UNIVERSIDADE FEDERAL DO AGRESTE DE PERNAMBUCO

Manoel Gustavo Galindo de Sales

PROXCLUSTER: CLUSTERIZAÇÃO INCREMENTAL PARA RESOLUÇÃO DE ENTIDADES

Trabalho de Graduação

Universidade Federal do Agreste de Pernambuco
coordenacao.bcc@ufape.edu.br
bcc.ufape.edu.br/curso

Garanhuns-PE
2023

Universidade Federal do Agreste de Pernambuco
Graduação em Ciência da Computação

Manoel Gustavo Galindo de Sales

PROXCLUSTER: CLUSTERIZAÇÃO INCREMENTAL PARA RESOLUÇÃO DE ENTIDADES

Trabalho apresentado ao Curso de Graduação em Ciência da Computação da Universidade Federal do Agreste de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Orientadora: *Priscilla Kelly Machado Vieira Azevêdo*

Garanhuns-PE
2023

Dados Internacionais de Catalogação na Publicação (CIP)
Universidade Federal do Agreste de Pernambuco
Sistema Integrado de Bibliotecas (SIB-UFAPE)

Sales, Manoel Gustavo Galindo de
S163p ProxCluster: clusterização incremental para resolução de entidades / Manoel Gustavo Galindo de Sales. – Garanhuns, 2023.
61 f. : il. color.

Orientador(a): Priscilla Kelly Machado Vieira Azevêdo.
Monografia (Graduação) - Universidade Federal do Agreste de Pernambuco, Bacharelado em Ciência da Computação, Garanhuns, BR-PE, 2023.

Inclui referências.

1. Banco de Dados 2. Computação 3. Algorítmos computacionais 4. Banco de dados I. Azevêdo, Priscilla Kelly Machado Vieira (orient.) II. Universidade Federal do Agreste de Pernambuco III. Título

CDD 004

Elaborado por Talita Morgana Arruda (-)

Trabalho de conclusão de curso apresentado por **Manoel Gustavo Galindo de Sales** ao programa de Graduação em Ciência da Computação da Universidade Federal do Agreste de Pernambuco, sob o título **ProxCluster: Clusterização incremental para Resolução de Entidades**, orientado pela **Profa. Priscilla Kelly Machado Vieira Azevêdo** e aprovado pela banca examinadora formada pelos professores:

Kádna Maria Alves Camboim Vale
Universidade Federal do Agreste de Pernambuco
Membro Interno

Maria Aparecida Amorim Sibaldo
Universidade Federal do Agreste de Pernambuco
Membro Interno

Priscilla Kelly Machado Vieira Azevêdo
Universidade Federal do Agreste de Pernambuco
Professora Orientadora

Garanhuns-PE
2023

Eu dedico este trabalho a meus familiares e companheiros,
pelo apoio incondicional nesta jornada.

Agradecimentos

Aqui expresso meus sinceros agradecimentos a todos que contribuíram para a realização deste trabalho e para a conclusão bem-sucedida da minha jornada acadêmica.

Primeiramente, gostaria de agradecer aos meus pais, Juraci e Sales, que sempre estiveram ao meu lado, me incentivando e me dando a base necessária para chegar até aqui. À minha namorada, Luiza, agradeço por sua paciência e apoio incansável ao longo desta jornada.

Agradeço a todos os amigos feitos ao longo do curso, por tornar a jornada acadêmica mais prazerosa, em especial ao Allysson e Allana, também companheiros de apartamento, nossos momentos de estudo e risadas compartilhadas serão saudosas lembranças.

Ao Messi, cuja filosofia sempre me inspirou para não desistir e prosseguir com meus objetivos. A UFAPE e os professores que passaram por minha grade curricular, minha gratidão também por contribuírem de maneira única para minha formação, seus conhecimentos, orientações e *feedbacks* foram fundamentais para que eu pudesse crescer academicamente.

Em especial, à minha orientadora, Professora Priscilla, cuja orientação, sempre atenciosa, desde o início deste projeto até seu desfecho moldaram não apenas este trabalho, mas também minha abordagem à pesquisa.

“Não há vantagem alguma em viver a vida correndo.“

—SHIKAMARU NARA

Resumo

Quando dois registros de dados referem-se à mesma entidade no mundo real, porém possuem discrepâncias mínimas em alguns *bits* ou *bytes*, eles indicam a presença de dados duplicados. Nos dias atuais, o aumento da tecnologia e a explosão de dados têm apresentado desafios significativos para as organizações lidarem com a detecção e gestão de informações duplicadas provenientes de múltiplas fontes. A existência dessas duplicatas pode prejudicar a qualidade das consultas e complicar a manutenção dos dados. No processo de integração de dados, a Resolução de Entidades (RE) é responsável pelas tarefas de identificação de registros duplicados. Esta pesquisa teve como fundamento o Algoritmo Base, desenvolvido por estudantes na disciplina de Projeto de Banco de Dados da Universidade Federal do Agreste de Pernambuco (UFAPE), que utilizava conceitos do algoritmo K-Means para indentificar duplicatas na base de dados MusicBrainz. A partir do Algoritmo Base, este trabalho teve como objetivo trazer melhorias substanciais ao código inicial, como a refatoração para maior legibilidade e usabilidade, otimização de desempenho e implementação de uma abordagem incremental, tornando o algoritmo mais dinâmico e escalável. Dessa forma, este estudo concentra-se na identificação e clusterização, isto é, agrupamento das duplicatas, com o desenvolvimento do ProxCluster, *framework* modularizado para Resolução de Entidades Incremental (REI), resultante da aprimoração do Algoritmo Base. A estratégia incremental é particularmente relevante no cenário atual, permitindo que o agrupamento de duplicatas seja atualizado à medida que novos dados são inseridos, contrapondo-se aos métodos tradicionais que requerem reprocessamento completo sempre que há alterações. Este trabalho explora os algoritmos e estratégias desenvolvidos, bem como as metodologias e técnicas empregadas. Acredita-se que essa pesquisa contribuirá para avanços nesse campo, fornecendo uma solução de REI flexível e adaptável para a detecção de duplicatas em diversos contextos e domínios, aproveitando conceitos semelhantes ao K-Means. No que concerne a estratégia incremental, os experimentos aplicados em três bases de dados distintas, mostraram que a estratégia incremental do ProxCluster não somente acompanhou as alterações nas bases de dados, resultando em tempos menores para concluir a resolução em comparação com a estratégia estática, mas também manteve a qualidade dos clusters formados.

Palavras-chave: Resolução de Entidades Incremental, Algoritmo K-Means, Clusters, Banco de Dados.

Abstract

When two data records refer to the same entity in the real world but have minimal discrepancies in terms of bits or bytes, they indicate the presence of duplicate data. Nowadays, the increase in technology and the explosion of data have presented significant challenges for organizations to deal with the detection and management of duplicated information coming from multiple sources. The existence of these duplicates can impair query quality and complicate data maintenance. In the process of data integration, *Entity Resolution* (ER) is responsible for tasks related to the identification of duplicate records. This research was based on the Base Algorithm, developed by students in the Projeto de Banco de Dados course at UFAPE. This algorithm utilized concepts from the K-Means algorithm to identify duplicates in the MusicBrainz database. Building upon the Base Algorithm, this work aimed to bring substantial improvements to the initial code, including refactoring for improved readability and usability, performance optimization, and the implementation of an incremental approach, making the algorithm more dynamic and scalable. Thus, this study focuses on the identification and clustering, which is, grouping of duplicates, with the development of ProxCluster, a modularized framework for *Incremental Entity Resolution* (IER), resulting from the enhancement of the Base Algorithm. This work explores the developed algorithms and strategies, as well as the employed methodologies and techniques. It is believed that this research will contribute to advancements in this field, providing a flexible and adaptable IER solution for detecting duplicates in various contexts and domains, leveraging concepts similar to K-Means. Regarding the incremental strategy, experiments applied to three different databases showed that the ProxCluster Incremental strategy not only kept up with changes in the databases, resulting in shorter times to complete resolution compared to the static strategy but also maintained the quality of the formed clusters.

Keywords: Incremental Entity Resolution, K-Means Algoritmh, Clusters, Database.

Lista de Figuras

Lista de Tabelas

Lista de Siglas

Lista de Algoritmos

Sumário

1

Introdução

Nos últimos anos, o avanço da tecnologia e a geração de grandes volumes de dados, têm evidenciado os desafios das organizações para detectar e gerenciar duplicatas em múltiplas fontes de dados. Segundo THWEL; SINHA (2020), quando dois registros de dados fazem referência a mesma entidade no mundo real, mas diferem em alguns *bits* ou *bytes*, indicam a origem de dados duplicados. A presença de duplicatas pode comprometer a qualidade e a eficiência das consultas realizadas, bem como dificultar a atualização e manutenção desses dados. O campo de estudo que lida com esse problema pode ser apresentado como deduplicação de registros, reconciliação de registros, correspondência de registros, *matching* de registros ou integração de registros, mas aqui optaremos pelo uso da expressão Resolução de Entidades (RE).

Neste trabalho, foi realizado o aperfeiçoamento do Algoritmo Base, que por sua vez é uma adaptação do algoritmo K-Means estático SALES et al. (2023) desenvolvido por alunos na disciplina de Projeto de Banco de Dados, ministrada pela professora Priscilla Kelly Machado Vieira Azevêdo, na Universidade Federal do Agreste de Pernambuco (UFAPE). O K-Means é uma das opcões mais famosas e simples para clusterização AGGARWAL; REDDY (2013). Na disciplina, foi utilizado para identificar duplicatas presentes na base de música Music Brainz LEIPZIG (2019).

Com base nesse projeto inicial, foram realizadas melhorias significativas no algoritmo. Em particular, a refatoração do código, visando aprimorar sua legibilidade, usabilidade e torná-lo mais genérico e dinâmico, bem como otimizar o desempenho e conduzir mais avaliações e testes. Além disso, foi incorporada uma estratégia incremental para a clusterização das duplicatas, visando melhorar a eficiência e a escalabilidade do algoritmo, nomeado de ProxCluster Incremental.

Considerando o atual contexto tecnológico, onde os dados estão em constante evolução, a adoção da estratégia incremental foi ponto chave no desenvolvimento desse projeto, pois é uma abordagem que permite atualizar e expandir os resultados do agrupamento à medida que novos dados são adicionados ao conjunto de dados existente. Ao contrário dos métodos tradicionais de RE estáticos que requerem o processamento completo do conjunto de dados sempre que há uma alteração, desconsiderando, portanto, a característica dinâmica de múltiplas fontes de dados.

Ao longo deste documento, serão apresentados os algoritmos e estratégias desenvolvidas, as metodologias utilizadas, as técnicas empregadas, os resultados obtidos e possíveis direcionamentos futuros. Acredita-se que os resultados alcançados contribuirão para o avanço do campo de estudo de bancos de dados, mais especificamente no campo de RE, proporcionando uma nova solução flexível e adaptável para a detecção de duplicatas em diferentes contextos e domínios utilizando de estratégias similares ao K-Means, algoritmo de grande popularidade e simplicidade.

1.1 Motivação

A tarefa de identificar e eliminar entidades duplicadas é fundamental para manter a integridade dos dados, garantir a precisão das informações, melhorar a eficiência operacional e a experiência do usuário ou cliente. Considerando um ambiente com grandes volumes de dados distribuídos em múltiplas fontes de dados dinâmicas e autônomas THWEL; SINHA (2020) CHRISTEN (2012), é essencial ter um método eficiente e escalável para lidar com o problema de RE. Este é um cenário no qual estratégias incrementais se destacam como uma abordagem fundamental, trazendo benefícios significativos, tais como maior eficiência no processamento e manutenção da consistência dos dados.

Dada a relevância do problema de RE na área de bancos de dados e a popularidade e simplicidade do K-Means, que foi utilizado como base do algoritmo desenvolvido, fica aparente a importância e contribuição deste trabalho. Neste projeto, o algoritmo K-Means é adaptado e avaliado em dados não numéricos e de forma incremental.

1.2 Objetivo Geral

Este trabalho tem como objetivo detalhar a solução e o processo de desenvolvimento do *framework* ProxCluster que traz uma nova abordagem para clusterização de duplicatas utilizando da estratégia incremental. O algoritmo trouxe melhorias e contribuições ao projeto desenvolvido anteriormente na disciplina de Projeto de Banco de Dados da UFAPE que foi ponto de partida para esta pesquisa.

1.3 Objetivos Específicos

Com base no objetivo geral, correspondem os objetivos específicos indicados a seguir:

- Sintetizar os algoritmos desenvolvidos;
- Generalizar o algoritmo base desenvolvido durante a disciplina, a fim de melhorar sua usabilidade e desempenho;

- Reavaliar o algoritmo original, desenvolvido anteriormente. Verificando eficácia e eficiência;
- Adaptar o algoritmo original para a estratégia incremental de RE;
- Avaliar o algoritmo incremental quanto a sua eficácia e desempenho;
- Comparar as duas abordagens de clusterização, estática e incremental.

1.4 Organização do Texto

O presente trabalho foi assim constituído:

- O Capítulo 2 discorre sobre conceitos básicos dos principais fundamentos adotados neste trabalho e relacionados ao tema científico, em que são abordados os temas de RE e suas etapas, clusterização e K-Means, também é realizada uma apresentação de trabalhos relacionados ao tema, suas semelhanças e diferenças à solução proposta no presente projeto;
- No Capítulo 3, são apresentados os métodos utilizados no desenvolvimento da solução, os processos do desenvolvimento, as estratégias utilizadas, informações sobre os testes e avaliações realizadas;
- No Capítulo 4, são apresentados e analisados os resultados obtidos nas avaliações;
- O Capítulo 5 discorre sobre as considerações finais, principais contribuições e trabalhos futuros.

2

Referencial Teórico

Neste capítulo será apresentada base teórica do trabalho. Serão detalhados os conceitos relacionados a Resolução de Entidades e suas principais etapas, tais como Blocagem e Correspondência. Além disto, será contextualizado o entendimento de Resolução de Entidades Incremental. Também serão definidos temas-chave, como a Clusterização e um dos principais algoritmos de clusterização particional, o K-Means AGGARWAL; REDDY (2013). Ademais, este capítulo também realiza uma análise de similaridades e diferenças entre este trabalho e outros trabalhos da literatura.

2.1 Resolução de Entidades

Atualmente, vivencia-se uma era de avanço tecnológico em que várias aplicações lidam com grandes volumes de dados, impulsionando a necessidade de soluções automatizadas para manipulá-los de forma eficiente. Alguns exemplos desse cenário incluem setores como a saúde e medicina, onde aplicações utilizam dados médicos, registros de pacientes e informações de tratamentos para aprimorar os cuidados, gestão e conduzir pesquisas relevantes. Outro exemplo é o setor de *e-commerce*, que gerencia uma vasta quantidade de dados de produtos, transações e preferências dos clientes CHRISTOPHIDES et al. (2020); VIEIRA; LOSCIO; SALGADO (2017); CHRISTEN (2012).

Nesse contexto, é comum que fontes de dados heterogêneas e autônomas precisem ser integradas e, neste processo de integração de dados, alguns registros podem se referir à mesma entidade do mundo real. Para melhorar a qualidade desses dados, uma tarefa comum é a RE, cujo objetivo é identificar e agrupar registros que fazem referência às mesmas entidades do mundo real, seja dentro de uma única fonte ou entre diferentes fontes de dados CHRISTOPHIDES et al. (2020).

A Figura 2.1 a seguir apresenta o fluxo de RE, destacando os principais passos que serão discutidos com mais profundidade ao longo desta subseção.

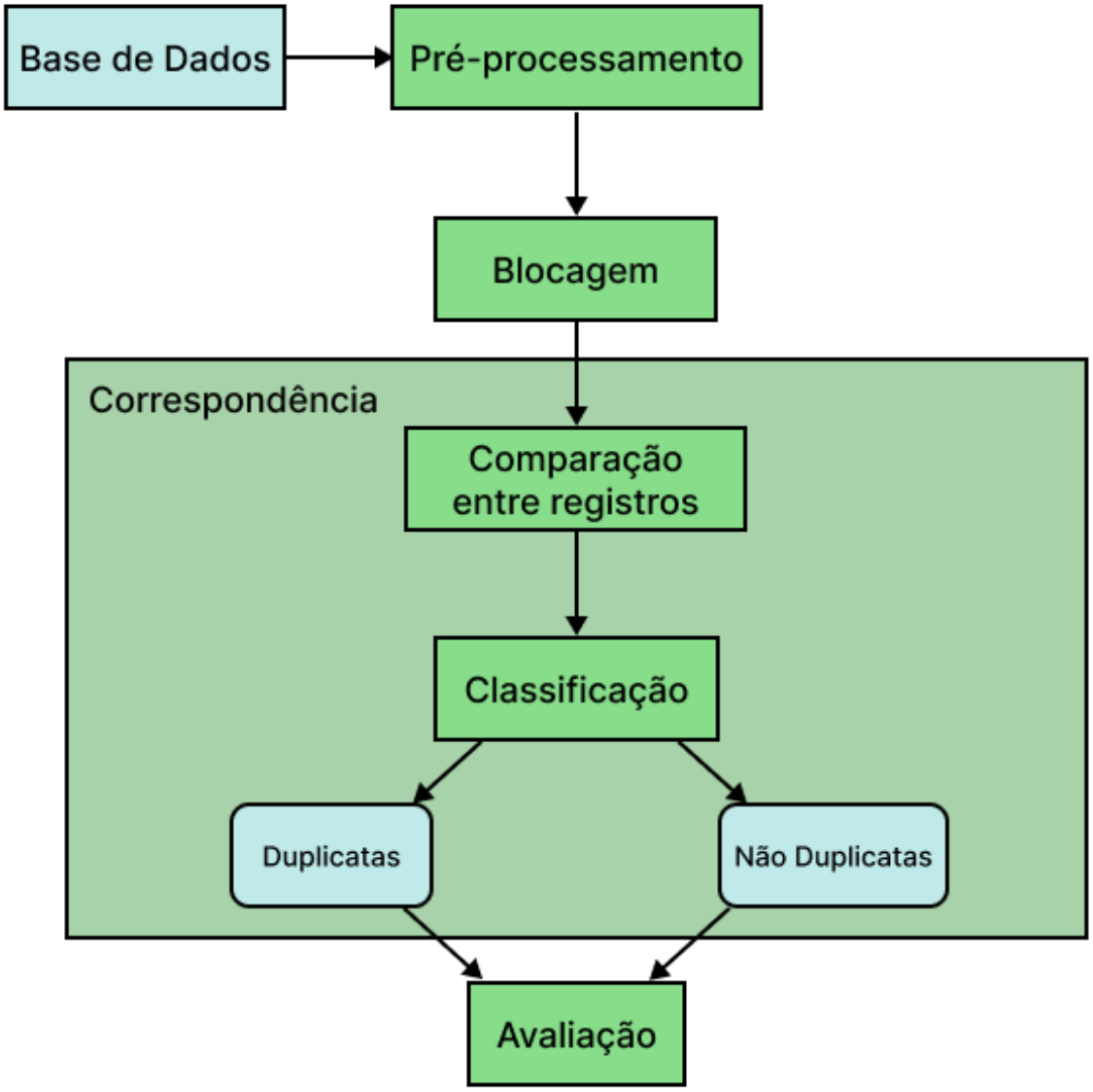

Figura 2.1: Fluxo da Resolução de Entidades
Fonte: Adaptado de CHRISTEN (2012)

Vamos considerar um exemplo de RE que envolva duas fontes de dados: Fonte A e Fonte B, que contêm informações sobre clientes de uma empresa. O objetivo é identificar os registros duplicados entre essas fontes para possibilitar a integração de dados das bases.

Previamente ao processo de RE, é necessário a etapa de Resolução de Esquema (*schema matching*), responsável pelo mapeamento dos atributos como nome, endereço, telefone e e-mail. Esses atributos podem ter formatos distintos entre as fontes, ou estarem distruibuídos entre outros atributos, por exemplo, o endereço na Fonte A consiste de vários atributos: rua, cep, cidade, estado e número. Todavia, na Fonte B é apenas um atributo textual contento todas essas informações. Esse primeiro passo é importante para que os registros possam ser comparados de forma adequada posteriormente no processo de RE.

Após a resolução de esquema, a RE inicia com a etapa de pré-processamento, conforme a Figura 2.1, passo importante para garantir que os registros estejam no formato desejado. É realizada a limpeza dos dados, o que inclui remover espaços em branco desnecessários, caracteres especiais e formatos inconsistentes, como abreviações ou variações na grafia de nomes e endereços ou campos faltando.

Após o pré-processamento, a Figura 2.1 segue com o processo de Blocagem (também conhecido como Indexação), que reduz o número de registros a serem comparados e enviados para a próxima etapa, denominada de Correspondência, que avalia a similaridade entre pares de registros e os classifica. Com o processo de Correspondência finalizado, as duplicatas encontradas são utilizadas para a avaliação e qualificação do resultado. Esses são os passos base para RE tradicional, e as etapas de Blocagem e Correspondência onde este trabalho focará, serão aprofundados em seguida.

2.1.1 Blocagem

Para identificar se pares de registros são duplicatas, é preciso que seja realizada a Correspondência entre eles, entretanto, segundo MCNEILL; KARDES; BORTHWICK (2012) realizar a comparação de todos os pares de registros de uma base A com os de uma base B é um processo quadrático quanto ao tamanho das bases a serem comparadas, na prática, essa estratégia ingênua é completamente inviável devido ao tamanho das fontes de dados. Para reduzir essa quantidade de comparações, técnicas de Blocagem são aplicadas. Os ganhos de desempenho são significativos, embora, esse processo possa aumentar o risco de falsos negativos, quando os métodos de blocagem utilizados acabam separando duplicatas que não corresponderam entre si aos critérios da blocagem.

Há diversos algoritmos para o processo de blocagem. Na abordagem tradicional, a base de dados é dividida em blocos de acordo com um ou mais critérios de blocagem, chamados de chave de blocagem. Dessa forma, apenas os registros pertecentes a um mesmo bloco, também referenciados como candidatos a duplicatas serão comparados entre si na etapa de Correspondência. Um exemplo simples seria utilizar o valor de um dos atributos da base como chave de blocagem, supondo que em uma base de dados com registros de pessoas, um dos atributos seja o CEP de cada indivíduo, e este campo seja identificado como um elemento bastante determinante como parâmetro de identificação das duplicatas, o valor do atributo de CEP, poderia ser utilizado como chave de blocagem, assim só seriam comparados registros de pessoas com o mesmo valor de CEP.

Entretanto, especialmente se tratando de campos com valores textuais, como nomes e endereços, é comum haver pequenas variações entre as duplicatas. Quando a abordagem simplória citada acima é aplicada em atributos textuais, a blocagem pode resultar na separação de registros em blocos diferentes ao encontrar qualquer variação tipográfica. Para resolver essa questão, algoritmos de codificação fonética podem ser empregados. Esses algoritmos são

técnicas utilizadas para representar palavras ou nomes de forma a capturar sua pronúncia ou som, independentemente da sua grafia CHRISTEN (2012). Um exemplo de agoritmo de codificação fonética é o Phonex, utilizando a disponibilização online da implementação da biblioteca[1] na linguagem R, é possível testar de forma prática que os nomes "smith" e "smythe" quando utilizados como entrada para o algoritmo, embora possuam variações tipográficas, resultam no mesmo código: s530. Logo, ao utilizar do código Phonex nos valores do atributo nome como chave de blocagem, ambos os registros com valor de nome "smith" e "smythe" seriam encaixados no mesmo bloco.

2.1.2 Correspondência

A etapa de Correspondência irá receber os registros em blocos resultantes do processo de blocagem e realizará dois processos, a Comparação entre Registros para calcular a similaridade entre eles, e em seguida a Classificação dos Registros como duplicatas ou não duplicatas, assim, retornando-os.

2.1.2.1 Comparação entre Registros

Para que seja determinado se um par de registros é duplicado ou não, é necessário que seja comparado de forma detalhada. Idealmente, mais atributos e estratégias mais robustas do que as utilizadas na blocagem serão levados em consideração para que seja possível calcular a semelhança entre os registros, no que é chamado de função de Similaridade. Usualmente, a função de similaridade retornará os valores de forma normalizada, por exemplo, valores de similaridade 1.0 são de registros com correspondência total entre si, de maneira análoga, similaridade de 0.0 são referentes a registros completamente dissimilares, já qualquer valor entre esses limites representa o grau de similaridade entre o par CHRISTEN (2012) GETOOR; MACHANAVAJJHALA (2012).

Devido aos diferentes tipos de dados presentes nos diversos atributos da base, é necessário o uso de diferentes funções de comparação de similaridade, resultando no que é chamado de Vetores de Comparação. Para atributos como datas, tempos, idades, tipos numéricos, diversas estratégias podem ser utilizadas a depender do cenário, como comparação de igualdade, diferença entre os valores ou funções mais sofisticadas podem ser empregadas. Já para os atributos textuais, também há diversas opções, como é o caso dos algoritmos de codificação fonética (citados anteriormente), aonde pode-se realizar uma comparação exata das codificações fonéticas, ou comparar os códigos fonéticos em si usando um algoritmo de correspondência de padrões para obter uma correspondência aproximada CHRISTOPHIDES et al. (2020) CHRISTEN (2012) CHRISTEN (2008).

Outra estratégia para funções de comparação de similaridade em atributos textuais são as técnicas de Correspondência de padrões, como a distância Levenshtein, método utilizado

[1] Implementação Phonex: `https://rdrr.io/cran/phonics/man/phonex.html`

para calcular a distância entre dois textos. Esse algoritmo mede o número mínimo de operações (inserção, deleção ou substituição de caracteres) necessárias para transformar um texto no outro. CHRISTEN (2006).

Neste trabalho, utilizamos a função de distância Levenshtein para comparação de atributos textuais nesta etapa de correspondência do processo de RE.

2.1.2.2 Classificação dos Registros

Tradicionalmente, nessa etapa, será realizada a classificação dos registros como duplicatas ou não baseado nos valores dos Vetores de Comparação ou na soma de suas similaridades que será comparada com um dado limiar mínimo para que os registros sejam classificados como a mesma entidade ou não CHRISTOPHIDES et al. (2020).

No âmbito deste trabalho, a clusterização foi utilizada para identificar as duplicatas. Nessa abordagem, todos os registros que pertencem ao mesmo cluster são considerados representantes de uma mesma entidade no mundo real. Essa estratégia permite simplificar a visualização e análise dos dados, facilitando a manipulação e tratamento das duplicatas.

2.1.2.3 Clusterização

De acordo com MAIMON; ROKACH (2010), a clusterização de objetos é tão antiga quanto a necessidade humana de descrever as características salientes de pessoas e objetos e identificá-los com um tipo. O objetivo da clusterização é descrever e agrupar os dados de forma a identificar conjuntos semelhantes, dessa forma, permitindo a organização eficiente das instâncias de dados em uma representação mais compacta, que ajuda a caracterizar a população estudada. Considerando a base de dados S, podemos descrever formalmente a estrutura de clusterização como um subconjunto $C = C_1, C_2, ..., C_k$ de S, de forma que $S = \bigcup_{i=1}^{k} C_i$ e $C_i \cap C_j = \emptyset$ para $i \neq j$. Assim, garantindo que qualquer instância de S pertence a apenas um e exatos um subconjunto.

A clusterização nos ajuda a entender a estrutura e a formar grupos distintos, um processo tão importante que abrange várias disciplinas científicas, como biologia, genética, medicina, entre outras. Por exemplo, ao realizar a clusterização em um conjunto de dados sobre consumidores, podemos descobrir grupos de pessoas com comportamentos de compra similares. Ao analisar dados genéticos, a clusterização pode ajudar a identificar diferentes grupos de indivíduos com características genéticas semelhantes MAIMON; ROKACH (2010). Neste trabalho é utilizada para o agrupamento dos registros duplicados. Dados os clusters C da base de dados S, o subconjunto $C_i \in S$, possue apenas registros que referenciam a mesma entidade do mundo real.

A categoria de clusterização particional está entre os mais estudados dentre os algoritmos de clusterização. Esses algoritmos têm sido amplamente utilizados em uma variedade de aplicações, principalmente devido à sua simplicidade e facilidade de implementação em relação a outros algoritmos de clusterização. Os algoritmos de clusterização particional têm como

objetivo descobrir os agrupamentos presentes nos dados, otimizando uma função objetivo específica e melhorando iterativamente a qualidade das partições. Eles requerem um conjunto de sementes iniciais (ou clusters) que são aprimorados iterativamente. Um dos algoritmos de clusterização mais simples e eficientes propostos na literatura de clusterização de dados é o K-Means AGGARWAL; REDDY (2013) MAIMON; ROKACH (2010).

O K-Means é uma técnica de clusterização que começa selecionando K pontos representativos como centróides iniciais. Cada ponto de dados é então atribuído ao centróide mais próximo com base em uma medida de proximidade específica escolhida, como a distância Euclidiana ou Manhattan. Uma vez que os clusters são formados, os centróides de cada cluster são atualizados. Esse processo é repetido iterativamente, com os pontos sendo reatribuídos aos centróides mais próximos a cada iteração e os centróides sendo recalculados a partir das novas atribuições. O algoritmo continua a repetir essas etapas até que os centróides não mudem significativamente ou até que outro critério de convergência seja atendido AGGARWAL; REDDY (2013) MAIMON; ROKACH (2010).

Conforme ilustrado pelo pseudo-código do Algoritmo 1, o K-Means recebe como entrada o conjunto de registros D e uma quantidade K de clusters. Na linha 1, temos a inicialização dos K centróides. Logo em seguida, inicia-se o laço de repetição, e nas linhas 3 e 4, respectivamente, cada registro t_i será atribuído ao cluster cujo centróide tem a menor distância para ele, e uma vez que os clusters são formados, os centróides de cada cluster são atualizados. Esse processo é repetido iterativamente, até que a condição de parada é atingida (linha 5). Por fim, é retornado o conjunto de clusters K.

Algorithm 1 Pseudo-código do Algoritmo K-Means

Input: $D = \{t1, t2 \ldots Tn\}$ ▷ Conjunto de registros
Input: K ▷ Quantidade de clusters
Output: K ▷ Conjunto de clusters

1: Atribua os valores iniciais para $m1, m2, \ldots m_K$
2: **repeat**
3: atribua cada item t_i ao cluster com o centróide mais próximo;
4: calcula o novo centróide para cada cluster;
5: **until** critério de convergência é atingido;

Fonte: Adaptado de MOHD et al. (2012)

O cálculo do centróide de cada cluster é realizado tomando-se a média de todas as instâncias pertencentes a esse cluster:

$$\mu_k = \frac{1}{N_k} \sum_{q=1}^{N_k} x_q$$

Onde N_k é a quantidade de instâncias pertences ao cluster k e μ_k é média do cluster k.

Várias condições de convergência podem ser utilizadas para determinar quando o algoritmo deve parar. Por exemplo, a busca pode parar quando o erro de particionamento não é mais reduzido com a realocação dos centróides, indicando que a partição atual é localmente ótima. Outros critérios de parada também podem ser empregados, como exceder um número pré-definido de iterações MAIMON; ROKACH (2010).

Em resumo, o K-Means é um algoritmo simples e eficiente de clusterização que agrupa os pontos de dados em clusters, procurando minimizar a variação dentro dos clusters e maximizar a separação entre eles. Ele é amplamente utilizado em aplicações de aprendizado de máquina e mineração de dados para identificar padrões e estruturas em conjuntos de dados não rotulados.

Neste trabalho, características do K-Means foram utilizadas como base para o desenvolvimento do algoritmo ProxCluster, cujo funcionamento será explicado no Capítulo 3.

2.2 Resolução de Entidades Incremental

Tradicionalmente, os processos de RE são feitos de maneira estática, também conhecido como *offline*, processando todos os dados disponíveis de uma só vez. Entretanto, as bases de dados no contexto atual são usualmente dinâmicas e em constante evolução, o que inviabiliza a estratégia tradicional em cenários como esses, pois é necessário realizar o processo de RE do completo zero no conjunto de dados sempre que há uma alteração, o que torna o processo extremamente custoso VIEIRA; LOSCIO; SALGADO (2019).

Em uma era de grandes volumes de dados, é essencial adotar um método eficiente e escalável para a RE. A estratégia incremental destaca-se como fundamental, trazendo benefícios como maior desempenho no tempo de execução do processamento, consistência dos dados e capacidade de lidar com múltiplas fontes de dados dinâmicas e autônomas. Essa abordagem permite acompanhar as mudanças nos dados em tempo real, garantindo precisão e atualização contínua dos resultados, reduzindo significativamente os custos gerais da RE THWEL; SINHA (2020); CHRISTOPHIDES et al. (2020); VIEIRA; LOSCIO; SALGADO (2019).

Tendo em vista que uma das etapas mais custosas do processo de RE é a Comparação entre Registros, um método-chave no contexto de Resolução de Entidades Incremental (REI) é o reuso dos resultados de iterações anteriores da resolução, o que permite evitar o reprocessamento completo de todo o conjunto de dados. Outro método também utilizado é realizar o processo de RE apenas em resultados de consultas (*queries*) sobre as bases de dados, a medida que são necessários para o usuário VIEIRA; LOSCIO; SALGADO (2019).

Considere um exemplo prático de aplicação de REI em um contexto de gerenciamento de publicações científicas de universidades parceiras de uma editora acadêmica. A editora, armezena as publicações de forma centralizada, entretanto, devido a submissão de publicações ser enviada de diversas fontes, é aplicado um processo de RE nas publicações da editora. Se não for adotada a estratégia de REI, sempre que novas submissões forem enviadas das universidades

para a editora, será necessário, realizar a resolução do completo zero na base de dados.

2.3 Trabalhos Relacionados

O problema de RE é um tópico já discutido por várias outras pesquisas. Destaca-se, inclusive, a importância de direcionar a atenção à otimização da velocidade de execução dessa tarefa, considerando a magnitude crescente das bases de dados atualmente. Assim como em estudos anteriores que abordaram soluções para RE, nesta subseção é realizada uma análise de similaridades e diferenças entre este trabalho e outros trabalhos da literatura.

Em DRAISBACH; NAUMANN (2010), é discutido RE apresentando algoritmos e técnicas desenvolvidos para melhorar a eficiência e eficácia do processo de identificação de registros duplicados. Nele, é apresentado o *duplicate detection toolkit* "DuDe", uma ferramenta modular para RE que pode ser estendida com novos algoritmos e componentes. Além disso, eles fornecem vários algoritmos, medidas de similaridade e conjuntos de dados com padrões de referência, cumprindo os requisitos para um *benchmarking* de RE. Suas contribuições incluem uma *toolkit* modular e conjuntos de dados com descrições detalhadas para uso como *benchmarking*, além da comparação de desempenho do DuDe com o Febrl[2], outro *framework* de RE.

Apesar das semelhanças do presente trabalho com o DuDe, no que concerne o desenvolvimento de um *framework* com módulos para RE com foco na usabilidade, o DuDe, embora possua mais ferramentas e componentes ofertados, não possui nenhuma solução integrada para REI.

Em VIEIRA; LOSCIO; SALGADO (2019), é apresentado o QUIPER, uma proposta de REI com o objetivo de reduzir custos, reutilizando resultados anteriores e processando apenas um conjunto selecionado de registros em cada iteração. A abordagem propõe um processo que utiliza clusterização para classificar pares de registros a partir de resultados de consultas. Cada iteração lida com um conjunto de registros provenientes de consultas executadas em múltiplas fontes de dados. O processo RE é realizado durante consultas, assumindo uma integração virtual de dados. A proposta foi testada com algoritmos de clusterização tradicionais, Single-Link e Average-Link, apresentando resultados de avaliação em fontes de dados reais e sintéticas.

O trabalho realizado em VIEIRA; LOSCIO; SALGADO (2019) se assemelha ao presente como uma solução de REI que utiliza de clusterização, entretanto, enquanto o ProxCluster, *framework* proposto nesse trabalho, realiza o processo de RE sobre toda a base de dados disponível, retornando-a completamente clusterizada, o QUIPER realiza o processo de RE nos resultados de consultas executadas nestas bases de dados.

Em PAPENBROCK; HEISE; NAUMANN (2015), são propostos dois algoritmos de RE progressivos, onde são identificadas a maioria dos pares de duplicatas no início do processo de detecção. Nas estratégias progressivas, em vez de reduzir o tempo geral necessário para concluir

[2]Febrl: `https://sourceforge.net/projects/febrl/`

todo o processo, procuram reduzir o tempo médio após o qual uma duplicata é encontrada, resultando em resultados mais completos do que abordagens tradicionais devido ao término precoce. O trabalho também aponta quais casos de uso melhores se encaixam para as soluções propostas, a exemplo de situações onde o tempo de execução é limitado.

Os algoritmos *Progressive Sorted Neighborhood Method* (PSNM) e o *Progressive Blocking* (PB), propostos por PAPENBROCK; HEISE; NAUMANN (2015), utilizam de estratégias progressivas, onde buscam indentificar a maioria das duplicatas no início da execução, enquanto, possivelmente aumentam ligeiramente o tempo de execução geral. O *framework* ProxCluster, entretanto, não possui nenhuma estratégia progressiva, identificando as duplicatas com frequência quase constante. Além disso, o ProxCluster possui uma estratégia incremental, o que possibilita a reutilização dos resultados computados anteriormente.

3

Proposta e Desenvolvimento do ProxCluster

Neste capítulo é apresentado detalhes da metodologia e desenvolvimento do ProxCluster, representando todo o *framework* de RE desenvolvido. Também será explorado o algoritmo base que foi aproveitado para o desenvolvimento do projeto, onde após a etapa de refatoração foi generalizado no algoritmo ProxCluster Estático que por sua vez foi estendido para possibilitar o uso de maneira incremental, ProxCluster Incremental. Além disso, será explorado um exemplo motivacional para melhorar o entendimento dos processos do ProxCluster. O foco deste capítulo recai sobre o desenvolvimento do projeto em sua plenitude, permitindo uma análise minuciosa dos métodos utilizados, das estratégias e abordagens empregadas.

3.1 Métodos Utilizados

Nesta seção, descreveremos os métodos utilizados na elaboração desta pesquisa. A base do trabalho foi estabelecida através de um estudo prévio realizado na disciplina de Projeto de Banco de Dados. Essa etapa proporcionou uma compreensão inicial dos conceitos e técnicas relevantes para a RE, ajudando a definir os objetivos do presente trabalho, visando aprimorar o que foi realizado anteriormente.

Além do embasamento do trabalho prévio, foi realizada uma extensa revisão bibliográfica. Foram consultados artigos científicos, periódicos e livros relacionados à RE e o processo incremental, incluindo estudos recentes e trabalhos de referência na área. Essa revisão permitiu a compreensão do estado da arte, das técnicas e das métricas utilizadas para o desenvolvimento e avaliação de eficácia dos algoritmos de RE.

Com base no conhecimento adquirido por meio do trabalho prévio, e da revisão bibliográfica, foram definidos os Objetivos Específicos. O algoritmo foi concebido levando em consideração as abordagens existentes, boas práticas e técnicas identificadas na literatura.

A avaliação do algoritmo proposto foi conduzida utilizando métricas consideradas relevantes para a avaliação de sistemas de RE. As métricas escolhidas foram aquelas recomendadas pelo CHRISTEN (2012), referência na área, e incluíram a precisão, *recall* e medida F, além, claro do tempo de execução. Essas métricas foram adotadas para mensurar tanto a capacidade

do algoritmo em identificar corretamente as entidades, quanto sua habilidade de evitar falsos positivos.

Para garantir a validade dos resultados obtidos, um planejamento experimental foi realizado. Esse planejamento envolveu a definição de diferentes bases de dados para os experimentos, comparação das diferentes estratégias desenvolvidas, estática e incremental, como também, experimentos comparando o ProxCluster com outros algoritmos. Essa abordagem metodológica possibilitou uma análise criteriosa do desempenho do algoritmo proposto e sua comparação com o projeto desenvolvido previamente e outras abordagens existentes na literatura e meio tecnológico.

Na elaboração deste projeto, foram empregadas tecnologias e ferramentas essenciais que permitiram o desenvolvimento eficaz e a execução dos objetivos propostos. A linguagem de programação escolhida foi o Python, na versão 3.10, reconhecida por sua versatilidade e ampla gama de bibliotecas de suporte. Dentro do ecossistema Python, destacam-se as bibliotecas utilizadas: o Pandas[1] e Polars[2] para manipulação e análise de dados, e as bibliotecas python-Levenshtein[3] e Fuzzy[4], que desempenharam um papel crucial na comparação e avaliação de similaridade entre elementos textuais.

Quanto ao ambiente de desenvolvimento, optou-se pelo Visual Studio Code (VSCode), uma ferramenta amplamente adotada pela comunidade de desenvolvedores devido à sua interface amigável, extensibilidade e suporte integrado para Jupyter Notebooks. A utilização de Notebooks trouxe agilidade no processo de prototipagem e experimentação, permitindo a visualização passo a passo do código e dos resultados obtidos.

3.2 Procedimento de Pesquisa

Esta seção será responsável por detalhar os procedimentos tomados para o desenvolvimento do presente trabalho.

A Figura 3.1 representa o fluxo de etapas para o desdobramento da pesquisa. Tomando partido do algoritmo base, cujos detalhes serão esclarecidos na subseção 3.2.1, foi realizado a Refatoração do código, etapa importante para que seja possível desenvolver as novas funcionalidades. Após a etapa de Refatoração do código base, resultando no ProxCluster Estático, foi dado início a implementação das funcionalidades novas, com início no desenvolvimento da Nova Blocagem, que utiliza outra biblioteca para gerar as chaves de blocagem, e a partir daí, a adição do método incremental para o ProxCluster 3.2.5, ponto chave do projeto. Por fim, foram definidos e desenvolvidos os métodos de avaliação do algoritmo, possibilitando a realização de diversos experimentos, cujos resultados, o capítulo 4 detalhará de maneira aprofundada.

[1]Pandas: `https://pandas.pydata.org/`
[2]Polars: `https://www.pola.rs/`
[3]python-Levenshtein: `https://pypi.org/project/python-Levenshtein/`
[4]Fuzzy: `https://pypi.org/project/Fuzzy/`

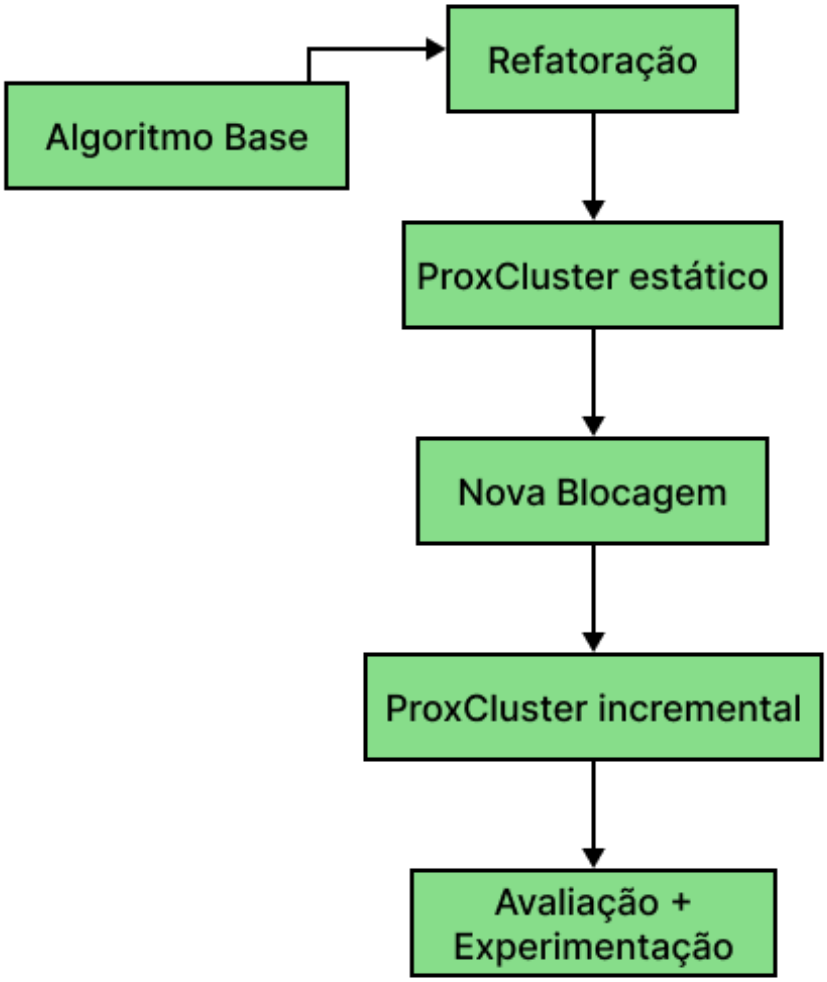

Figura 3.1: Fluxo do Procedimento de pesquisa
Fonte: Autoria própria

3.2.1 Algoritmo Base

Como já mencionado, neste trabalho foi realizado o aperfeiçoamento do algoritmo desenvolvido na disciplina de Projeto de Banco de Dados SALES et al. (2023), ministrada pela professora Priscilla Kelly Machado Vieira Azevêdo, na UFAPE. Portanto, esta seção será dedicada a discorrer sobre o funcionamento daquele que foi ponto de partida para o desenvolvimento do ProxCluster.

Foi decidido pelos alunos, como proposta para o projeto da disciplina de Projeto de Banco de Dados, utilizar o algoritmo K-Means para identificação e clusterização de duplicatas de uma base dados, utilizando a linguagem Python. A base escolhida foi a Music Brainz LEIPZIG (2019). Ela possui registros de músicas reais da base MusicBrainz[5], que é uma enciclopédia musical aberta que coleta metadados musicais e os disponibiliza ao público. A base utilizou o

[5]MusicBrainz - Enciclopédia Aberta de Música: `https://musicbrainz.org/`

gerador de dados DAPO[6] para criar duplicatas com valores de atributos modificados. O conjunto de dados gerado consiste em cinco fontes e contém duplicatas para 50% dos registros originais. Alguns dos atributos da base são: o título, álbum, nome do artista ou banda, ano de publicação, duração da música e o número da faixa em relação ao álbum.

Conforme o projeto foi sendo desenvolvido, foi identificado que devido aos tipos dos atributos da base de dados escolhida serem em maioria textuais, inviabilizaria a utilização do K-Means, que utiliza funções de distâncias numéricas, como a Euclidiana ou Manhattan. Portanto, foram desenvolvidos uma função de distância customizada e um algoritmo inspirado no K-Means para a clusterização da base Music Brainz.

Quanto ao método desenvolvido para realizar a blocagem dos dados da base, foi calculado a chave de blocagem do título dos registros utilizando a biblioteca Phonex[7] implementada em Python. Assim, com todos os registros calculados o valor "phonex"que varia de 0 a 1, a base de dados é então ordenada com base nesses valores de chave de blocagem, também conhecidos como *Blocking Key Value* (BKV), e como resultado, as músicas com títulos que possuem uma pronúncia semelhantes ficam próximas umas das outras. Por fim, a base é dividida em blocos de tamanhos iguais, tamanho esse, enviado como parâmetro. Dependendo do tamanho escolhido para os blocos, é possível que o último bloco tenha menos registros do que os outros (quando a divisão do número total de registros da base pelo tamanho dos blocos não é exata), ficando com o restante.

Para metrificar a distância entre os registros, foram considerados três atributos textuais, título, álbum e artista, e dois atributos numéricos, ano e número da faixa. Para os atributos textuais, foi utilizado a distância Levenshtein normalizada entre 0 e 1, e para as distâncias dos anos e dos números das faixas foram calculadas retornando distância de 0 se forem iguais e 1 (distância máxima) caso contrário. A distância retornada é a média ponderada das distâncias dos atributos dos registros fornecidos, quanto aos pesos, foram ajustados com experimentação, visando aumentar a precisão das classificações. Foram atribuídos os seguintes pesos aos atributos: título (1.2), álbum (1), artista (1), ano (0.5) e número da faixa (0.8).

3.2.2 Refatoração

A refatoração do código do Algoritmo Base foi uma etapa muito importante para o continuamento do trabalho. A base do projeto precisa estar bem firmada para que as novas funcionalidades e etapas sejam implantadas da melhor maneira. A refatoração teve o objetivo de melhorar a legibilidade e usabilidade do código, torná-lo mais genérico e dinâmico, além de otimizar seu desempenho. O repositório[8] contendo o código do presente trabalho pode ser acessado pelo link no rodapé.

O código base apresentava características de baixa legibilidade, dificultando a compreen-

[6]DAPO: `https://ieeexplore.ieee.org/abstract/document/7809119`
[7]Phonex Python: `https://github.com/damiencorpataux/phonex`
[8]Repositório ProxCluster: `https://github.com/Gust4voSales/proxcluster-deduplicator`

são do seu funcionamento e manutenção. Todas as etapas e lógicas da RE, do pré-processamento a avaliação, estavam em um mesmo arquivo, além disso, o código possuia diversas variáveis globais, funções acopladas umas com as outras, entre outros pontos que precisavam ser corrigidos. Todo o código do Algoritmo Base foi pensado para resolver apenas um problema, a RE da base Music Brainz para a atividade da disciplina de Projeto de Bando de Dados, portanto, era menos genérico e reutilizável. Com o intuito de resolver essas questões, o processo de refatoração foi realizado.

Segundo THORBEN (2023), o *Single Responsibility Principle* (SRP), também conhecido como Princípio da Responsabilidade Única, é um dos princípios fundamentais da programação orientada a objetos. Ele estabelece que uma classe deve ter apenas uma única responsabilidade. De acordo com o SRP, uma classe deve ter um único propósito e ser responsável por executar uma única tarefa específica dentro do sistema. Isso significa que a classe deve conter apenas os métodos e atributos necessários para cumprir essa responsabilidade, evitando que ela se torne excessivamente complexa e difícil de entender.

Para que fosse possível aplicar os conceitos do SRP, primeiro, foi necessário desacoplar o código em classes que foram criadas para as diferentes etapas que o algoritmo é responsável, as classes PhonexStaticBlocking e SoundexBlocking para blocagem, a ProxCluster que identifica e clusteriza as duplicatas, a classe Evaluator para a avaliação, e porteriormente, a classe Deduplicator, responsável por abstrair todo o processo de RE realizado pela combinação das classes citadas anteriormente, possibilitando assim, uma utilização mais alto nível e simplificada do algoritmo, contribuindo, também, para o ponto de usabilidade.

O conceito de Legibilidade refere-se à qualidade do código que permite que ele seja facilmente compreendido e interpretado pelos desenvolvedores. Um código legível é aquele que segue boas práticas de escrita, tornando-o claro, organizado e de fácil entendimento OLIVEIRA et al. (2020). Para atender a esse tópico, foram aplicados alguns conceitos para melhorar a legibilidade do código. Isso incluiu a adoção de nomes significativos para variáveis, métodos e classes, além da organização e indentação adequadas do código-fonte. Com essas melhorias, o código tornou-se mais compreensível, facilitando o entendimento de sua estrutura e funcionalidades.

Com o objetivo de tornar o código mais genérico e dinâmico, foram realizadas alterações estruturais, como a criação de parâmetros nos métodos e classes, para que o uso fosse customizável ao usuário, permitindo, assim, a reutilização dos componentes para diferentes bases de dados e até, possivelmente, diferentes contextos. Com isso, foi possível desenvolver um código mais flexível, adaptável a diferentes necessidades e extensível para futuras implementações. Essa abordagem também contribui para reduzir a duplicação de código e melhorar a manutenibilidade do sistema.

Durante a refatoração, foram realizadas otimizações no código para melhorar o desempenho do sistema. Isso incluiu a identificação e correção de trechos de código ineficientes, ou até mesmo, inúteis. Essas melhorias resultaram em um sistema mais rápido. Além disso, o processo de SRP também contribuiu para o desempenho do sistema ao separar processos de

etapas diferentes do mesmo contexto de execução.

Ainda sobre desempenho, outro ponto importante foi a migração da biblioteca utilizada para lidar com as tabelas e dados. Inicialmente, o código base utilizava a biblioteca Pandas, amplamente conhecida no ecossistema Python para manipulação e análise de dados. Embora o Pandas seja uma biblioteca poderosa e versátil, percebeu-se a oportunidade de melhorar significativamente o desempenho e a eficiência do algoritmo ao migrar para outra biblioteca.

A biblioteca Polars é uma alternativa de alta performance para manipulação de dados tabulares, especialmente adequada para grandes conjuntos de dados. O Polars é escrito na linguagem compilada Rust, e disponibiliza uma API para Python, o que permite combinar a eficiência do Rust com a familiaridade e facilidade de uso do Python.

A decisão de migrar para o Polars foi impulsionada por resultados consistentes encontrados em diversos artigos que compararam o desempenho entre as bibliotecas Pandas e Polars. Esses estudos revelaram que, na maioria dos testes realizados, o Polars apresentava uma performance significativamente superior LUNA (2023); CHAUDHARY (2023). Com base nessas análises, ficou claro que a migração para o Polars seria uma medida estratégica para otimizar o desempenho do algoritmo.

A migração para o Polars envolveu uma revisão cuidadosa do código que tratava da manipulação de dados, substituindo as chamadas ao Pandas pelas correspondentes operações no Polars. Embora o Polars tenha uma API similar ao Pandas, algumas adaptações foram necessárias, e a migração foi bem-sucedida.

A Figura 3.2 mostra o resultado de um teste realizado comparando os tempos de execuções das implementações do algoritmo ProxCluster com Pandas e Polars, respectivamente. A base utilizada foi a CD Information[9] de 9763 itens, a mesma será melhor explorada no capítulo 4. É possível observar que enquanto a implementação com Pandas levou 29.7s, a implementação com Polars executou aproximadamente três vezes mais rápido. Essa melhoria foi essencial para o desempenho geral do sistema, especialmente ao lidar com grandes volumes de dados.

[9]CD Information: `https://hpi.de/naumann/projects/repeatability/datasets/cd-datasets.html`

Figura 3.2: Comparação do tempo de execução: ProxCluster implementado com Pandas vs Polars

Fonte: Autoria própria

3.2.3 ProxCluster Estático

Nesta seção, será explicado o funcionamento do algoritmo resultante após etapa de Refatoração do que foi desenvolvido para lidar com as etapas de comparação, classificação e clusterização dos registros da base de dados Music Brainz.

O Algoritmo 2 tem a representação do pseudo-código ProxCluster estático, cujo entradas são: D (os registros à serem deduplicados), $distanceFn$ (função de distância) e o *threshold* (limite superior de distância). Nas linhas 1 a 3, o algoritmo inicia selecionando o primeiro registro como um centróide, isto é, o elemento representante daquele cluster. Em seguida, os registros são percorridos (linha 4) e a distância do registro atual da iteração para todos os centróides é calculada utilizando a função de distância definida anteriormente, linhas 6 à 13. Se a distância do centróide for de fato menor que o *threshold*, o registro atual é considerado uma duplicata do centróide correspondente àquela menor distância, e será adicionado ao cluster desse centróide. Caso contrário, se o elemento não for considerado uma duplicata de nenhum centróide (verificação da linha 14) ou seja, mesmo a menor distância desse registro para os centróides, é acima do *threshold*, ele é definido como um novo centróide, conforme a linha 15, e na linha posterior, a lista de centróides é atualizada com o novo centróide, por fim, a iteração continua. Esse processo é repetido até que todos os registros passados como entrada para o algoritmo tenham sido percorridos e é finalizado retornando os clusters.

Algorithm 2 ProxCluster Estático

Input: $D, distanceFn, threshold$

Output: C ▷ clusters

```
1:  firstRegister ← D_0
2:  C ← C.insert(firstRegister.id, [])
3:  centroids ← getItemsByIDs(C.keys())
4:  for k ← 1 to length(D) do
5:      foundCentroid ← false
6:      for centroid in centroids do
7:          dist ← distanceFn(D_k, centroid)
8:          if dist < threshold then
9:              C.get(centroid.id).push(D_k)
10:             foundCentroid ← true
11:             break
12:         end if
13:     end for
14:     if !foundCentroid then
15:         C ← C.insert(D_k.id, [D_k])
16:         centroids.push(D_k)
17:     end if
18: end for
```

Quanto aos clusters, são armazenados em um dicionário da linguagem Python, esse tipo de estrutura comumente conhecido como *hash table*, é uma estrutura de armazenamento chave-valor. A chave do cluster é o ID único do centróide de cada cluster, já o valor, é a lista de registros daquele cluster, considerados, portanto, duplicatas entre si.

3.2.4 Nova Blocagem

Conforme explicado na seção 2.1.1, a blocagem é uma etapa fundamental no processo de RE, pois ao agrupar registros semelhantes, reduz a quantidade de comparações necessárias nas etapas posteriores de comparação e classificação, dessa forma, melhorando a velocidade de execução dos algoritmos de RE. Entretanto, é necessário equilibrar essa troca de eficiência e velocidade de desempenho.

A abordagem de blocagem do Algoritmo Base, que após o processo de Refatoração se tornou ProxCluster Estático, modularizou a etapa de blocagem na classe PhonexStaticBlocking, cuja estratégia pode apresentar limitações quando se trata de agrupar corretamente registros que deveriam pertencer ao mesmo bloco, tendo em vista que registros muito semelhantes ou até mesmo com títulos iguais poderiam ser separados em blocos diferentes caso o limite de registros daquele bloco fosse atingido.

Para superar essa limitação, desenvolveu-se um novo método de blocagem para o continuamento do presente trabalho, a classe SoundexBlocking, que utiliza um método semelhante ao PhonexStaticBlocking, mas possui blocos de tamanhos dinâmicos. Nesse método, os BKV são gerados utilizando outra biblioteca de codificação fonética, o Fuzzy[10], com o algoritmo Soundex.

O algoritmo Soundex gera os mesmos códigos de fonética para palavras semelhantes THE SOUNDEX INDEXING SYSTEM (2007), portanto, não é mais necessário uma etapa de ordenação baseado nesse código, como foi feito no PhonexStaticBlocking. No SoundexBlocking, os blocos são gerados agrupando registros que possuem o mesmo BKV independente de quantos registros cada bloco irá possuir.

É importante ressaltar também que após o processo de generalização da etapa de Refatoração, o atributo utilizado para chave de blocagem é um parâmetro passado para as classes PhonexStaticBlocking e SoundexBlocking, dessa forma, possibilitando o uso das mesmas para diferentes bases e seus respectivos atributos.

Em resumo, o desenvolvimento desse novo método de blocagem com tamanhos de blocos dinâmicos garante o agrupamento correto de registros com atributos de chave de blocagem semelhantes, assegurando que eles permaneçam no mesmo bloco. Essa abordagem melhorou a eficiência da RE do trabalho, evitando a separação indesejada de registros que deveriam pertencer ao mesmo bloco.

3.2.5 ProxCluster Incremental

Dados incrementais fazem referência as ações de inserção de novos registros, atualização e remoção. Para o contexto do presente trabalho, os algoritmos desenvolvidos lidam apenas com a inserção, portanto, os registros clusterizados previamente permanecem inalterados quanto às atualizações e remoções.

A adição da estratégia incremental foi realizada modificando as duas classes, ProxCluster e Deduplicator. O processo de desenvolvimento foi implementado primeiramente na classe ProxCluster, que recebeu um parâmetro opcional, os clusters. Quando fornecido, os centróides, que anteriormente eram inicializados utilizando o primeiro elemento dos registros a serem resolvidos, passaram a ser iniciados com os clusters passados como parâmetro. Logo, o processo de REI que será realizado com os dados fornecidos, irá prosseguir da mesma maneira que o ProxCluster Estático, entretanto, como foram enviados clusters, caracterizando o processo como incremental, o algoritmo já terá seus centróides iniciais definidos.

Conforme pode-se observar no Algoritmo 3 do ProxCluster Incremental, foi acrescentado os condicionais nas linhas 1 à 5. Quando os *clusters* são fornecidos ao algoritmo, eles são atribuídos à variável C, que são os clusters resultantes do algoritmo e que serão preenchidos no processo iterativo da linha 8. Caso o parâmetro *clusters* não seja passado, a inicialização é realizada da mesma maneira que antes.

[10]Fuzzy: `https://github.com/yougov/Fuzzy`

Algorithm 3 ProxCluster Incremental

Input: $D, distanceFn, threshold, clusters$
Output: C ▷ clusters

1: **if** $clusters \neq NULL$ **then** ▷ Inicialização incremental
2: $C \leftarrow clusters$
3: **else** ▷ Inicialização estática
4: $firstRegister \leftarrow D_0$
5: $C \leftarrow C.insert(firstRegister.id, [])$
6: **end if**
7: $centroids \leftarrow getItemsByIDs(C.keys())$
8: **for** $k \leftarrow 1$ to $length(D)$ **do**
9: $foundCentroid \leftarrow false$
10: **for** $centroid$ *in* $centroids$ **do**
11: $dist \leftarrow distanceFn(D_k, centroid)$
12: **if** $dist < threshold$ **then**
13: $C.get(centroid.id).push(D_k)$
14: $foundCentroid \leftarrow true$
15: **break**
16: **end if**
17: **end for**
18: **if** $!foundCentroid$ **then**
19: $C \leftarrow C.insert(D_k.id, [D_k])$
20: $centroids.push(D_k)$
21: **end if**
22: **end for**

Essa primeira etapa de adaptação do algoritmo foi relativamente simples, porém, um desafio surgiu quanto à definição dos clusters que seriam enviados ao realizar a REI dos dados.

Note que no processo estático, ou seja, quando clusters não são enviados, o algoritmo inicia com apenas um centróide, portanto, a quantidade de comparações para os registros da iteração, é inicialmente uma (um centróide para comparar) e vai aumentando conforme os registros não se classificam como duplicatas dos centróides. Idealmente, os registros enviados para o ProxCluster, são resultantes do processo de blocagem, e, portanto, potenciais duplicatas entre si. Logo, a quantidade de centróides, e consequentemente, de comparações realizadas serão baixas para cada bloco.

Entretanto, no caso incremental, enviar como parâmetro todos os clusters (de todos os blocos) que foram resultantes do processo anterior (clusters resolucionados), significa uma quantidade potencialmente alta de centróides, ou seja, de comparações à serem realizadas para os novos registros incrementais desde a primeira iteração. Comparado ao processo estático, que

começa com uma comparação apenas, o envio de todos os clusters resolucionados anteriormente, tornará a execução desnecessariamente mais lenta.

O ideal, é que apenas clusters que compartilhem o mesmo BKV dos registros incrementais é que sejam enviados. Afinal, supondo que estes dados incrementais estivessem a disposição desde o começo, no processo estático, teriam sido blocados juntos, e portanto, clusterizados conjuntamente. Selecionar apenas os clusters com o mesmo BKV do bloco incremental é uma tarefa essencial, e tornará o processo do ProxCluster incremental de fato eficiente. A classe Deduplicator ficou responsável por identificar qual bloco de clusters devem ser utilizados quando novos registros incrementais são processados.

O Deduplicator assumiu uma função crucial no processo de REI do projeto. Além de lidar com o processo de blocagem e armazenamento dos clusters indexados por cada bloco, ele também armazena em uma estrutura *hash table*, o mapeamento entre cada BKV com o índice desse bloco clusterizado. Essa estratégia revelou-se de extrema importância para o processo incremental.

O funcionamento do Deduplicator pode ser representado pelo diagrama de sequência da Figura 3.3. Nele temos a sequência de etapas que são tomadas, iniciando com a blocagem dos registros com o módulo SoundexBlocking. Com os blocos retornados, é possível ver na Figura 3.3, o laço de repetição para os blocos obtidos, onde é realizado a clusterização por meio do módulo ProxCluster para cada bloco da iteração. Note (bloco *alt* da Figura 3.3) que também é de responsabilidade do Deduplicator selecionar os clusters resolucionados anteriormente para o caso da REI de blocos com o mesmo BKV. Por fim, o desempenho dos clusters gerados podem ser avaliados utilizando a classe Evaluator que será melhor detalhada na seção 3.2.6.

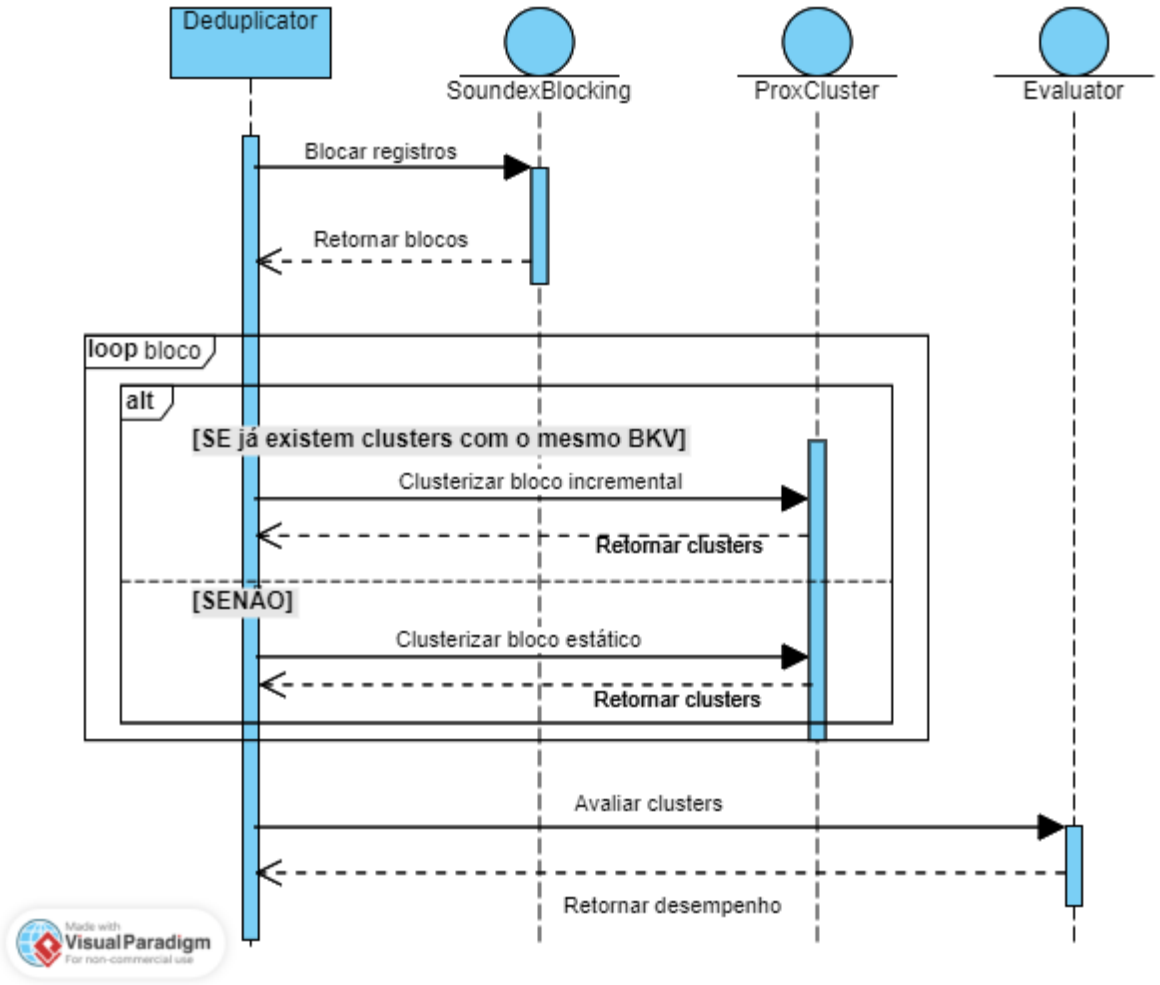

Figura 3.3: Diagrama de Sequência do Deduplicator

Fonte: Autoria própria

Conforme o pseudo-código do Algoritmo 4 da classe Deduplicator, é possível observar que as entradas para a inicialização da classe são as configurações necessárias para o processo de resolução. São elas: o atributo utilizado como chave de blocagem (*blockingAttr*), a função de distância que será passada para o ProxCluster (*distance*), o ID global para referenciar os registros (*uID*) e o limite de distância para classificação de duplicatas (*threshold*). O método *RUN* é responsável por executar as etapas de RE e lidar com a lógica incremental, retornando todos os clusters, incluindo os de execuções anteriores.

Nas linhas 1 à 4, é realizado a inicialização dos componentes que serão utilizados, tais como, as instâncias do SoundexBlocking e do ProxCluster, como também o mapeamento que será feito dos BKV e seus clusters, na *hash table bkvClustersMap*. O método *RUN*, da linha 5 é responsável por receber os registros e realizar o processo de REI. Começando pela blocagem realizada na linha 6, os blocos são percorridos iterativamente (linha 7) e nas linhas 8 e 9 são recuperados os clusters de mesmo BKV do bloco atual, caso existam, estes clusters também serão enviados para o ProxCluster realizar a clusterização do bloco de maneira incremental, conforme as linhas 10 à 14. As últimas linhas da estrutura de repetição, 15 e 16, são responsáveis

por atualizar os clusters, tanto do mapeamento *bkvClustersMap*, como dos clusters totais, *C*. Por fim, na linha 18, é retornado todos os clusters resolucionados.

Algorithm 4 Deduplicator

Input: $blockingAttr, distance, uID, threshold$

```
1:  blocker ← SoundexBlocking(blockingAttr)
2:  proxCluster ← ProxCluster(distance,threshold)
3:  bkvClustersMap ← {}
4:  C ← {}                                          ▷ Inicialização dos clusters vazios
5:  function RUN(D)
6:      newBlocks ← blocker.generateBlocks(D)
7:      for block in newBlocks do
8:          bkv ← getBlockBlockingKeyValeu(block)
9:          clusterizedBlock ← bkvClustersMap.get(bkv)
10:         if clusterizedBlock ≠ NULL then          ▷ Incremental
11:             blockClusters ← proxCluster.cluster(block,clusterizedBlock)
12:         else                                     ▷ Novo bloco
13:             blockClusters ← proxCluster.cluster(block)
14:         end if
15:         bkvClustersMap[bkv] ← blockClusters
16:         C.update(blockClusters)
17:     end for
18:     return C
19: end function
```

Em razão da lógica adotada pelo Deduplicator, é realizado a seleção inteligente dos clusters existentes permitindo que o algoritmo lide com registros incrementais de forma mais eficiente. Em vez de recalculá-los completamente a cada nova entrada, o ProxCluster pode aproveitar clusters já existentes, economizando tempo computacional e recursos.

3.2.6 Avaliação

Nesta seção, será apresentado os métodos de avaliação utilizados para medir o desempenho dos experimentos realizados neste trabalho, destacando as métricas de avaliação de acurácia, precisão, *recall* e medida F.

O processo de RE pode ser visto como um processo de classificação, se um par de registros for classificado como duplicatas, presume-se que ambos os registros no par se referem à mesma entidade no mundo real. Por outro lado, se um par de registros for classificado como não duplicatas, presume-se que os dois registros no par se referem a duas entidades diferentes no mundo real CHRISTEN (2012). No caso do ProxCluster, todos os elementos presentes no

cluster formam pares de registros considerados duplicatas entre si, já os pares de registros entre clusters diferentes são considerados não duplicatas entre si.

Para auxiliar na avaliação do desempenho do ProxCluster, foi utilizado a matriz de confusão, uma tabela que apresenta o número de classificações corretas e incorretas feitas pelo algoritmo em um conjunto de dados. Para preencher a matriz é desejável ter um conjunto de dados com pares de registros previamente rotulados como duplicatas (verdadeiros positivos). Esse conjunto de dados é chamado de *ground-truth data set* (conjunto de dados verdade), pois contém o gabarito real de duplicatas entre os registros CHRISTEN (2012). A matriz de confusão é composta por quatro elementos principais:

- Verdadeiros Positivos (VP): Número de registros corretamente identificadas como pertencentes à classe positiva (registros corretamente resolvidos).
- Falsos Positivos (FP): Número de registros erroneamente classificadas como pertencentes à classe positiva quando, na verdade, são negativas (registros incorretamente resolvidos).
- Verdadeiros Negativos (VN): Número de registros corretamente identificadas como pertencentes à classe negativa (registros corretamente não resolvidos).
- Falsos Negativos (FN): Número de registros erroneamente classificadas como pertencentes à classe negativa quando, na verdade, são positivas (registros não resolvidos erroneamente)

Dessa forma, a matriz de confusão permite visualizar o desempenho de algoritmos de classificação em relação às classes positivas e negativas, facilitando a compreensão de seus acertos e erros. Com a matriz de confusão é possível calcular medidas que podem ser utilizadas para qualificar o processo de RE. A lista à seguir apresenta algumas destas principais medidas, segundo CHRISTEN (2012).

- A **acurácia** é uma medida geral de desempenho que avalia a proporção de classificações corretas em relação ao total de classificações. A fórmula para calcular a acurácia é:

$$\frac{VP+VN}{VP+VN+FP+FN}$$

  É principalmente útil em situações em que as classes estão balanceadas, ou seja, quando o número de pares de registros duplicatas e não duplicatas é mais ou menos o mesmo. Entrentanto, na maioria dos casos do contexto de RE, a maior parte dos pares de registros corresponde a não duplicatas verdadeiras (VN). Como resultado, a medida de acurácia não é adequada para avaliar corretamente a qualidade da RE, tendo em vista que o valor de VN domina o cálculo da acurácia.

Para exemplificar, imagine o cenário de RE de duas bases de dados, cada uma com 100.000 registros. Ao cruzá-las (ignorando a etapa de blocagem para simplificar), obtemos 5.000.000 de pares de registros possíveis. Dentre esses, 50.000 pares são duplicatas verdadeiras. Supondo que o ProxCluster identificou 60.000 pares como duplicatas, dos quais 40.000 são realmente verdadeiros. Isso resulta em: VP = 40.000, FN = 10.000, FP = 20.000 e VN = 4.930.000. Calculando a acurácia com base nesses valores: acurácia = 0,994, ou seja, 99,4%. No entanto, essa medida não é representativa, pois somente 40.000 das 50.000 duplicatas verdadeiras foram corretamente identificadas. Mesmo se classificássemos todos os pares de registros como não duplicatas (VP = 0, FN = 50.000 e FP = 0), a acurácia ainda seria muito alta.

Por este motivo, a medida de acurácia não será utilizada como métrica de avaliação dos experimentos.

- A **precisão** mede a proporção de VP em relação ao total de resultados positivos classificados pelo algoritmo. Ela é especialmente relevante quando se deseja minimizar os FP. A fórmula para calcular a precisão é:

$$\frac{VP}{VP+FP}$$

- O ***recall***, também conhecido como sensibilidade, avalia a capacidade do algoritmo em identificar todas as instâncias positivas corretamente. É importante em cenários onde os FN devem ser minimizados. A fórmula para calcular o recall é:

$$\frac{VP}{VP+FN}$$

- A **medida F** é uma métrica que combina tanto a precisão quanto o *recall*, proporcionando uma visão mais completa do desempenho do algoritmo. É calculada através da média harmônica entre precisão e *recall*:

$$2 \times \frac{precisao \times recall}{precisao + recall}$$

Para alcançar um alto valor de medida F é necessário encontrar o melhor equilíbrio entre precisão e *recall*.

A classe Evaluator ficou responsável por calcular as métricas de precisão, *recall* e medida F dos clusters gerados pelo ProxCluster. Para tal, ela recebe dois tipos de entrada essenciais: os clusters resolucionados e um vetor de pares como conjunto de dados verdade que serve como o gabarito.

O processo de avaliação começa com a utilização dos clusters enviados para gerar os pares de duplicatas. Isso é feito gerando pares de registros que estão contidos no mesmo cluster. Esses pares de duplicatas gerados serão então usados para criar a matriz de confusão utilizando os pares do conjunto verdade como gabarito. Por fim, com base nas contagens presentes na matriz de confusão, o Evaluator calcula as três métricas citadas anteriormente.

3.3 Exemplo Motivacional

Nesta seção, será exemplificado um cenário para melhor entendimento dos processos realizados pelo ProxCluster, além de ilustrar uma problemática adequada à solução proposta neste trabalho.

Supondo um cenário onde a Empresa Atacado Barato possui diversas filiais, cada uma com um banco de dados onde são armazenadas as vendas, clientes, funcionários, entre outras informações relevantes. É desejo da empresa integrar esses bancos de dados, oriundos de suas diferentes filiais, para que seja possível analisar essas informações e realizar processos de estratégia de negócio considerando todas as filiais. Entretanto, é de conhecimento da Empresa Atacado Barato, a existência de duplicatas entre esses bancos de dados, visto que não há integração entre eles, estas ocorrências podem ter sido ocasionadas devido a funcionários terem sido realocados entre as filiais e clientes que podem ter consumido em mais de uma filial, além de outros cenários mais atípicos, como erro humano na utilização dos sistemas.

Para simplificar o exemplo, será analisado o processo de integração apenas sobre os registros das bases de clientes das filiais da Empresa Atacado Barato.

Para integrar os bancos de dados, previamente a etapa de RE que irá identificar as duplicatas, é realizado o processo de Resolução de Esquema (*schema matching*), responsável pelo mapeamento dos atributos das diferentes bases. Com essa etapa, supondo que a filial X possua dois atributos, “primeiro nome” e “sobrenome”, para os nomes dos clientes, mas as demais filiais da empresa, possuam apenas um campo “nome”, é necessário realizar essa mapeamento entre os atributos. Por exemplo, mapear todos os atributos relacionados ao nome do cliente para um atributo só, “nome”.

Após a etapa de Resolução de Esquema, é iniciado o processo de RE. Inicialmente, é feito o pré-processamento das bases, pois é necessário remover tudo aquilo que for indesejado, tais como caracteres especiais e em branco, padronizar capitalizações e outros formatos dentre os diferentes atributos. Dessa maneira, os registros de clientes, já padronizados e limpos, são todos enviados para o Deduplicator, módulo mais alto nível do *framework* ProxCluster, que irá identificar as duplicatas e retorná-las agrupadas em clusters.

Em primeiro lugar, na execução do Deduplicator, os registros serão separados em blocos utilizando o módulo SoundexBlocking (detalhado na subseção 3.2.4). Conforme ilustrado pela Figura 3.4, supondo que o atributo escolhido como chave de blocagem tenha sido o “nome” , os registros com nomes foneticamente semelhantes geram a mesma BKV e, portanto, irão ficar

no mesmo bloco, como é o caso dos registros de “id” 034 e 024 da Figura 3.4, que possuêm a mesma BKV, t200.

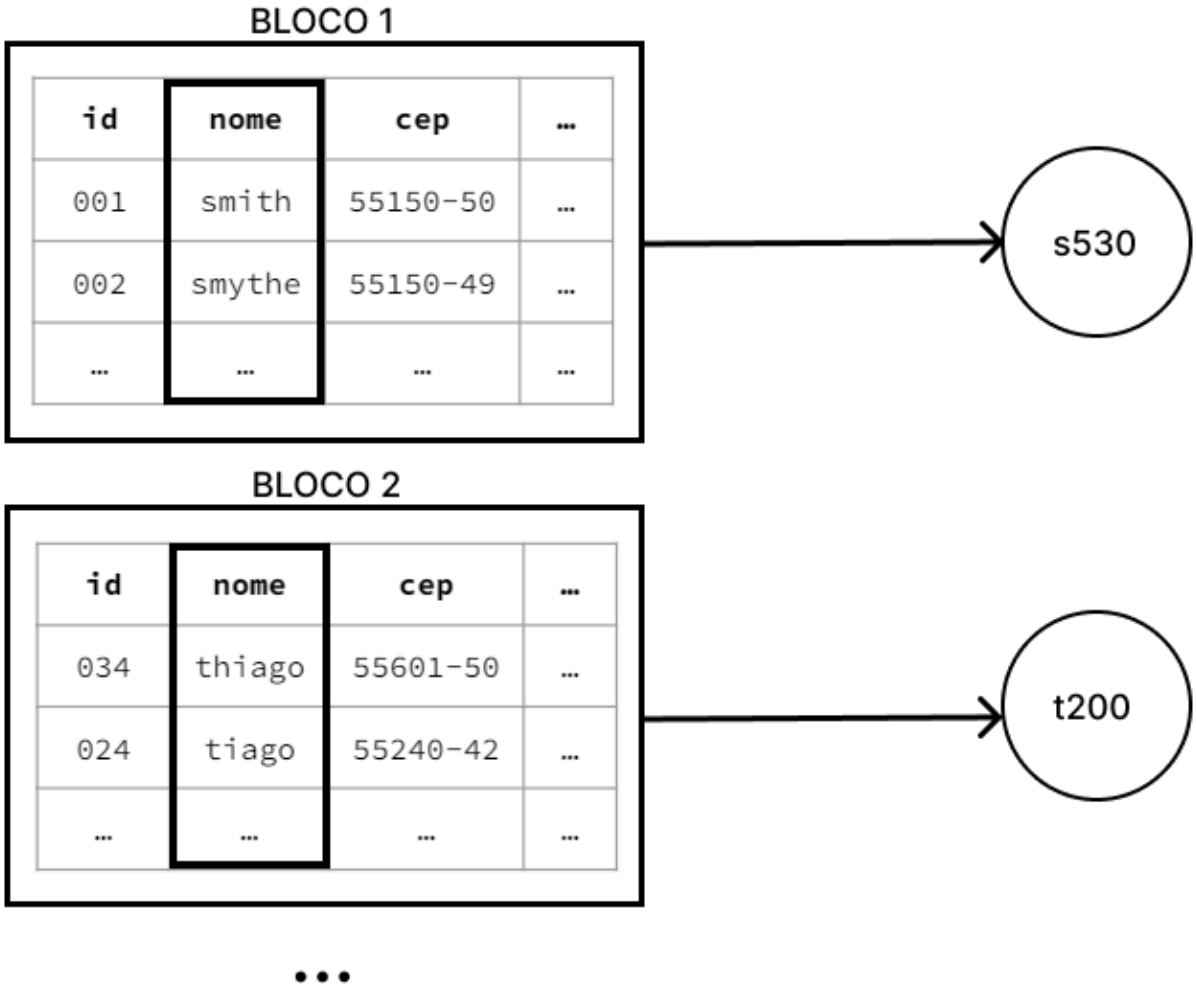

Figura 3.4: Exemplo do processo de blocagem - SoundexBlocking
Fonte: Autoria própria

Após a blocagem dos registros, o Deduplicator percorre cada bloco gerado e verifica se existem clusters resolvidos anteriormente com a mesma BKV do bloco da iteração atual, caso sim, estes clusters também serão enviados para a etapa de correspondência, realizada pelo módulo de mesmo nome do *framework*, ProxCluster, e que foi detalhado na subseção 3.2.5. Entretanto, não será o caso para nenhum dos blocos gerados, já que neste exemplo, esta será a primeira execução realizada, logo, cada bloco será enviado para o módulo ProxCluster para serem clusterizados de maneira estática, pois não há nenhum cluster para ser aproveitado em uma possível estratégia incremental.

No algoritmo ProxCluster, responsável pela etapa de correspondência, conforme explicado anteriormente, como não há resultados de processos anteriores para serem aproveitados, a execução será de maneira estática. A lista de centróides será inicializada arbitrariamente com o primeiro registro do bloco enviado. A partir daí, os registros são iterados e para cada registro

é calculado a distância do mesmo para os centróides. Caso uma dessas distâncias seja menor que o *threshold* definido, então o registro da iteração atual é colocado no cluster do centróide identificado como duplicata. No caso de nenhum centróide ter se encaixado como duplicata, o próprio registro é inserido na lista de centróides e a iteração continua.

Quanto a função de distância, a mesma é definida pelo usuário, e é essencial que seja feito de maneira atenciosa, pois é responsabilidade dela comparar o par de registros. Sua saída é entre 0 e 1, sendo 0 a distância mínima, indicando que os registros tem similiridade máxima, e a distância 1, o contrário. A definição da função de distância, por exemplo, pode ser realizada utilizando o algoritmo Levenshtein para calcular a distância entre os atributos textuais dos registros, assim como utilizado na subseção 3.2.1. Para atributos numéricos, pode-se utilizar a diferença normalizada entre 0 e 1 dos valores. Por fim, a partir das distâncias de cada atributo do par de registros comparados, é calculada a média aritmética, possibilitando inclusive, adicionar pesos maiores para atributos considerados como mais determinantes na classificação das duplicatas. No contexto do exemplo dessa seção, atributos como o “nome” e “CPF” são mais indicativos de que os registros são duplicatas (quando próximos) do que o atributo “idade”, portanto, “nome” e “CPF” terão pesos maiores.

Após todos os registros terem sido clusterizados pelo algoritmo ProxCluster, os clusters de cada bloco são salvos no contexto do Deduplicator, que também relaciona-os com a BKV de cada bloco, possibilitando assim, reutilizar os clusters numa futura execução incremental, enviando apenas clusters com a mesma BKV do bloco incremental.

Supondo que novos registros tenham chegado nas filiais, enquanto esses bancos de dados ainda não são integrados entre si, se faz necessário realizar a inserção desses incrementos nos resultados encontrados do processo anterior. Graças a estratégia incremental do ProxCluster, esses registros incrementais podem ser enviados para o o Deduplicator, que irá realizar as mesmas etapas já citadas anteriormente sem precisar executar tudo do zero com os registros novos. Iniciando pela blocagem dos registros incrementais, cada bloco incremental irá ser enviado para a etapa de correspondência do algoritmo ProxCluster, é de responsabilidade do Deduplicator selecionar clusters de mesma BKV do bloco incremental, caso existam, para que esses resultados possam ser reaproveitados. Sendo esse o caso, no algoritmo ProxCluster, os centróides e clusters já são inicializados de acordo com os clusters recebidos e o processo percorre da mesma maneira, conforme detalhado no algoritmo 3.

O foco desse trabalho recai apenas na identificação e clusterização das duplicatas. Entretanto, no contexto da problemática escolhida como exemplo, as duplicatas encontradas, isto é, registros que ficaram em um mesmo cluster, ainda precisam ser tratadas. As duplicatas podem ser enviadas para um processo chamado de Fusão de Dados. Segundo BLEIHOLDER; NAUMANN (2009), este processo tem a responsabilidade de aplicar várias estratégias de resolução de conflitos para fundir múltiplos registros que representam a mesma entidade em uma representação única, consistente e limpa.

Após o processo de integração de dados concluído, com a identificação e tratamento das

duplicatas entre as bases de dados, a Empresa Atacado Barato pode utilizar os registros de todas suas filiais de maneira mais confiável.

4

Análise dos resultados

Este capítulo apresenta os resultados obtidos por meio da condução de uma série de experimentos com o propósito de avaliar o desempenho e a eficácia das estratégias e algoritmos desenvolvidos no presente estudo. Foram realizados quatro tipos de experimentos distintos, visando avaliar aspectos diferentes das soluções propostas no presente trabalho. Note que o desempenho das métricas dos experimentos realizados é resultante de um conjunto de fatores, como a base utilizada, função de distância, estratégia de blocagem, e o *threshold* (limite de distância superior) escolhidos. Todos esses fatores têm influência nos resultados, e portanto, buscou-se manter um padrão nessas escolhas entre os experimentos realizados.

Quanto as bases de dados escolhidas, note que a quantidade de duplicatas é referente a quantidade de pares de duplicatas, portanto, três registros A, B e C, duplicatas entre si, formam três pares de duplicatas, AB, AC e BC. As duplicatas de cada base foram identificadas previamente em processo manual pelos seus disponibilizadores.

A base de dados CD Information foi utilizada nos quatro tipos de experimentos. A mesma foi proposta por DRAISBACH; NAUMANN (2010), e é composta por um total de 9763 registros aleatoriamente selecionados a partir do banco de dados "freeDB", abrangendo um conjunto de 299 pares de duplicatas. Cada um destes CDs possui informações que englobam atributos fundamentais, como título, artista, gênero, ano, álbum, a listagem das faixas contidas no CD, entre outros dados relevantes. A seleção das características específicas para a função de distância focalizou nos campos de título, artista e a primeira faixa ("track01") de cada CD. Esses atributos foram escolhidos com base na sua capacidade de representar de maneira concisa e significativa as características distintivas de cada CD. A função de distância adotada para os experimentos consistiu na aplicação da média dos valores da distância Levenshtein normalizada sobre esses campos. O *threshold* escolhido foi 0.25, ou seja, os registros precisam ter uma distância menor que 0.25 para serem clusterizados como duplicatas, e o atributo utilizado como chave de blocagem foi o título dos registros.

Já apresentada na subseção Algoritmo Base 3.2.1, a base de dados MusicBrainz de 19375 registros foi utilizada nos experimentos de comparação entre as versões estática e incremental do ProxCluster. A função de distância, *threshold* e atributo escolhido como chave de blocagem

foram os mesmos que utilizados no trabalho prévio desenvolvido, cuja discussão foi realizada na subseção 3.2.1.

Outra base utilizada nos experimentos de comparação entre ProxCluster Estático e Incremental foi a Geographic Settlements LEIPZIG (2019), a mesma contém informações geográficas de entidades do mundo real, provenientes de quatro diferentes fontes de dados. Esta base compreende um total de 3054 registros, dos quais foram identificados 4391 pares de duplicatas. Os atributos presentes são *label* (rótulo), latitude e longitude. É importante destacar que os campos de latitude e longitude podem conter valores ausentes. A função de distância empregada é calculada tomando como base a média da distância entre os *labels* dos registros, utilizando a distância de Levenshtein normalizada. Além disso, quando as informações de latitude e longitude estavam presentes, a distância entre elas era calculada da seguinte forma: se a diferença entre as coordenadas de latitude/longitude dos itens fosse menor que 100, a distância seria considerada como 0. Caso contrário, a distância seria atribuída o valor de 1. Devido ao fato de que as coordenadas podem estar frequentemente ausentes nos registros, a *label* teve o dobro do peso em relação às coordenadas de latitude e longitude. A *label* também foi utilizada como chave de blocagem e o *threshold* escolhido foi 0.2.

4.1 Pandas vs Polars

O primeiro experimento teve como objetivo analisar o tempo de processamento da implementação do algoritmo utilizando duas diferentes bibliotecas: Pandas e Polars, conforme foi apresentado na subseção de Refatoração 3.2.2. Foram coletados os tempos de execução para cada implementação ao realizar o processo de RE na base CD Information.

Os resultados destes experimentos forneceram informações acerca do desempenho comparativo das bibliotecas em termos de eficiência de processamento e conforme o a Tabela 4.1, é possível observar o melhor desempenho apresentado pela implementação com a biblioteca Polars, atigindo um tempo de processamento aproximadamente três vezes mais rápido que a concorrente.

O melhor desempenho do Polars pode ter se dado pelos seguintes aspectos: O Pandas utiliza um DataFrame baseado em matriz numpy 2D, enquanto o Polars adota um DataFrame com base no formato de dados Arrow da linguagem Rust. Isso faz com que o Polars seja mais rápido, pois o formato Arrow é mais eficiente para manipulação de dados. Além disso, o Polars possui algoritmos otimizados de menor complexidade temporal, resultando em melhor desempenho para conjuntos de dados grandes. Em contrapartida, o Pandas usa algoritmos com maior complexidade temporal, o que o torna mais lento ao lidar com grandes conjuntos de dados ALMOGKLEIN (2023).

Implementação	Tempo (s)
Pandas	29.7
Polars	9.4

Tabela 4.1: Tempo de processamento: ProxCluster implementado com Pandas vs Polars

4.2 PhonexStaticBlocking vs SoundexBlocking

O segundo experimento concentrou-se na comparação de desempenho entre duas estratégias distintas de blocagem desenvolvidas no presente trabalho, PhonexStaticBlocking e SoundexBlocking. Foram conduzidas análises das métricas de eficácia, tempo de execução e quantidades de blocos gerados da RE na base de CD Information utilizando ambas estratégias de blocagens disponíveis. Lembrando que o PhonexStaticBlocking agrupa os registros em blocos de tamanho fixo, portanto, o experimento foi realizado variando esse tamanho em 50, 25, 15 e 5.

De acordo com os resultados da Tabela 4.2, é possível observar no PhonexStaticBlocking, que quanto menos blocos gerados, e consequentemente, mais registros por bloco, o tempo de execução é maior, já que possivelmente mais comparações serão realizadas em cada bloco.

A partir da Tabela 4.2, foi gerado os gráficos da Figura 4.1. Quanto ao tempo de execução com o algoritmo SoundexBlocking, apesar de ter gerado a maior quantidade de blocos, teve o segundo pior tempo. Acredita-se que o motivo disso seja porque apesar de ter gerado muitos blocos, devido a sua natureza dinâmica e a variedade de títulos (atributo utilizado como chave de blocagem) dos registros da base de CD Information, diversos blocos foram gerados com poucos, ou até mesmo, apenas um elemento, em contrapartida, alguns blocos ficaram com muitos elementos, assim, aumentando o tempo de execução.

Algoritmo	Tempo (s)	Precisão (%)	Recall (%)	Medida-F (%)	Qntd Blocos
PhonexStaticBlocking-50	12	96.3	78.3	86.3	196
PhonexStaticBlocking-25	8.6	96.2	75.6	84.6	391
PhonexStaticBlocking-15	7.7	96.9	74.2	84.1	651
PhonexStaticBlocking-5	7.7	96.1	58.2	72.5	1953
SoundexBlocking	9.2	96.4	80.3	87.6	2628

Tabela 4.2: Algoritmos de blocagem PhonexStaticBlocking e SoundexBlocking

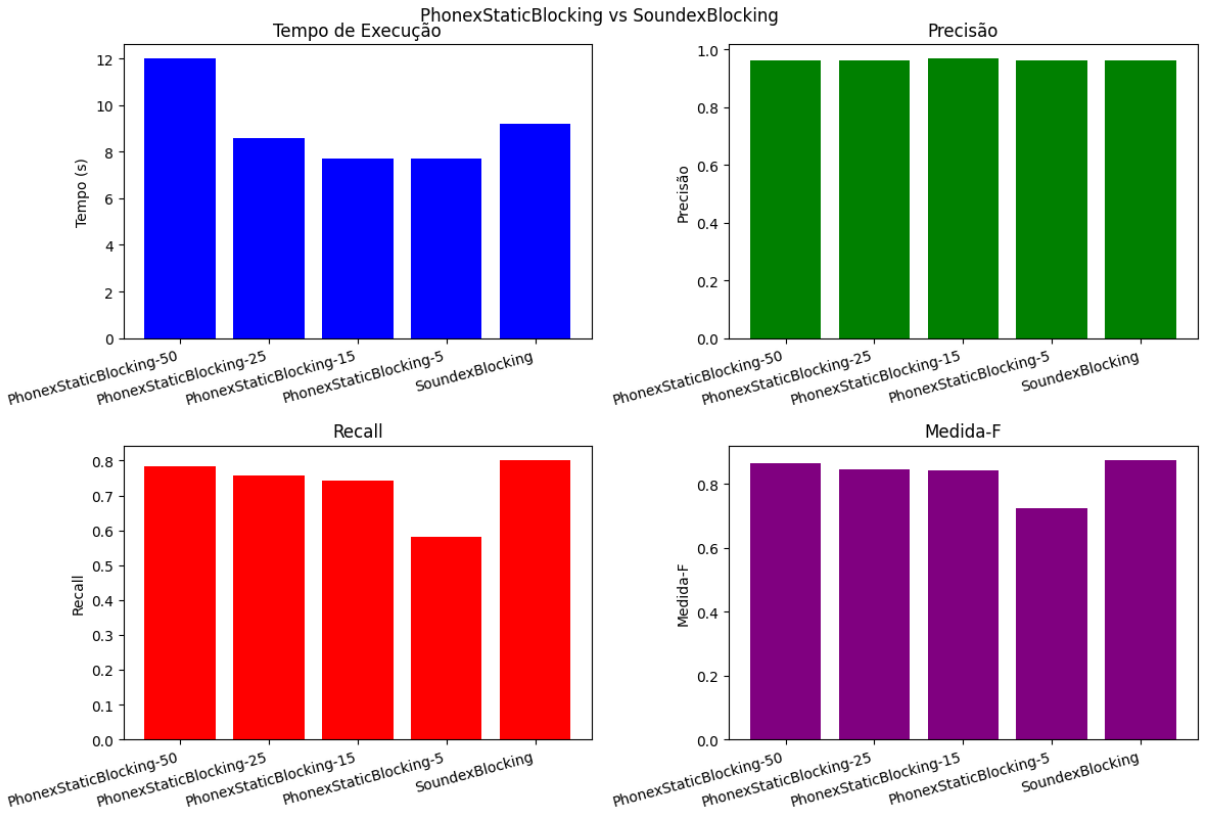

Figura 4.1: Desempenho - PhonexStaticBlocking vs SoundexBlocking
Fonte: Autoria própria

Quanto as métricas de qualidade, é possível observar na Figura 4.1, pouca variação na precisão, com o PhonexStaticBlocking de tamanho 15 obtendo o melhor resultado, seguido do SoundexBlocking. Já o *recall*, têm uma relação mais íntima com a etapa de blocagem, pois, mede a capacidade de identificar corretamente todas as duplicatas em relação ao total de duplicatas reais. Portanto, é possível observar uma maior variação nessa métrica.

O PhonexStaticBlocking de tamanho 5, por sua vez, possui o pior valor de *recall*, isso deve ter se dado por ter poucos elementos por bloco e devido a natureza de repartição fixa dos blocos, oriunda da estratégia do PhonexStaticBlocking, ocasionou na separação de diversos elementos duplicados. O SoundexBlocking ficou com o mais alto valor de *recall*, mesmo possuindo muito mais blocos que o segundo colocado nesse sentido, o PhonexStaticBlocking de tamanho 50. O SoundexBlocking, também ficou com a maior pontuação de Medida-F, já que é a relação entre as duas métricas anteriores, cujo o mesmo possui bons resultados.

Isso comprova a eficiência da estratégia dinâmica adotada pelo SoundexBlocking, pois é possível observar que enquanto no PhonexStaticBlocking, quanto menos blocos, ou seja, mais registros por bloco, maior é o valor alcançado no *recall*, o SoundexBlocking, por sua vez, conseguiu o maior valor nessa métrica sem fazer necessário a escolha de um tamanho como parâmetro.

4.3 ProxCluster Estático vs Incremental

O terceiro conjunto de experimentos teve como alvo a comparação entre as versões estática e incremental do ProxCluster. Além disso, foram empregadas três bases de dados distintas para avaliar o desempenho e a escalabilidade das abordagens em contextos diversos. Para cada base, foi executado o ProxCluster de maneira estática, ou seja, sobre 100% da base, e para o incremental, foram realizadas partições de 50%, 35% e 15%, cada partição executada nessa ordem, resultando, de maneira incremental após três execuções, na base toda resolucionada. Além disso, afim de analisar a relação da blocagem com os resultados obtidos, foi executado o ProxCluster sem nenhuma etapa de blocagem, no que foi rotulado de Estático Ingênuo. As métricas de interesse abrangeram o tempo de processamento e as métricas de qualidade discutidas anteriormente. Quanto aos parâmetros escolhidos para cada base, foram mantidos fixos entre os experimentos, conforme explicado no início do capítulo.

4.3.1 CD Information

No que se refere ao tempo de processamento, a Tabela 4.3 apresenta os resultados esperados, o tempo de processamento do ProxCluster estático foi de 8.7 segundos, já os tempos para cada uma das execuções incrementais obtiveram valores menores comparados ao processamento estático da base. Para incrementos menores, como a partição dos 15% restantes da base, torna ainda mais notório a importância de uma estratégia incremental, pois o processo de REI permitiu a resolução dos novos dados em um tempo de execução aproximadamente 70% mais rápido do que seria necessário para resolucionar os 100% da base do completo zero (estático). Note que o experimento incremental de 15% representa bem a importância dessa estratégia, pois é um exemplo mais usual da realidade dos problemas de REI, onde geralmente já terá sido resolucionado uma grande quatidade de registros, no caso do experimento, 85%, e é necessário resolucionar um incremento menor de novos dados. Sem a estratégia incremental, todo os registros, ou seja, 100% da base no caso do experimento, teriam que ser resolucionados, processo que demorou 8.7 segundos. Com a REI dos 15%, a resolução teve duração de apenas 2.6 segundos.

Quanto ao processo Estático Ingênuo, a diferença abismal de tempo de processamento demonstra a importância da etapa de blocagem para o processo de RE se tratando de tempo de execução e recursos computacionais. O mesmo não foi incluído no gráfico de tempo da Figura 4.2, para não afetar negativamente a visualização, pois seu valor é muito acima dos demais.

ProxCluster	Incremento (%)	Tempo (s)
Estático	100	8.7
Incremental	50	4.9
Incremental	35	4.2
Incremental	15	2.6
Estático Ingênuo	100	1140

Tabela 4.3: Tempo de processamento - ProxCluster Estático vs Incremental - CD Information

No processo de REI é importante o balanceamento entre o ganho de tempo de processamento e a perca de desempenho que pode ocorrer nos processos incrementais quando comparados ao estático. A Figura 4.2, gerada com os dados das Tabelas 4.3 e 4.4, mostra que para esse experimento, não houve perca de desempenho significativo nas métricas observadas. Mantendo os mesmos valores para a estratégia estática e incremental.

É possível observar, entretanto, que ao remover a etapa de blocagem (Estático Ingênuo), o *recall* aumentou cerca de 4.3%, isso ocorreu devido a redução de FN que comumente ocorrem devido a separação errônea de registros duplicados na etapa de blocagem. Por outro lado, a precisão diminuiu 2%, devido a um aumento na quantidade de FP, pois sem a etapa de blocagem, muito mais comparações foram realizadas, consequentemente, classificações erradas ocorreram, isso pode ser ajustado alterando a função de distância e o *threshold*.

ProxCluster	Precisão (%)	Recall (%)	Medida-F (%)
Estático	96.4	80.3	87.6
Incremental	96.4	80.3	87.6
Estático Ingênuo	94.4	84.6	89.2

Tabela 4.4: Desempenho - ProxCluster Estático vs Incremental - CD Information

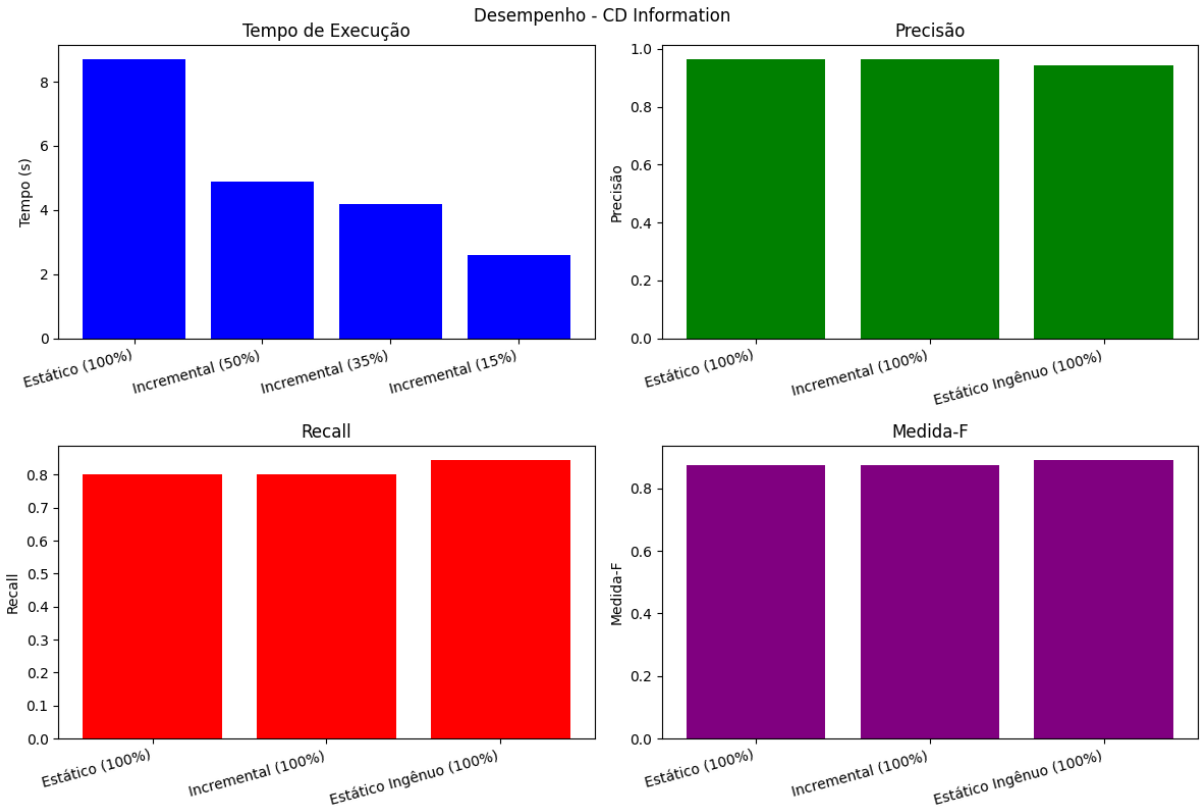

Figura 4.2: CD Information - Estático vs Incremental

Fonte: Autoria própria

4.3.2 MusicBrainz

Conforme a Tabela 4.5, similarmente aos resultados do tempo de processamento para a base de CD (Tabela 4.3), cada uma das execuções incrementais obtiveram valores menores comparados ao processamento estático da base MusicBrainz, com a última partição incremental dos 15% restantes alcançando um ganho de 65% de tempo de processamento comparado ao processo Estático.

ProxCluster	**Incremento (%)**	**Tempo (s)**
Estático	100	17.2
Incremental	50	10.2
Incremental	35	9.5
Incremental	15	6
Estático Ingênuo	100	2129

Tabela 4.5: Tempo de processamento - ProxCluster Estático vs Incremental - MusicBrainz

Quanto as comparações de desempenho, a Figura 4.3 mostra que os valores de *recall* se

mantiveram os mesmos para a estratégia Estática e Incremental, já a precisão viu um aumento de 2.1% no Incremental, isso pode ocorrer já que a ordem que os registros são enviados para o ProxCluster pode impactar no processo de clusterização.

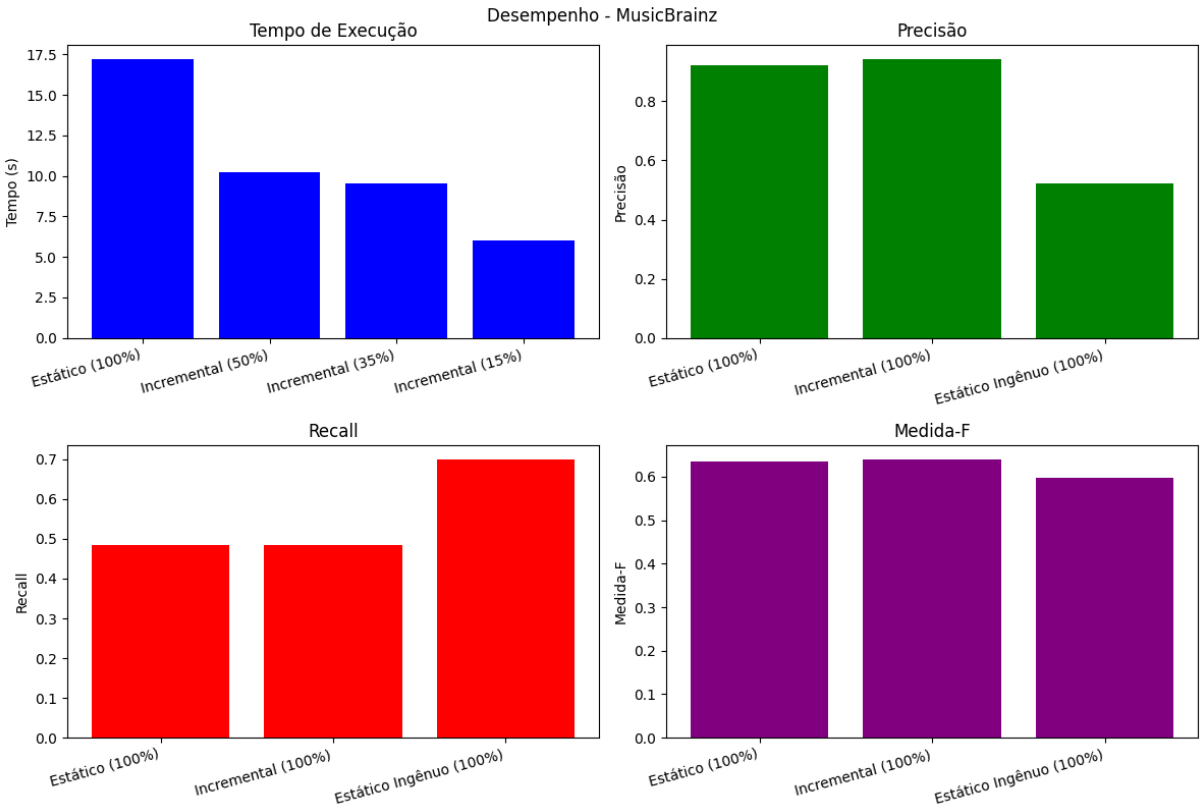

Figura 4.3: MusicBrainz - Estático vs Incremental
Fonte: Autoria própria

A base de dados MusicBrainz é bastante "poluída"e com bastante valores faltando, isso impacta tanto no processo de clusterização, ao medir a distância dos registros, como também no processo de blocagem, pois as variações dos títulos de cada música podem acabar distribuindo-os em diferentes blocos, como também, os registros com títulos faltando acabam no mesmo bloco sem necessariamente possuírem similaridade. Para verificar se de fato é a blocagem a principal responsável pelos valores baixos de *recall* obtidos nos processamentos Estático e Incremental, a execução de Estático Ingênuo foi realizada.

Com o resultado obtido no Estático Ingênuo, ou seja, sem a etapa de blocagem, houve a redução drástica da quantidade de FN, resultando no aumento de 21.6% do valor de *recall*, indicando, assim, a blocagem como principal responsável pelo desempenho abaixo do *recall* nos experimentos Estático e Incremental da base MusicBrainz.

Quanto ao valor da precisão do Estático Ingênuo, houve uma queda grande de 40% comparado ao Estático, isso se deve mais uma vez, ao aumento de FP, resultantes do acréscimo de comparações executadas pelo algoritmo sem utilizar da etapa de blocagem, isso ocasionou em um impacto negativo ainda maior que no caso do CD Information, devido a base MusicBrainz

ser bem mais poluída (atrapalhando no cálculo da distância) e o *threshold* utilizado para os experimentos com a MusicBrainz ser maior.

ProxCluster	**Precisão (%)**	**Recall (%)**	**Medida-F (%)**
Estático	92.2	48.5	63.6
Incremental	94.3	48.5	64.1
Estático Ingênuo	52.2	70.1	59.9

Tabela 4.6: Desempenho - ProxCluster Estático vs Incremental - MusicBrainz

4.3.3 Geographic Settlements

Os resultados do tempo de processamento para a base de Geographic Settlements, Tabela 4.7, foi similar aos anteriores, com a última partição incremental alcançando um ganho de 63% de tempo de processamento comparado a execução com o ProxCluster Estático.

ProxCluster	**Incremento (%)**	**Tempo (s)**
Estático	100	1.9
Incremental	50	1.4
Incremental	35	0.2
Incremental	15	0.7
Estático Ingênuo	100	1444

Tabela 4.7: Tempo de processamento - ProxCluster Estático vs Incremental - Geographic Settlements

A Tabela 4.8 e seus respectivos gráficos da Figura 4.4, mostram como o desempenho do ProxCluster Estático e Incremental foram semelhantes entre si, com um pequeno aumento tanto na precisão quanto no *recall* na execução Incremental. Mais uma vez, o experimento Estático Ingênuo foi utilizado para analisar o impacto da blocagem, entretanto, diferentemente das outras bases, não houve melhoria no valor de *recall*, que reduziu um pouco. Quanto a precisão, despencou 28.4%.

Diferentemente das bases anteriores, todas as métricas de qualidade pioraram para o experimento Estático Ingênuo. Isso se deve ao aumento enorme da quantidade de FP e um leve decréscimo na quantidade de VP. Esses resultados encontrados no experimento são uma consequência de múltiplos fatores, incluindo a função de distância, o número limitado de atributos e a presença de valores ausentes. Com apenas três atributos disponíveis - *label*, latitude e longitude - a quantidade de informações para a análise de similaridade é restrita. Isso dificulta a diferenciação entre duplicatas e registros únicos, uma vez que a variação das informações é limitada. Outro fator impactante é a presença de valores ausentes nas coordenadas de latitude

e longitude em alguns registros. Isso impossibilita a comparação desses atributos quando as informações estão ausentes. A consequência disso é que muitas comparações dependem exclusivamente das *labels*, que muitas vezes, também possuem muitas diferenças entre si mesmo em comparação de duplicatas.

Essa combinação complexa de elementos contribui para uma capacidade limitada de identificar corretamente as duplicatas, destacando a necessidade de considerar as particularidades da base de dados ao interpretar seus resultados.

ProxCluster	**Precisão (%)**	**Recall (%)**	**Medida-F (%)**
Estático	91.1	49.9	64.4
Incremental	91.2	50.0	64.6
Estático Ingênuo	62.7	49.3	55.2

Tabela 4.8: Desempenho - ProxCluster Estático vs Incremental - Geographic Settlements

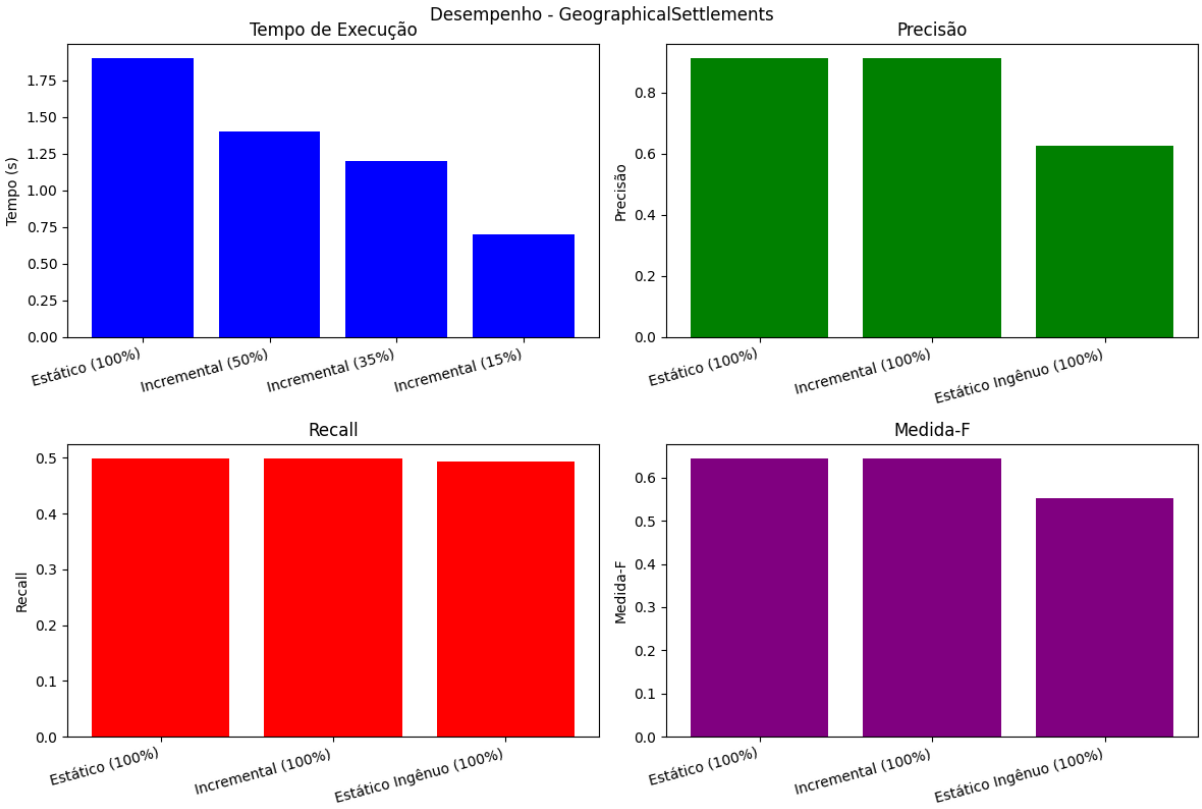

Figura 4.4: Geographical Settlements - Estático vs Incremental

Fonte: Autoria própria

4.4 ProxCluster vs DuDe

Por fim, o último experimento buscou a comparação do desempenho e da eficácia do ProxCluster com um framework alternativo, apresentado na subseção Trabalhos Relacionados do capitulo 2, o DuDe. Foram realizadas análises quanto as métricas de qualidade e tempo de processamento. Esses resultados proporcionam percepções sobre a posição competitiva do ProxCluster em relação a outras soluções existentes na literatura.

O experimento foi realizado utilizando como base as instruções presentes em DRAISBACH; NAUMANN (2010) para RE com a base de CD Information (ofertada pelos mesmos autores), adaptando, entretanto, alguns pontos para que o processo seja o mais semelhante possível com os métodos e estratégias utilizadas no ProxCluster. A blocagem utilizada pelo DuDe foi trocada para a NaiveBlockingAlgorithm, que realiza a blocagem com a mesma estratégia base do SoundexBlocking, isto é, todos os registros que possuem a mesma BKV são agrupados no mesmo bloco. Note que, ainda assim, são processos com resultados diferentes, pois enquanto a BKV do ProxCluster utilizando o SoundexBlocking é calculada com o algoritmo Soundex, o DuDe com o NaiveBlockingAlgorithm utiliza outro método (não especificado na documentação) baseado no valor textual. Quanto a chave de blocagem, foi alterado para utilizar apenas o título, assim como foi feito no experimento do ProxCluster. Por fim, o *threshold* escolhido foi equivalentemente o mesmo utilizado no ProxCluster, considerando que para o DuDe, é comparado a similaridade entre os registros, não a distância entre eles.

Os pares de duplicatas gerados pelo DuDe foram exportados e importados para serem avaliados pelo módulo Evaluator desenvolvido no trabalho, dessa maneira, garantindo que as mesmas métricas e estratégias serão utilizadas para a avaliação de ambos os processos.

O desempenho, apresentado na Tabela 4.9 e seus respectivos gráficos da Figura 4.5, demonstram que em termos de tempo de execução, o DuDe possui uma larga vantagem, isso se deve ao algoritmo de blocagem do mesmo gerar menos candidatos a pares de duplicatas, como também por ser desenvolvido com Java, uma linguagem compilada, em contrapartida ao Python (linguagem do ProxCluster), que é interpretada, o que introduz uma camada adicional de interpretação do código e pode levar a um desempenho relativamente mais lento.

Algoritmo	**Tempo (s)**	**Precisão (%)**	**Recall (%)**	**Medida-F (%)**
ProxCluster	8.6	96.4	80.3	87.6
DuDe	0.8	96.6	48.2	64.3

Tabela 4.9: Desempenho - ProxCluster vs DuDe - CD Information

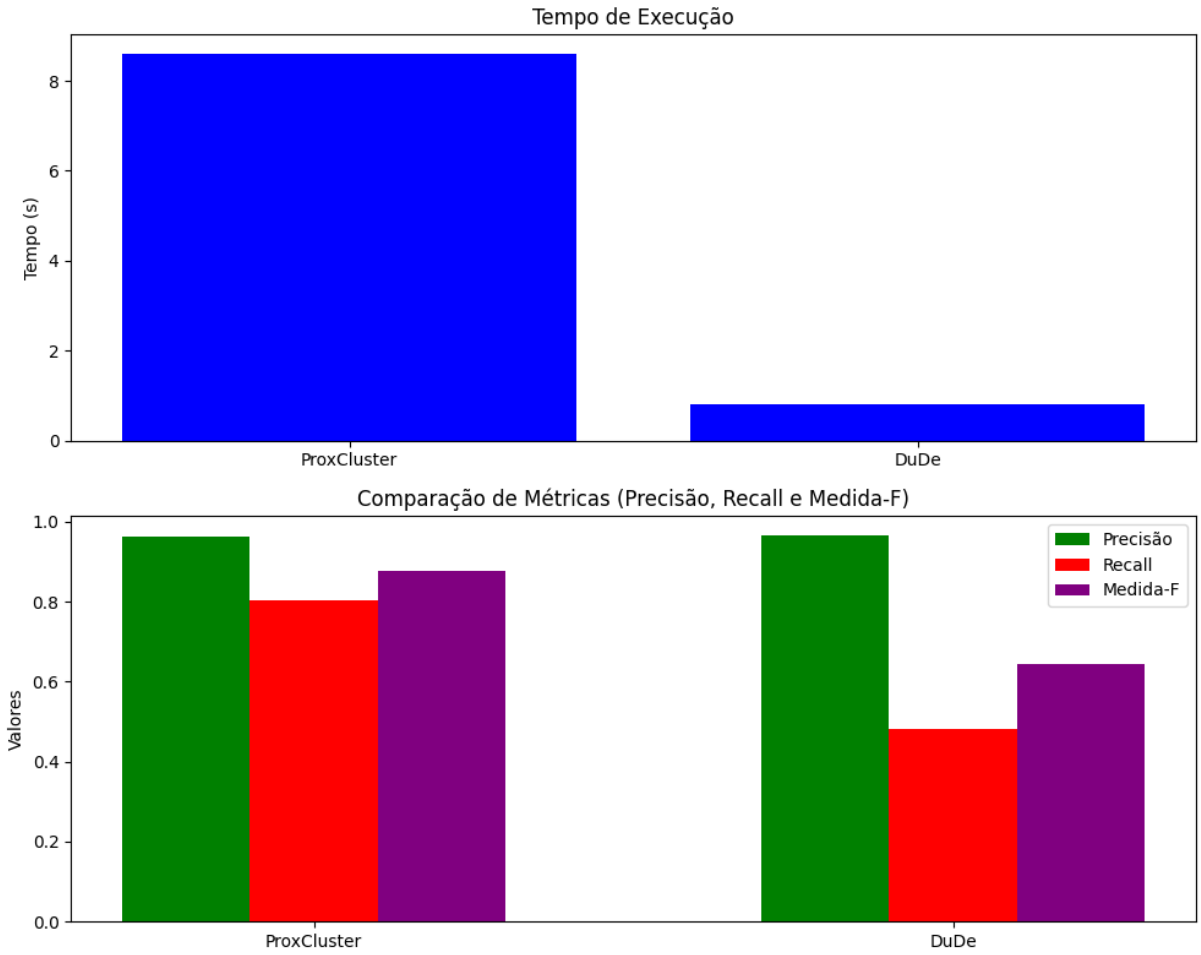

Figura 4.5: ProxCluster vs DuDe
Fonte: Autoria própria

Quanto à precisão, o resultado foi próximo ao DuDe (diferença de 0.2%), mas o ponto de maior diferença foi o *recall*, o ProxCluster atingiu um valor bem melhor que o concorrente, 32.1% maior. O DuDe gerou muito mais FN, assim, reduzindo o valor do *recall*.

Comparando o ProxCluster e o Dude, é possível concluir que no balanço de precisão e *recall* (Medida-F) para a base CD Information, o ProxCluster obteve melhor resultado que o DuDe. Entretanto, o tempo de processamento do DuDe se mostrou bastante superior, além do mesmo ofertar muito mais componentes, funções utilitárias e algoritmos para RE. Todavia, falta algoritmos e soluções para estratégias de REI, diferente do ProxCluster, que possui uma solução incremental.

5

Conclusão

O presente trabalho abordou a RE e a importância deste problema na área de banco de dados, uma vez que a correta identificação e agrupamento de entidades são fundamentais para uma variedade de aplicações, incluindo recuperação de informações, integração e análise de dados. A crescente complexidade das bases de dados e a proliferação de informações tornaram a RE uma tarefa ainda mais crucial em um processo de integração de dados, exigindo abordagens inovadoras e eficazes. A diversidade de contextos e tipos de dados reforça a necessidade de abordagens flexíveis e adaptáveis para enfrentar os desafios específicos de cada domínio. Nesse contexto, a presente pesquisa buscou oferecer uma nova abordagem para a clusterização de entidades, incluindo uma estratégia incremental que se mostrou especialmente valiosa diante da constante evolução e expansão das bases de dados.

A solução proposta neste estudo, denominado de ProxCluster, se insere como mais um avanço na literatura, ao introduzir uma nova perspectiva que traz uma estratégia adaptada do K-Means para REI. Os experimentos realizados para avaliar o desempenho da estratégia incremental em comparação com a abordagem estática revelaram resultados promissores. Ao serem aplicados a três bases de dados distintas, os experimentos evidenciaram que a estratégia incremental não apenas acompanha as mudanças nas bases de dados, concluindo a resolução em menor tempo que a estratégia estática necessitaria, como também mantém a qualidade dos clusters formados.

Além disso, foi realizado um experimento comparando o ProxCluster com o DuDe, que foi utilizado com parametrizações e configurações semelhantes ao ProxCluster (conforme detalhado na seção 4.4), possibilitando, dessa forma, uma análise do desempenho dos processos realizados pelo ProxCluster comparado a uma das grandes opções presentes na literatura. Com o resultado do experimento, foi indicado que o ProxCluster apresenta um desempenho geral superior ao DuDe em relação as métricas de qualidade observadas, devido ao *recall* obtido pelo ProxCluster ter sido muito superior, juntamente com a precisão que obteve uma desvantagem insignificativa, a métrica de qualidade, medida-F, que relaciona precisão e *recall*, foi 23.3% superior em favor do ProxCluster. Sendo este um ponto relevante para assegurar a qualidade dos resultados gerados pelo ProxCluster.

5.1 Principais Contribuições

As principais contribuições com o desenvolvimento do ProxCluster serão apresentadas a seguir:

- Sintetização do algoritmo utilizado como base para o presente trabalho, como também, síntese dos algoritmos aqui desenvolvidos;
- Generalização, aprimoramento e modularização do algoritmo desenvolvido durante a disciplina;
- Desenvolvimento e adoção da estratégia incremental, permitindo que o ProxCluster acompanhe de maneira mais eficaz as mudanças em bases de dados;
- Avaliação da eficácia e desempenho do algoritmo adaptado para a estratégia incremental comparado a abordagem estática, demonstrando sua capacidade de lidar com a evolução das bases de dados;

5.2 Trabalhos Futuros

Este estudo pavimentou caminhos para possíveis trabalhos futuros, incluindo:

- Extensibilidade da biblioteca, a estrutura modular do ProxCluster permite a inclusão de novas funcionalidades e algoritmos, ampliando o leque de opções disponíveis para a RE;
- A adição de algoritmos de blocagem mais sofisticados, visto que podem aprimorar ainda mais os resultados obtidos, permitindo a escolha de abordagens mais especializadas para diferentes contextos;
- Futuros estudos podem explorar uma variedade mais ampla de bases de dados, testando diferentes combinações de parâmetros, blocagens e comparando com algoritmos alternativas presentes na literatura.

Em conclusão, este trabalho contribuiu para a área de RE ao introduzir uma nova abordagem de clusterização ligada a uma estratégia incremental para identificação e agrupamento de entidades. As contribuições fornecem um bom alicerce para futuras explorações e inovações no campo em constante evolução da RE, e mais especificamente, REI.

Referências

AGGARWAL, C. C.; REDDY, C. K. (Ed.). **Data Clustering**: algorithms and application. [S.l.]: Chapman and Hall/CRC, 2013. 89p.

ALMOGKLEIN. **Pandas 2.0 vs Polars 0.17.7**: battle for supremacy in speed and syntax. Disponível em: <https://betterprogramming.pub/data-duel-pandas-2-0-and-polars-0-17-7-battle-for-supremacy-in-speed-a Acesso em: 29 de ago. de 2023.

BLEIHOLDER, J.; NAUMANN, F. Data Fusion. **ACM Computing Surveys**, [S.l.], 2009.

CHAUDHARY, P. **Pandas 2.0 vs Polars**: the ultimate battle. Disponível em: <https://medium.com/cuenex/pandas-2-0-vs-polars-the-ultimate-battle-a378eb75d6d1>. Acesso em: 20 de jul. de 2023.

CHRISTEN, P. A Comparison of Personal Name Matching: techniques and practical issues. **in 'The Second International Workshop on Mining Complex Data (MCD'06)**, [S.l.], dez. 2006.

CHRISTEN, P. Febrl -: an open source data cleaning, deduplication and record linkage system with a graphical user interface. **Association for Computing Machinery**, [S.l.], 2008.

CHRISTEN, P. **Data Matching**: concepts and techniques for record linkage, entity resolution, and duplicate detection. [S.l.]: Springer, 2012.

CHRISTOPHIDES, V. et al. An Overview of End-to-End Entity Resolution for Big Data. **Association for Computing Machinery**, [S.l.], jun. 2020.

DRAISBACH, U.; NAUMANN, F. DuDe: the duplicate detection toolkit. **VLDB**, [S.l.], 2010.

GETOOR, L.; MACHANAVAJJHALA, A. Entity resolution: theory, practice open challenges. **Proceedings of the VLDB Endowment**, [S.l.], v.5, ago. 2012.

LEIPZIG, A. D. **Benchmark datasets for entity resolution**. Disponível em: <https://dbs.uni-leipzig.de/research/projects/object_matching/benchmark_datasets_for_entity_resolution>. Acesso em: 26 de jun. de 2023.

LUNA, J. C. **High Performance Data Manipulation in Python**: pandas 2.0 vs. polars. Disponível em: <https://www.datacamp.com/tutorial/high-performance-data-manipulation-in-python-pandas2-vs-polars>. Acesso em: 20 de jul. de 2023.

MAIMON, O.; ROKACH, L. **Data mining and knowledge discovery handbook**. [S.l.]: Springer, 2010. v.14.

MCNEILL, W.; KARDES, H.; BORTHWICK, A. Dynamic Record Blocking: efficient linking of massive databases in mapreduce. , [S.l.], 2012.

MOHD, W. et al. **MaxD K-Means**: a clustering algorithm for auto-generation of centroids and distance of data points in clusters. [S.l.: s.n.], 2012. v.316, p.192–199.

OLIVEIRA, D. et al. Evaluating Code Readability and Legibility: an examination of human-centric studies. **2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)**, [S.l.], set. 2020.

PAPENBROCK, T.; HEISE, A.; NAUMANN, F. Progressive Duplicate Detection. **IEEE Transactions on Knowledge and Data Engineering**, [S.l.], 2015.

SALES, M. G. G. et al. **Uma proposta para identificar duplicidade em um dataset de informações sobre músicas, envolvendo técnicas de blocagem, distância Levenshtein e clusterização com K-means**. Disponível em: <https://github.com/Gust4voSales/duplicates-clusterization/tree/main>. Acesso em: 26 de jun. de 2023.

THE Soundex Indexing System. Disponível em: <https://www.archives.gov/research/census/soundex>. Acesso em: 19 de jul. de 2023.

THORBEN. **SOLID Design Principles Explained**: the single responsibility principle. Disponível em: <https://stackify.com/solid-design-principles/>. Acesso em: 16 de jul. de 2023.

THWEL, T. T.; SINHA, D. G. R. **Data Deduplication Approaches**: concepts, strategies, and challenges. [S.l.]: Academic Press, 2020. 1p.

VIEIRA, P. K. M.; LOSCIO, B. F.; SALGADO, A. C. Dynamic Indexing for Incremental Entity Resolution in Data Integration Systems. **Proceedings of the 19th International Conference on Enterprise Information Systems**, [S.l.], 2017.

VIEIRA, P. K. M.; LOSCIO, B. F.; SALGADO, A. C. Incremental entity resolution process over query results for data integration systems. **Journal of Intelligent Information Systems**, [S.l.], 2019.