambiente construído de acordo com o número de amostrar analisadas, que compara a execução individual da função Damerau-Levenshtein, em seguida a execução da combinação dos algoritmos Damerau-Levenshtein, Jaro-Winkler e após a combinação dos algoritmos Damerau-Levenshtein, Jaro-Winkler e Needleman-Wunsch. Dessa maneira, foram testados os casos com 2 mil, 4 mil, 8 mil e 16 mil dados iniciais retirados da base de dados do SIVAT. Foram executados cinco vezes cada teste para testar cada situação, e calculadas as médias para comparar o tempo de execução dos algoritmos individualmente e em sequência. A medida de distância máxima utilizada foi de novo igual a 3.

A Tabela 4 mostra os resultados dos testes de desempenho do ambiente, que compara o tempo de execução dos algoritmos Damerau-Levenshtein, Damerau-Levenshtein + Jaro-Winkler, e Damerau-Levenshtein + Jaro-Winkler + Needleman-Wunsch. Como observado na tabela, o tempo de execução aumenta de acordo com o aumento do número de amostras presentes no conjunto de dados.

Quando se trata do algoritmo Damerau-Levenshtein individual, observamos que ele leva cerca de 2,5 segundos para processar 2 mil amostras, cerca de 4,8 segundos para processar 4 mil amostras, cerca de 9,4 segundos para processar 8 mil amostras e cerca de 17,9 segundos para processar 16 mil amostras.

Quando combinado com o algoritmo Jaro-Winkler, o tempo de execução aumenta para cerca de 4,1 segundos para 2 mil amostras, cerca de 7,9 segundos para 4 mil amostras, cerca de 15,8 segundos para 8 mil amostras e cerca de 28,7 segundos para 16 mil amostras.

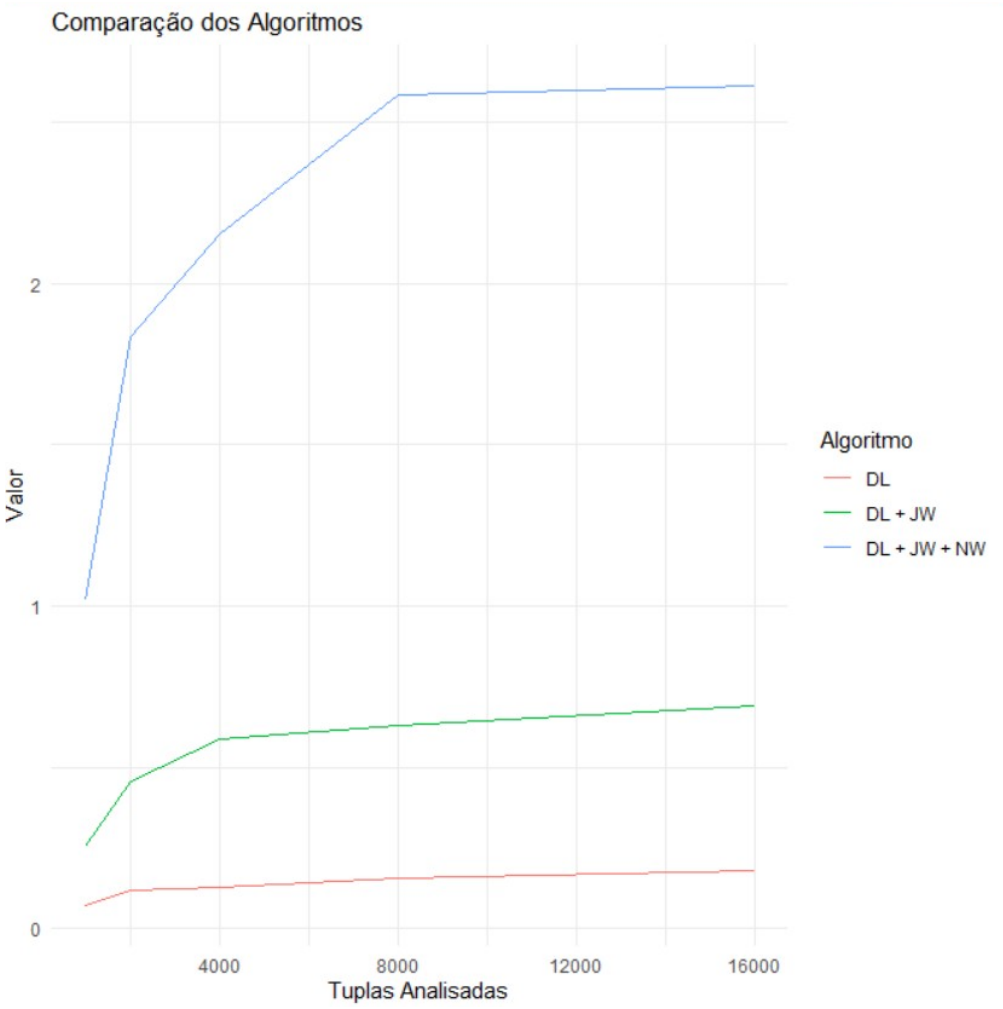
E quando combinado com o algoritmo Needleman-Wunsch, o tempo de execução aumenta para cerca de 5,5 segundos para 2 mil amostras, cerca de 9,7 segundos para 4 mil amostras, cerca de 18,2 segundos para 8 mil amostras e cerca de 34,9 segundos para 16 mil amostras.

Tabela 4 – Comparação de tempo de execução entre os algoritmos de similaridade

Tempo de execução (s)			
Tuplas analisadas	**Damerau-Levenshtein**	**Damerau-Levenshtein, Jaro-Winkler**	**Damerau-Levenshtein, Jaro-Winkler, Needleman-Wunsch**
1000	0,072	0,256	1,019
2000	0,120	0,456	1,836
4000	0,130	0,589	2,156
8000	0,158	0,631	2,588
16000	0,178	0,689	2,613

Fonte: elaborada pelo autor

Figura 11 – Comparação de tempo de execução entre os algoritmos



Fonte: elaborada pelo autor

A partir da análise dos dados presentes na Tabela 4 e da Figura 11, é possível observar que a relação entre o número de tuplas analisadas e o tempo de execução dos algoritmos é consistente com uma curva logarítmica. Isso significa que o aumento no número de tuplas analisadas resulta em um aumento proporcional na potência logarítmica do tempo de execução. Esse comportamento pode ser explicado pela complexidade dos algoritmos envolvidos, que resultam em uma relação não linear entre o número de tuplas analisadas e o tempo de execução.

A Figura 11 fornece uma representação gráfica das curvas logarítmicas para cada um dos algoritmos analisados. É possível ver que, conforme o número de tuplas analisadas aumenta, o tempo de execução também aumenta de maneira proporcional à potência

logarítmica do número de tuplas. Esse comportamento é esperado para algoritmos cuja complexidade é logarítmica.

Além disso, é possível ver que a implementação de Jaro-Winkler junto com Damerau-Levenshtein resulta em melhor desempenho em comparação com a implementação de Damerau-Levenshtein sozinho. Isso sugere que Jaro-Winkler é menos complexo computacionalmente e, portanto, é capaz de aumentar o desempenho sem comprometer muito a precisão. Por outro lado, a adição de Needleman-Wunsch tem um impacto negativo no desempenho, o que pode ser explicado pela complexidade computacional elevada do algoritmo Needleman-Wunsch em comparação com Jaro-Winkler.

Em conclusão, a escolha dos algoritmos a serem utilizados para a análise de tuplas deve ser feita com base em um equilíbrio entre a precisão e o tempo de execução desejados. É importante considerar cuidadosamente as implicações da adição de algoritmos mais complexos, uma vez que eles podem resultar em um aumento significativo no tempo de execução sem garantir uma melhoria significativa na precisão.

4.4 Considerações finais

Os testes apresentados neste capítulo tiveram como objetivo avaliar o desempenho e qualidade das funções de similaridade executadas individualmente e em combinação. Foi utilizada a base de dados SIVAT e as métricas de precisão, revocação e medida F foram utilizadas para avaliar os resultados dos testes de qualidade.

Os resultados dos testes de qualidade mostraram que a combinação de múltiplas funções de similaridade é uma abordagem eficiente para melhorar a precisão, e revocação e a medida F na detecção de registros duplicados. Isso foi evidenciado quando comparado a utilização de uma função individual, onde a combinação de Damerau-Levenshtein com Needleman-Wunsch e Damerau-Levenshtein com Jaro-Winkler apresentou os melhores resultados, com as maiores F-medida. Isso sugere que combinar múltiplas funções de similaridade permite abranger contextos mais amplos e completos, o que aumenta a acurácia nos resultados.

Os testes de desempenho também mostraram que a utilização de uma combinação de algoritmos tem um impacto no aumento do tempo de execução, devido ao processamento adicional de dados necessário para aplicar múltiplas funções de similaridade. Isso pode ser esperado pois, quanto mais funções de similaridade são aplicadas, mais dados precisam ser

processados. No entanto, mesmo com esse impacto, a combinação de algoritmos ainda pode ser relevante em um contexto de Big Data.

Isso se dá pelo fato de que, em um contexto de Big Data, é comum trabalhar com grandes quantidades de dados e a detecção de registros duplicados pode se tornar um desafio. Nessa situação, é importante garantir a qualidade dos dados, o que pode ser alcançado ao utilizar múltiplas funções de similaridade. Isso permite abranger contextos mais amplos e completos, o que resulta em uma detecção mais precisa e completa de registros duplicados. Além disso, trabalhar com Big Data exige uma estratégia computacional escalável, e a combinação de algoritmos pode ser uma forma de maximizar a eficiência na detecção de duplicatas.

5 Conclusão

Ao longo deste trabalho foi desenvolvido um ambiente para identificação de tuplas duplicadas com recursos de paralelização, e tem como base o trabalho "Ambiente para identificação de tuplas duplicadas com recursos de paralelização: pré-processamento eficiente para limpeza de dados" (Pepe, 2019). O ambiente desenvolvido consiste em uma base de dados inserida pelo usuário, configurações definidas pelo usuário e etapas de pré-processamento e identificação de tuplas duplicadas.

O objetivo deste trabalho foi compreender se a adição de novas funções de similaridade poderia resultar em uma identificação de tuplas mais precisa, e para isso foram utilizadas três funções de similaridade: Damerau-Levenshtein, Jaro-Winkler e Needleman-Wunsch. Além disso, também foi avaliado o uso da computação paralela com o Apache Spark para melhorar o desempenho do ambiente.

Os testes realizados mostraram que a combinação das três funções de similaridade resultou em uma precisão significativamente maior na identificação de tuplas duplicadas. No entanto, também foi observado um aumento no tempo de processamento computacional, o que era esperado devido ao processamento adicional necessário para aplicar as funções adicionais. Apesar disso, em um contexto de Big Data, onde o volume de dados é muito grande, o uso de múltiplas funções de similaridade ainda é viável pois permite uma melhor precisão e alcance de contextos mais amplos nos dados, o que é crucial para a extração de conhecimento valioso.

Assim, é importante levar em conta a relação entre acurácia e custo computacional ao implementar esse método em um novo contexto. É preciso avaliar se a preocupação é a acurácia dos dados gerados ou um custo computacional menor. No entanto, é possível concluir que o uso de funções de similaridade em sequência e o uso de recursos de paralelização e processamento em memória são uma abordagem eficiente para melhorar o desempenho do sistema na detecção de registros duplicados.

5.1 Contribuição científica

O ambiente foi implementado com recursos de paralelização e processamento em memória com o Apache Spark, o que melhora o desempenho e a eficiência do algoritmo, que o torna mais adequado para o contexto Big Data. Com base nas análises de desempenho e qualidade realizadas, conclui-se que o uso de múltiplas funções de similaridade em conjunto resultou em melhores resultados de precisão, recall e F-measure, além de diminuir o tempo de execução. Isso sugere que a combinação de vários algoritmos proporciona uma abordagem mais completa e aumenta o desempenho na identificação de registros duplicados no conjunto de dados, como é observado na Tabela 5. Nela é apresentado que este trabalho tem como diferencial combinar sucessivas funções de similaridade para a identificação de tuplas duplicadas em um ambiente paralelizado. Essa contribuição científica é importante para a área de limpeza e tratamento de dados, pois ajuda a tornar os processos mais precisos e eficientes.

A análise dos resultados obtidos mostrou que a utilização de múltiplas funções de similaridade em sequência aumentou significativamente a precisão na identificação de tuplas duplicadas, mas houve um aumento no tempo de processamento computacional. Apesar disso, em um contexto de Big Data, o aumento no tempo de processamento é aceitável, pois a prioridade é a acurácia dos dados gerados. A contribuição científica deste trabalho é oferecer uma metodologia otimizada para a identificação de tuplas duplicadas que utiliza recursos de paralelização e funções de similaridade múltiplas, capaz de lidar com grandes volumes de dados e garantir uma precisão elevada nos resultados.

Tabela 5 – Comparativo entre os trabalhos correlatos a este trabalho

	(PEPE, 2019)	(BAYRAK, 2018)	(MAHDI, M. AHMAD, A. R. ISMAIL (2018)	Trabalho Atual
Estratégias propostas ao cenário *Big Data*				
Discussão sobre técnicas de similaridade textual				
Realizam a combinação de sucessivas funções de similaridade para otimizar o cálculo de similaridade				

Fonte: Elaborada pelo autor.

5.2 Atividades futuras

As atividades futuras podem incluir uma análise mais aprofundada das outras categorias e subcategorias de funções de similaridade disponíveis. Por exemplo, é possível investigar a eficácia de combinar funções de similaridade baseadas em token, corpus ou conhecimento, além das funções baseadas em strings e caracteres utilizados neste trabalho. Isso poderia ser feito ao usar técnicas de normalização de token, remoção de stopwords, stemming, entre outras, antes de aplicar as funções de similaridade. Também poderia ser realizado testes ao usar diferentes bases de dados, para verificar se os resultados obtidos são consistentes em diferentes contextos e tipos de dados.

Ademais, outras abordagens também poderiam ser exploradas, como o uso de modelos de aprendizado de máquina, para tentar identificar automaticamente quais combinações de funções de similaridade são as melhores para determinado conjunto de dados.

Assim, trabalhos futuros podem proporcionar insights adicionais sobre como combinar funções de similaridade de forma eficiente e otimizar o desempenho da identificação de tuplas duplicadas em contextos específicos.

REFERÊNCIAS

ELMAGARMID, A. K.; IPEIROTIS, P. G.; VERYKIOS, V. S. Duplicate Record Detection: A Survey. **IEEE Transactions on Knowledge and Data Engineering**, v. 19, n. 1, p. 1–16, jan. 2007.

MITRA, S.; ACHARYA T. Data mining : concepts and algorithms from multimedia to bioinformatics. **John Wiley & Sons, Inc.**, New York, NY, 2003.

ANDRADE, T. L.; SOUZA, R. C. G.; BABINI, M.; VALENCIO, C. R. Optimization of Algorithm to Identification of Duplicate Tuples Through Similarity Phonetic Based on Multithreading. Parallel and Distributed Computing, Applications and Technologies. **12th International Conference on**, 2011.

SILVEIRA, M.; MARCOLIN, C. B.; FREITAS, H. M. R.; Uso corporativo do Big Data: uma Revisão de Literatura. **Revista de Gestão e Projetos-GeP**, v. 6, n. 3, p. 44-59, 15 out. 2015.

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, 2017.

BHARATHI, B. et al. DUPLICATE RECORD DELETION IN RELATIONAL DATABASE MANAGEMENT SYSTEMS. **International Journal of Scientific & Engineering Research**, v. 8, n. 5, 2017.

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

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.

RIDZUAN, F.; WAN ZAINON, W. M. N. A Review on Data Cleansing Methods for Big Data. **Procedia Computer Science**, v. 161, p. 731–738, 2019.

AYE, T.T.; Web log cleaning for mining of web usage patterns., **IEEE 3rd International Conference on Computer Research and Development Shanghai**, 2011.

SETTE, B. S.; MARTINS, C. A.; Pré-processamento textual para a extração de informação em bases de patentes. **Anais da Escola Regional de Informática da Sociedade Brasileira de Computação (SBC)–Regional de Mato Grosso**, v. 7, 2016.

ALAM, S.; YAO, N. Big Data Analytics, Text Mining and Modern English Language. **Journal of Grid Computing.** 2018. p. 1-10.

VIJAYARANI, S.; ILAMATHI, Ms J.; NITHYA, Ms. Preprocessing techniques for text mining-an overview. **International Journal of Computer Science & Communication Networks**, 2015.

PRASETYA, D. D.; PRASETYA WIBAWA, A.; HIRASHIMA, T. The performance of text similarity algorithms. **International Journal of Advances in Intelligent Informatics**, 2018.

GOMAA, W. H.; FAHMY, A. A. A Survey of Text Similarity Approaches. **International Journal of Computer Applications**, 2013.

SINGH, D.; REDDY, C. K. A survey on platforms for big data analytics. **Journal of big data,** 2015.

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.

ALENAZI, S. R.; AHMAD, K. Record Duplication Detection in Database: **A Review. International Journal on Advanced Science, Engineering and Information Technology**, v. 6, n. 6, p. 838-845, 2016.

PEPE, L. M. Pré-processamento eficiente para limpeza de dados: Ambiente para identificação de tuplas duplicadas com recursos de paralelização. **Monografia de Conclusão de Curso para o curso Bacharel em Ciência da Computação - Instituto de Biociências, Letras e Ciências Exatas, Universidade Estadual Paulista Júlio de Mesquita Filho**, 2019.

CHAWLA, S., SHARMA, A. (2020). Apache Spark: A Comprehensive Overview. **International Journal of Computer Applications**, 2020.

PEREIRA, G. M., CARDOSO, A. A. Uma Análise Comparativa entre Tecnologias de Big Data para Processamento de Dados em Nuvem. **International Journal of Computer Applications**, 2020.

XU, Y.; WU, X. An Improved Data Cleaning Method Based on Similarity Calculation. **International Journal of Grid and Distributed Computing**, 2016.

HU, Y., LIAN, Y. A novel incremental duplicate record detection algorithm based on the successive similarity function. **International Journal of Advanced Computer Science and Applications**, 2018.

XU, J., WU, Y. (2016). Research on the detection of duplicate records based on the similarity function. **Journal of Computational Information Systems**, 2016.

MAHDI, M.; AHMAD, A. R.; ISMAIL, R. Similarity Search Techniques in Exploratory Search: A Review. **IEEE Transactions on Knowledge and Data Engineering,** 2018.

BAYRAK, A. T. et al. Near duplicate detection in relational databases. **IEEE Transactions on Knowledge and Data Engineering**, 2018.

ZHAO, C., SAHNI S., "Efficient computation of the Damerau-Levenshtein distance between biological sequences**," 2017 IEEE 7th International Conference on Computational Advances in Bio and Medical Sciences (ICCABS)**, Orlando, FL, USA, 2017, pp. 1-1, doi: 10.1109/ICCABS.2017.8114295.

WANG, Y., QIN, J., WANG, W. (2017). Efficient Approximate Entity Matching Using Jaro-Winkler Distance. In: , et al. **Web Information Systems Engineering – WISE 2017. WISE 2017. Lecture Notes in Computer Science(),** vol 10569. Springer, Cham.

A. E. E. -D. Rashed, H. M. Amer, M. El-Seddek and H. E. -D. Moustafa, "Sequence Alignment Using Machine Learning-Based Needleman–Wunsch Algorithm," **in IEEE Access, vol. 9, pp. 109522-109535, 2021**, doi: 10.1109/ACCESS.2021.3100408.