UNIVERSIDADE FEDERAL DE SÃO CARLOS
CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA
DEPARTAMENTO DE COMPUTAÇÃO

Kaique Venuto Mancuzo

Análise de Similaridade em Código Fonte Usando Sinal 1D

São Carlos - SP

2025

Kaique Venuto Mancuzo

Análise de Similaridade em Código Fonte Usando Sinal 1D

Trabalho de Graduação do Curso de Graduação em Engenharia de Computação da Universidade Federal de São Carlos para a obtenção do título de bacharel em Engenharia de Computação.

Orientação Prof. Dr. André Ricardo Backes

São Carlos - SP

2025

Agradecimentos

Agradeço primeiramente aos meus pais, Edson e Patrícia, que são minhas maiores influências e me ofereceram o melhor apoio possível ao longo da minha graduação.

Aos meus amigos, Fernando e Guilherme, por todos os momentos e conversas ao longo desses anos, fundamentais para minha caminhada.

Ao ao meu orientador, Professor André Backes, por sua paciência, atenção, pelas correções feitas e por toda a dedicação na orientação deste trabalho.

Por fim, agradeço aos meus avós, Aparecida e Francisco, e à minha irmã, Thais, por estarem sempre presentes.

"Quando a neve cai e os ventos brancos sopram, o lobo solitário morre, mas a matilha sobrevive."

(George R. R. Martin)

Resumo

O plágio em código-fonte é um problema recorrente no meio acadêmico, e sua detecção manual torna-se inviável devido ao alto volume de trabalhos desenvolvidos em disciplinas de programação. Diante disso, este trabalho propõe uma abordagem que mede a similaridade entre códigos-fonte por meio de técnicas de processamento de sinais, tratando-os como sinais unidimensionais. Parte-se da hipótese de que essa abordagem pode ser mais resistente a técnicas de ofuscação do que os métodos convencionais. Foram exploradas três abordagens: análise no domínio do tempo, com Transformada de Fourier e Transformada de Wavelet. As métricas aplicadas foram avaliadas em bases de códigos contendo casos previamente identificados de plágio e comparadas com as ferramentas MOSS e JPlag. Os resultados indicam que a análise no domínio do tempo, especialmente com a Distância DTW e a Correlação de Pearson, apresentou maior eficácia na identificação de plágio, aproximando-se do desempenho das ferramentas tradicionais.

Palavras-chave: Plágio, análise de similaridade, processamento de sinais

Abstract

Source code plagiarism is a recurring issue in academia, and its manual detection is a highly time-consuming task due to the large volume of assignments in programming courses. To address this, this study proposes an approach to measuring similarity between source codes using signal processing techniques, treating them as one-dimensional signals. The hypothesis is that this approach may be more resistant to obfuscation techniques than conventional methods. Three approaches were explored: time-domain analysis, Fourier Transform, and Wavelet Transform. The applied metrics were evaluated on datasets containing previously identified cases of plagiarism and compared with the MOSS and JPlag tools. The results indicate that time-domain analysis, particularly with Dynamic Time Warping (DTW) distance and Pearson correlation, was the most effective in identifying plagiarism, achieving performance comparable to traditional tools. Abstract in English

Keywords: Plagiarism, similarity analysis, signal processing

Lista de ilustrações

Lista de abreviaturas e siglas

CIL	*Common Intermediate Language*
DTW	*Dynamic Time Warping*
DFT	*Discrete Fourier Transform*
DWT	*Discrete Wavelet Transform*
EMD	*Earth Mover's Distance*
FFT	*Fast Fourier Transform*
GCC	*GNU Compiler Collection*
JPlag	*Java Plagiarism Detection Tool*
MOSS	*Measure of Software Similarity*

Sumário

1 Introdução

No contexto acadêmico, o plágio é uma prática antiética que se caracteriza pela reutilização do trabalho ou das ideias de outra pessoa sem a devida autorização ou atribuição ao autor original (ĐURIĆ; GAŠEVIĆ, 2013; KUSTANTO; LIEM, 2009). Essa prática tem se mostrado comum em ambientes educacionais e também se manifesta de forma recorrente nos cursos que incluem disciplinas relacionadas à computação, representando um grande desafio enfrentado na área (MOZGOVOY; KAKKONEN; COSMA, 2010). Em uma pesquisa com 55 estudantes da USP (BAZALUK et al., 2023), 23,6% dos participantes admitiram ter cometido algum tipo de plágio em código-fonte. De forma similar, em um outro estudo realizado com acadêmicos de diversas instituições do Reino Unido (CHUDA et al., 2011), dos 313 alunos entrevistados, 42% afirmaram já ter sido alvo de plágio, enquanto 33% admitiram ter plagiado trabalhos, evidenciando a recorrência de tal prática.

O plágio em código-fonte pode ser definido como a reutilização de código originalmente criado por outra pessoa, sem o devido reconhecimento ao autor, seja de forma intencional ou acidental (COSMA; JOY, 2008). A crescente facilidade de acesso dos estudantes a uma vasta gama de recursos disponíveis na internet, como repositórios de código, fóruns especializados e plataformas colaborativas, combinada com a troca de soluções entre colegas, cria um ambiente com condições favoráveis para a disseminação da prática de plágio nas submissões acadêmicas. Essa dinâmica facilita reprodução de respostas prontas, ao invés da compreensão dos conceitos e do desenvolvimento das habilidades necessárias para realizar as tarefas propostas (BURROWS; TAHAGHOGHI; ZOBEL, 2007).

Devido ao grande número de estudantes em turmas de disciplinas de programação e à natureza dessas matérias, que frequentemente exigem múltiplas submissões de trabalhos práticos, a avaliação manual de plágio pelo corpo docente é uma tarefa impraticável (SRAKA; KAUCIC, 2009). Para enfrentar essa problemática, diversas ferramentas e *frameworks* foram desenvolvidos para automatizar a detecção de plágio em código-fonte. Algumas das mais bem sucedidas são o JPlag (PRECHELT et al., 2002), e o MOSS (*Measure of Software Similarity*) (SCHLEIMER; WILKERSON; AIKEN, 2003). Essas ferramentas utilizam algoritmos de análise de cadeias de caracteres para transformar códigos-fonte em representações de *tokens*, abstraindo elementos como nomes de variáveis e formatações. Em seguida, os *tokens* são comparados entre pares de arquivos para identificar padrões de similaridade em segmentos de código, permitindo o cálculo do grau de correspondência entre eles (COSMA; JOY, 2011).

Embora eficazes em muitos casos, essas técnicas baseadas em análise sintática apresentam limitações significativas quando confrontadas com estratégias de ofuscação e transformações de código que preservam a semântica (LUO et al., 2014). Essas técnicas de plágio podem incluir

alterações como a substituição de identificadores, reordenação de comandos, introdução de operações redundantes ou até mesmo modificações no formato do código, que, embora não alterem seu comportamento funcional, podem confundir métodos que dependem exclusivamente da análise de *tokens* ou da estrutura textual (KARNALIM, 2016).

Como alternativa a essas limitações, foram propostas soluções para análise de plágio em código-fonte baseadas na representação de código em baixo nível (JURIČIĆ, 2011; KARNALIM, 2021; JI; WOO; CHO, 2008). Essas abordagens utilizam as conversões intermediárias realizadas por compiladores e interpretadores, oferecendo vantagens como a abstração de detalhes específicos das linguagens e a preservação da semântica do programa, oferecendo maior robustez contra transformações que introduzem operações redundantes ou reorganizam comandos, além de ignorar elementos como comentários e construções sintáticas que não impactam a lógica do código (RABBANI; KARNALIM, 2017).

Considerando a representação de baixo nível gerada por compiladores para códigos em linguagem C, o presente estudo propõe uma abordagem que analisa os códigos gerados como sinais unidimensionais. Explorando técnicas e métricas da área de análise e processamento de sinais, este estudo busca avaliar a similaridade entre códigos compilados, com o propósito de identificar possíveis casos de plágio. O trabalho investiga a eficácia dessa abordagem em comparação com métodos já consolidados, analisando os resultados em relação a ferramentas amplamente utilizadas na área, como MOSS e JPlag.

Este trabalho está estruturado da seguinte forma: o capítulo 2 apresenta a fundamentação teórica necessária para os métodos utilizados, enquanto o capítulo 3 aborda as ferramentas mais populares para detecção de plágio, incluindo estudos sobre aquelas que analisam código em baixo nível. No capítulo 4, são descritos os detalhes do desenvolvimento do estudo, seguido pelo capítulo 5, que apresenta os resultados obtidos. Por fim, o capítulo 6 traz as conclusões.

1.1 Objetivos

O objetivo geral deste trabalho é, a partir do uso de métodos de processamento de sinais baseados na representação de baixo nível dos códigos, investigar e identificar as métricas e técnicas mais adequadas para analisar similaridade entre códigos-fonte, comparando as métricas propostas com as ferramentas disponíveis no estado da arte.

Para alcançar esse objetivo geral, os objetivos específicos são:

- Implementar uma abordagem para tratar os arquivos binários gerados por compiladores como sinais unidimensionais, criando um ambiente de teste para análises com técnicas de processamento de sinais;
- Identificar e selecionar métricas e técnicas de similaridade adequadas para avaliar plágio em códigos-fonte a partir de seus binários compilados;

- Implementar e testar as métricas propostas utilizando uma base de dados contendo casos conhecidos de plágio em código-fonte;
- Comparar o desempenho das métricas desenvolvidas com ferramentas existentes no estado da arte.
- Analisar os resultados obtidos, identificando limitações e propondo novas investigações para futuros trabalhos.

2 Fundamentação teórica

Este trabalho adota uma abordagem baseada na representação de baixo nível dos códigos-fonte, tratando-os como sinais unidimensionais e assim viabilizando técnicas de processamento de sinais. Ao utilizar uma representação de baixo nível, a análise torna-se mais resiliente a técnicas de ofuscação e a transformações superficiais, uma vez que alterações como a reorganização de comandos ou a substituição de identificadores têm impacto reduzido na estrutura subjacente.

Este capítulo apresenta os conceitos e fundamentos teóricos que embasam o desenvolvimento deste estudo, fornecendo o contexto necessário para compreender a relevância e os métodos empregados na análise proposta.

2.1 Compiladores e o Processo de Geração de Arquivos Binários

O processo de tradução de código-fonte escrito em linguagens de programação de alto nível, como C, para um formato executável compreensível pelos processadores dos sistemas computacionais envolvem uma série de etapas realizadas por ferramentas chamadas de compiladores (COOPER; TORCZON, 2022).

Segundo (ALFRED; MONICA; JEFFREY, 2007), os compiladores operam em uma sequência de fases em que cada uma transforma a representação do código fonte, avançando em direção ao formato final. Essas fases, em ordem, são:

- **Análise léxica:** Também chamada de *scanning*, é a fase inicial em que é processada a sequência de caracteres que constitui o código-fonte em componentes chamados de lexemas, para serem usados na fase seguinte.
- **Análise sintática:** Resulta em uma árvore sintática a partir dos lexemas, na qual cada nó interno representa uma operação, e os filhos do nó representam os argumentos dessa operação.
- **Análise semântica:** O analisador semântico verifica a consistência do programa com a definição da linguagem, utilizando a árvore sintática e a tabela de símbolos. Ele também realiza verificação de tipos e, quando permitido, aplica coerções, como a conversão de inteiros para ponto flutuante em operações mistas.
- **Geração de código intermediário:** Os compiladores, nesta etapa, são capazes de criar uma ou mais representações intermediárias, como árvores sintáticas, que auxiliam na organização e análise do programa. Após essa fase, é gerada uma representação intermediária de baixo nível.

- **Otimização de código:** A etapa de otimização de código busca melhorar o código intermediário para gerar um código final mais eficiente.

- **Geração de código:** Na geração de código, o compilador transforma a representação intermediária em uma linguagem de destino, como código de máquina. Essa fase envolve a seleção de registradores ou locais de memória para variáveis e a tradução de instruções intermediárias em instruções de máquina.

A tabela de símbolos, responsável por armazenar informações sobre o código-fonte, como nomes de variáveis, tipos de dados, escopos e localização na memória, é uma estrutura utilizada por todas as fases do compilador. Durante a análise léxica, ela registra identificadores encontrados no código. Na análise semântica, verifica a consistência de tipos e associações.

2.2 Medidas de similaridade no domínio do tempo

As seguintes medidas de distância, para identificar e quantificar similaridade entre sinais, foram aplicadas no desenvolvimento deste estudo.

2.2.1 Distância Euclidiana

A distância euclidiana entre dois vetores corresponde à menor distância linear entre os pontos que eles representam em um espaço multidimensional (BOULOS; CAMARGO, 1987), sendo amplamente utilizada como métrica de similaridade.

Matematicamente, a distância euclidiana entre dois vetores $a = [a_1, a_2, \ldots, a_n]$ e $b = [b_1, b_2, \ldots, b_n]$ é calculada como:

$$d(a,b) = \sqrt{\sum_{i=1}^{n} (a_i - b_i)^2} \tag{2.1}$$

2.2.2 Correlação de Pearson

A correlação de Pearson quantifica a intensidade e a direção da relação linear entre dois vetores, sendo útil para avaliar padrões de associação em dados normalizados. (THOMPSON; PANCHEV; OAKES, 2015).

Matematicamente, a correlação de Pearson entre dois vetores $a = [a_1, a_2, \ldots, a_n]$ e $b = [b_1, b_2, \ldots, b_n]$ é calculada como:

$$r = \frac{\sum_{i=1}^{n} (a_i - \bar{a})(b_i - \bar{b})}{\sqrt{\sum_{i=1}^{n} (a_i - \bar{a})^2} \sqrt{\sum_{i=1}^{n} (b_i - \bar{b})^2}} \tag{2.2}$$

2.2.3 Distância Manhattan

A distância Manhattan, também conhecida como distância de taxista, mede a soma das diferenças absolutas entre os elementos correspondentes de dois vetores.

De acordo com (KRAUSE, 1973), essa métrica é inspirada na geometria de uma grade urbana, onde a distância entre dois pontos é calculada percorrendo caminhos ao longo de eixos ortogonais, como ruas e avenidas em uma cidade. Diferentemente da distância euclidiana, que considera a menor distância linear, a distância Manhattan reflete o deslocamento necessário ao longo de um sistema estruturado em "blocos", contabilizando separadamente os movimentos horizontais e verticais.

Matematicamente, para dois vetores $a = [a_1, a_2, \ldots, a_n]$ e $b = [b_1, b_2, \ldots, b_n]$, a distância Manhattan é dada por:

$$d(a,b) = \sum_{i=1}^{n} |a_i - b_i| \tag{2.3}$$

2.2.4 Distância DTW (*Dynamic Time Warping*)

A distância DTW (*Dynamic Time Warping*) é um algoritmo que utiliza a abordagem de programação dinâmica para calcular a similaridade entre duas sequências, mesmo quando apresentam variações no tempo ou no espaço. Ele funciona ajustando as sequências de forma não linear, permitindo que sejam "esticadas" ou "comprimidas" ao longo do eixo temporal para encontrar o melhor alinhamento entre seus pontos (MATUSCHEK; SCHLÜTER; CONRAD, 2008; YADAV; ALAM, 2018).

Sejam $X = \{x_1, x_2, ..., x_n\}$ e $Y = \{y_1, y_2, ..., y_m\}$ duas sequências temporais de comprimento n e m, respectivamente. Definimos a matriz de custo acumulado $D(i, j)$ como:

$$D(i,j) = |x_i - y_j| + \min \begin{cases} D(i-1, j) \\ D(i, j-1) \\ D(i-1, j-1) \end{cases} \tag{2.4}$$

com as condições iniciais:

$$D(0,0) = 0, \quad D(i,0) = \sum_{k=1}^{i} |x_k - y_1|, \quad D(0,j) = \sum_{k=1}^{j} |x_1 - y_k| \tag{2.5}$$

a distância DTW entre as sequências X e Y é então definida como o custo acumulado ótimo ao final do alinhamento:

$$\mathrm{DTW}(X,Y) = D(n,m) \tag{2.6}$$

onde $D(n,m)$ representa o custo total mínimo para alinhar as duas sequências considerando as distorções temporais permitidas pelo algoritmo.

2.2.5 Distância EMD (*Earth Mover's Distance*)

A *Earth Mover's Distance* (EMD), também conhecida como distância de Wasserstein, é uma métrica de distância entre distribuições. Intuitivamente, ela mede o menor esforço necessário para transformar uma distribuição em outra, considerando o custo de transporte, que é proporcional a quantidade de "massa" movida e à distância percorrida (RUBNER; TOMASI; GUIBAS, 1998)

Para duas distribuições unidimensionais, pode ser definida matematicamente como (RAMDAS; TRILLOS; CUTURI, 2017):

$$l_1(u,v) = \inf_{\pi \in \Gamma(u,v)} \int_{R\times R} |x-y| \, d\pi(x,y) \tag{2.7}$$

2.2.6 Similaridade Cosseno

Também conhecida como coeficiente de Ochiai, a similaridade do cosseno é uma métrica utilizada para medir a relação entre dois vetores em um espaço vetorial. Diferente de outras medidas baseadas em distância, ela considera a orientação dos vetores, sendo independente de sua magnitude. (WANG; DONG, 2020)

$$\text{sim}(\mathbf{a},\mathbf{b}) = \frac{\mathbf{a}\cdot\mathbf{b}}{\|\mathbf{a}\|\|\mathbf{b}\|} \tag{2.8}$$

onde:

- $\mathbf{a}\cdot\mathbf{b} = \sum_{i=1}^{n} a_i b_i$ é o produto escalar dos vetores $\mathbf{a}$ e $\mathbf{b}$.
- $\|\mathbf{a}\| = \sqrt{\sum_{i=1}^{n} a_i^2}$ e $\|\mathbf{b}\| = \sqrt{\sum_{i=1}^{n} b_i^2}$ são as normas dos vetores $\mathbf{a}$ e $\mathbf{b}$, respectivamente.

Os valores da similaridade cosseno variam entre -1 e 1, sendo interpretados como segue:

- 1: os vetores são perfeitamente alinhados (mesma direção).
- 0: os vetores são ortogonais (sem similaridade).
- -1: os vetores têm direções opostas.

2.3 Transformada de Fourier

A transformada de Fourier é uma ferramenta matemática utilizada para representar funções ou sinais como uma soma ou integral de componentes exponenciais periódicas. Essa

técnica é aplicada na análise de sistemas lineares e no processamento de sinais, permitindo a conversão de informações do domínio do tempo para o domínio da frequência, facilitando sua interpretação e manipulação (BRACEWELL; KAHN, 1966). Além disso, é amplamente utilizada em áreas como compressão de dados, análise de imagens e reconhecimento de padrões.

A transformada discreta de Fourier (DFT) é uma versão da transformada de Fourier que trabalha com dados discretos. Diferentemente da versão contínua, a DFT processa uma sequência finita de valores e a transforma em uma soma de componentes periódicas associadas a frequências discretas (COOLEY; LEWIS; WELCH, 1969).

A DFT calcula diretamente essas componentes, mas sua complexidade computacional cresce de forma quadrática com o tamanho da entrada, sendo $O(N^2)$, onde N é o número de pontos de entrada. Para lidar com esse problema, foi desenvolvido o algoritmo *Fast Fourier Transform* (FFT), que é uma implementação eficiente da DFT, reduzindo sua complexidade para $O(N \log N)$ (PRESS, 2007).

A DFT e sua inversa são definidas matematicamente pelas seguintes equações:

Transformada Discreta de Fourier (DFT):

$$X[k] = \sum_{n=0}^{N-1} x[n] \cdot e^{-j\frac{2\pi}{N}kn}, \quad k = 0, 1, \ldots, N-1 \tag{2.9}$$

Transformada Inversa Discreta de Fourier (IDFT):

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] \cdot e^{j\frac{2\pi}{N}kn}, \quad n = 0, 1, \ldots, N-1 \tag{2.10}$$

2.4 Transformada de Wavelet

Wavelets são ferramentas matemáticas empregadas na análise de sinais, possibilitando sua decomposição em dois tipos de coeficientes: coeficientes de wavelets e coeficientes de escala. Essa decomposição é realizada utilizando filtros wavelet específicos e baseia-se em funções com duração limitada. Essas funções permitem localizar informações tanto no tempo quanto na frequência. O processo de decomposição continua até que o comprimento do filtro exceda o tamanho do vetor wavelet resultante, permitindo uma análise detalhada dos sinais em diferentes escalas.(CHUN-LIN, 2010)

Comparadas a métodos tradicionais, como a Transformada de Fourier, as wavelets apresentam vantagens notáveis. Elas permitem identificar o momento exato em que uma frequência específica ocorre e ajustar a resolução temporal e de frequência de acordo com a escala desejada. Além disso, as wavelets podem reduzir significativamente a quantidade de dados necessária para processamento, o que as torna úteis em aplicações como processamento de imagens, análise de mercados financeiros e detecção de clones de código. (KARUS; KILGI, 2015)

A definição metemática é

$$W_{\psi}(a,b) = \frac{1}{\sqrt{|a|}} \int_{-\infty}^{\infty} f(t)\, \psi^{*} \left(\frac{t-b}{a} \right) dt, \tag{2.11}$$

onde:

- $\psi(t)$ é a função wavelet mãe, que é escalada e transladada para gerar as wavelets-filhas.
- $a \neq 0$ é o parâmetro de escala, que dilata ou comprime a wavelet.
- b é o parâmetro de translação, que desloca a wavelet no tempo.
- ψ^{*} é o complexo conjugado da função $\psi(t)$.

A condição de admissibilidade, necessária para que a transformada seja invertível, é dada por:

$$C_{\psi} = \int_{-\infty}^{\infty} \frac{|\hat{\psi}(\omega)|^2}{|\omega|}\, d\omega < \infty, \tag{2.12}$$

onde $\hat{\psi}(\omega)$ é a transformada de Fourier da wavelet $\psi(t)$.

3 Ferramentas para a detecção de plágio

3.1 Ferramentas mais populares

Algumas das ferramentas mais populares e bem-sucedidas propostas para a detecção de plágio entre códigos fontes são o MOSS, JPLAG e Plaggie que serão apresentadas a seguir

3.1.1 Moss

O Moss *(Measure Of Software Similarity)* (SCHLEIMER; WILKERSON; AIKEN, 2003) é um sistema automatizado para determinar a similaridade entre códigos fontes de programas. Disponível gratuitamente na internet, ele é amplamente utilizado em contextos acadêmicos e educacionais para detectar plágio em trabalhos de programação, permitindo a comparação de múltiplos códigos para avaliar correspondências e similaridades. O Moss é considerado a ferramenta de detecção de plágio mais utilizada, destacando-se por ser generalista, com suporte para 24 linguagens de programação (DEVORE-MCDONALD; BERGER, 2020).

Segundo (SCHLEIMER; WILKERSON; AIKEN, 2003), o MOSS se destaca como detector de plágio pelas seguintes características: capacidade de ignorar alterações superficiais, como espaços em branco, capitalização e comentários; apresenta uma baixa taxa de falsos positivos ao reportar apenas trechos significativos de código copiado, evitando que coincidências triviais sejam identificadas como plágio e pela capacidade de identificar similaridades mesmo quando os trechos plagiados estão reorganizados espacialmente.

No funcionamento do MOSS, o código-fonte é inicialmente dividido em substrings de tamanho k, chamadas de *k-grams*. Cada *k-gram* é convertido em um *hash*, criando uma sequência que representa o conteúdo do programa. O algoritmo de *winnowing* é então aplicado a essa sequência de *hashes* para selecionar as *fingerprints*, que são representações compactas do código. O objetivo do algoritmo é gerar *fingerprints* que contenham informações suficientes para identificar similaridades sem comprometer a precisão na análise. Essas *fingerprints* são usadas para comparar e identificar trechos semelhantes entre arquivos, permitindo detectar correspondências entre os códigos analisados (DEVORE-MCDONALD; BERGER, 2020)

Para a utilização do MOSS, é possível inserir arquivos base que servem como referência para a comparação, geralmente contendo código fornecido pelos instrutores ou padrões que não devem ser considerados plágio. A Figura 1 apresenta a interface gráfica da versão para Windows do MOSS, onde é possível visualizar opções como a escolha de arquivos base, a seleção de arquivos de código-fonte para comparação e a configuração de diferentes parâmetros para a análise.

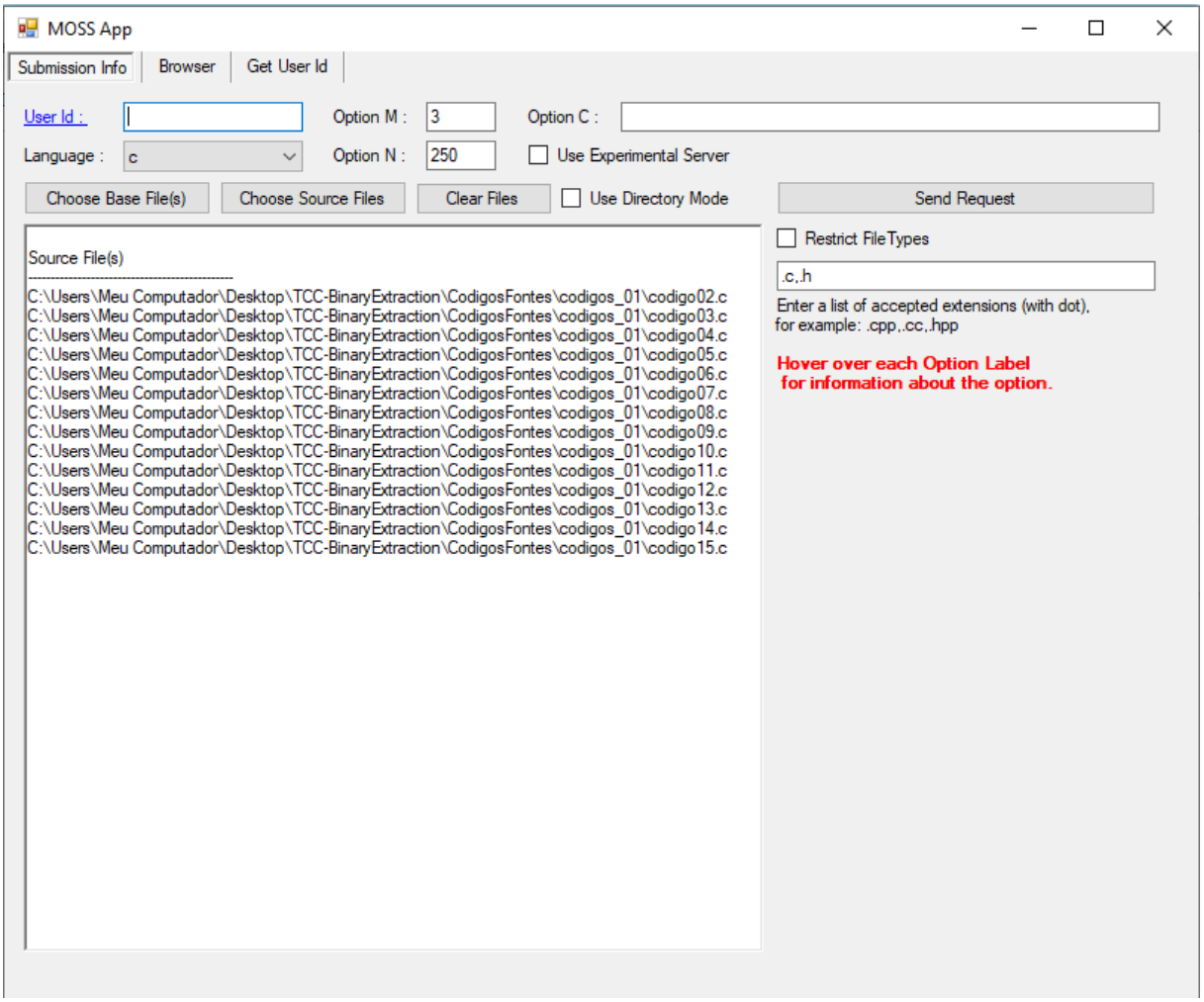

Figura 1 – Interface gráfica da versão para Windows do MOSS

Fonte: O autor.

3.1.2 JPlag

O JPlag (PRECHELT et al., 2002) é uma ferramenta para a detecção de similaridades entre programas, também muito utilizada em contextos acadêmicos para identificar casos de plágio em submissões de código. Assim como o MOSS, o JPlag opera através da comparação de pares de programas dentro de um conjunto fornecido. Inicialmente projetado para a análise de códigos em Java, a ferramenta também oferece suporte às linguagens C, C++ e Scheme.

O Jplag analisa a similaridade entre códigos fontes e exibe um relatório em HTML que classifica as similaridades identificadas em percentuais. O relatório apresenta os resultados destacando pares de programas com maior índice de similaridade. Além disso, o JPlag inclui uma interface que permite visualizar os códigos comparados lado a lado, com as semelhanças marcadas por meio de cores, para facilitar a identificação das regiões correspondentes.

A arquitetura do JPlag é baseada no algoritmo *Greedy String Tiling*, que identifica regiões similares nos códigos-fonte. O algoritmo converte os programas em sequências de tokens que

representam suas estruturas essenciais, ignorando elementos como espaçamento, comentários e nomes de variáveis. Essa abordagem visa detectar similaridades estruturais mesmo em casos onde os códigos tenham sido modificados para alterar a aparência externa.

Os resultados gerados pela ferramenta podem ser visualizados por meio da interface do JPlag Report Viewer, apresentada na Figura 2. Essa interface permite a análise das comparações realizadas, fornecendo uma visualização dos índices de similaridade entre os códigos avaliados. No lado esquerdo, há um gráfico que exibe a distribuição das similaridades entre os pares de códigos, categorizando-os em intervalos percentuais. No lado direito, são listados os pares de arquivos com maior grau de similaridade, apresentando métricas como similaridade média e similaridade máxima.

,

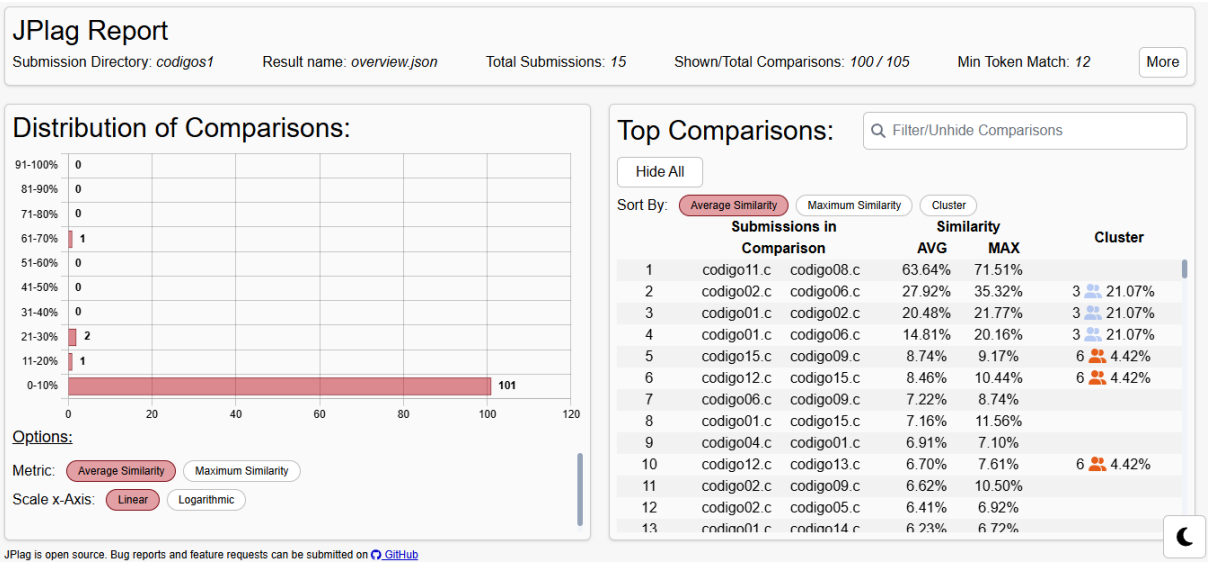

Figura 2 – Interface do JPlag Report Viewer

Fonte: O autor.

3.1.3 Plaggie

Plaggie é uma ferramenta de detecção de plágio em códigos-fonte, (AHTIAINEN; SURAKKA; RAHIKAINEN, 2006), como uma aplicação autônoma voltada para a análise de códigos em Java. A ferramenta foi projetada para abordar limitações identificadas em outras soluções, como a dificuldade em diferenciar código fornecido em exercícios de código produzido pelos estudantes. Plaggie é disponibilizado como um *software* de código aberto, permitindo customizações e ajustes para atender às demandas específicas das instituições.

Plaggie utiliza algoritmos baseados em tokenização e *Greedy String Tiling* para comparar códigos-fonte e identificar similaridades. Diferentemente de ferramentas como JPlag, que funcionam como serviços web, Plaggie é instalado e executado localmente, oferecendo

maior controle sobre os dados submetidos. Essa característica reduz o risco de exposição de informações sensíveis, como dados de identificação dos estudantes. Por ser de código aberto, Plaggie também pode ser integrado a outros sistemas educacionais ou adaptado para realizar testes internos detalhados de detecção de plágio.

3.2 Trabalhos relacionados

Diferentes abordagens têm sido propostas para detectar plágio em códigos-fonte utilizando representações de baixo nível. Ji et al. (JI; WOO; CHO, 2008) propuseram um método para detecção de plágio que utiliza sequências de *bytecode*, uma representação de baixo nível da linguagem Java, em vez de sequências de *tokens*, como ocorre em abordagens tradicionais. Eles argumentaram que essa técnica apresenta a vantagem de dispensar o acesso direto ao código-fonte original, garantindo maior segurança durante o processo de análise.

Em seu estudo, (KARNALIM, 2016) expandiu o trabalho de Ji (JI; WOO; CHO, 2008) incorporando novas técnicas ao processo de detecção, como generalização e interpretação de instruções e tratamento de recursividade. Essas melhorias, segundo os pesquisadores, tornaram o método mais robusto contra ataques de plágio mais avançados, comuns em cursos introdutórios de programação. Além disso, o estudo avaliou tipos específicos de ataques de plágio em Java, classificando-os e analisando a eficácia do método frente a cada tipo identificado.

Em seus trabalhos, (JURIČIĆ, 2011) e (JURIČIĆ; JURIĆ; TKALEC, 2011) exploraram métodos de detecção de plágio em linguagens que utilizam a *Common Intermediate Language (CIL)*, uma representação intermediária da plataforma .NET. Apesar de ambos os trabalhos empregarem o CIL para comparar códigos, eles diferem nos algoritmos de similaridade adotados. Os resultados destacam vantagens no uso do CIL por abstrair detalhes específicos das linguagens, reduzindo a influência de transformações sintáticas, como comentários e formatação de código.

4 Desenvolvimento

Este capítulo destina-se a apresentar o desenvolvimento do trabalho visto o objetivo geral de identificar casos de plágio adotando-se uma abordagem de análise de baixo nível dos códigos:

4.1 Base de Dados

Foi utilizada uma base de dados composta por três conjuntos contendo casos de plágio previamente identificados, fornecida por um professor da Universidade Federal de São Carlos (UFSCar). Esses conjuntos foram utilizados para avaliar a efetividade dos métodos estudados. A base, estruturada com arquivos em linguagem C, é descrita a seguir:

- Base 1: Skiplist
  Contém implementações de operações em *skiplist*, totalizando 14 códigos. Três casos de plágio foram identificados.

- Base 2: Árvore B
  Inclui implementações de operações em árvores B, com um total de 15 códigos. Um caso de plágio foi identificado.

  Base 3: Operações Matemáticas
  Composta por 12 códigos relacionados à manipulação e exibição de resultados de operações matemáticas baseadas em inputs do usuário. Três casos de plágio foram identificados.

4.2 Pré-Processamento e Compilação dos Arquivos

Usando a linguagem Python 3 em um ambiente configurado no Jupyter Notebook (arquivos com extensão .ipynb), foi realizado o processo de compilação dos arquivos de cada base de código. Para isso, foi utilizado o compilador GCC (*GNU Compiler Collection*) com as flags -c e -o, gerando arquivos objeto (.o) a partir de cada arquivo-fonte escrito em linguagem C.

4.3 Análise e Identificação de Similaridade

Após a obtenção dos arquivos objetos de cada base, iniciou-se o processo de análise desses arquivos, considerando-os como sinais unidimensionais (1D), onde cada byte do arquivo representa um ponto do sinal. Foram realizadas três abordagens principais para análise, utilizando diversas métricas de similaridade com o objetivo de investigar as mais efetivas.

4.3.1 Análise no Domínio do Tempo

A primeira abordagem consistiu na análise no domínio do tempo, com a realização de comparações diretas entre os sinais. Para isso, foram aplicadas as seguintes métricas de similaridade: Distância Euclidiana, Distância Manhattan, Similaridade Cosseno, Distância Wasserstein (*Earth Mover's Distance*), Correlação Cruzada, *Dynamic Time Warping* (DTW) e Correlação de Pearson com um ajuste de deslocamento temporal. Esta última mede a correlação entre dois sinais após ajustar o segundo sinal para um atraso ideal, identificado pela maior correlação cruzada. Após o alinhamento, calcula-se o coeficiente de correlação de Pearson entre os sinais.

Para a implementação das métricas de similaridade, foram utilizados os módulos scipy.stats e scipy.spatial.distance da biblioteca SciPy, além do módulo fastdtw para a linguagem Python. Cada métrica foi aplicada individualmente a todos os pares de sinais. Os resultados foram registrados e comparados com casos conhecidos de plágio e com as ferramentas MOSS e JPlag.

4.3.2 Análise com a Transformada de Fourier

Na segunda abordagem os sinais foram analisados no domínio da frequência utilizando a Transformada de Fourier, precedida por etapas de pré-processamento. Esse pré-processamento incluiu a remoção do componente DC, que corresponde à média do sinal no tempo, e a aplicação de um filtro passa-alta configurado com uma frequência de corte de 0,1 (em unidades normalizadas). Essa configuração permitiu descartar frequências muito baixas, preservando apenas as componentes mais relevantes do sinal.

As métricas extraídas do espectro de frequência foram:

- **Energia Total do Espectro**: Soma das magnitudes quadradas do espectro de frequência, representando a quantidade total de energia presente no sinal.
- **Entropia Espectral**: Mede a dispersão das magnitudes no espectro, indicando a uniformidade ou concentração da energia ao longo das frequências.
- **Centro de Gravidade Espectral**: Representa a média ponderada das frequências no espectro, refletindo onde está concentrada a maior parte da energia do sinal.
- **Planicidade Espectral**: Indica o quão uniforme é o espectro. Valores baixos sugerem a presença de picos bem definidos, enquanto valores altos indicam um espectro mais homogêneo ou ruidoso.
- **Largura de Banda (Bandwidth)**: Mede a dispersão das magnitudes em torno da frequência média, indicando o intervalo de frequências que concentra a maior parte da energia do sinal.

Para a implementação da transformada de Fourier, utilizou-se o módulo numpy para a linguagem Python. As métricas mencionadas foram calculadas para cada sinal extraído a partir do espectro de frequência gerado pela Transformada de Fourier e, em seguida, normalizadas para garantir uma escala uniforme. Com base na similaridade de cosseno, foi gerada uma matriz de similaridade que facilitou a comparação quantitativa entre os sinais. Os resultados foram apresentados de forma gráfica por meio de mapas de calor (heatmaps).

4.3.3 Análise com a Transformada de Wavelet

Na terceira abordagem os sinais foram analisados por meio da Transformada Wavelet Discreta (DWT), que decompõe os dados em diferentes escalas, capturando informações tanto no domínio do tempo quanto no domínio da frequência.

A transformada foi realizada utilizando a wavelet Daubechies de ordem 1 (`db1`), uma wavelet ortogonal amplamente utilizada por sua simplicidade computacional e sua eficiência em representar transições abruptas nos sinais. O número de níveis de decomposição foi determinado automaticamente com base no tamanho de cada sinal, sendo calculado como o maior número inteiro possível que representa o logaritmo de base 2 do tamanho do sinal. Dessa forma, sinais maiores foram decompostos em mais níveis, enquanto sinais menores resultaram em menos níveis de decomposição, garantindo que a análise fosse adaptada à resolução de cada dado.

Para a aplicação da transformada de Wavelet, utilizou-se o módulo pywt da linguagem Python. Os sinais foram decompostos em coeficientes de aproximação, que representam os componentes de baixa frequência (estrutura global do sinal), e coeficientes de detalhe, que capturam as variações de alta frequência (transições rápidas e detalhes finos). As seguintes métricas foram extraídas de cada nível de decomposição:

- **Energia**: Soma dos quadrados dos coeficientes, representando a intensidade associada a cada nível.
- **Entropia**: Mede a dispersão dos coeficientes, indicando a uniformidade ou complexidade em cada escala.
- **Média**: Valor médio dos coeficientes no nível analisado.
- **Desvio Padrão**: Mede a variação ou dispersão dos coeficientes em relação à média.
- **Curtose**: Avalia a forma da distribuição dos coeficientes, indicando a concentração em torno da média.
- **Assimetria (*Skewness*)**: Mede o desvio ou simetria da distribuição dos coeficientes.

Os níveis de decomposição 3 a 8 foram selecionados para a análise por capturarem as informações mais relevantes do sinal. Níveis mais baixos, como 1 e 2, foram evitados, pois

contêm componentes de alta frequência que podem ser dominados por ruídos ou transições abruptas irrelevantes. Já os níveis acima de 8 foram desconsiderados, pois tendem a capturar apenas componentes de baixa frequência ou tendências globais do sinal, que têm menor relevância para a análise de similaridade. Dessa forma, os níveis intermediários permitem equilibrar os detalhes significativos e as características estruturais do sinal.

Para a análise de similaridade, as métricas extraídas dos níveis selecionados foram organizadas em vetores de características para cada sinal. Esses vetores foram normalizados para garantir que todas as métricas tivessem o mesmo peso no cálculo de similaridade. A matriz de similaridade foi construída utilizando o cosseno de similaridade, que mede o grau de alinhamento entre os vetores de características. Os resultados foram apresentados graficamente por meio de mapas de calor, permitindo uma análise visual clara das relações de similaridade entre os sinais analisados.

5 Resultados

Este capítulo apresenta os resultados da análise de similaridade entre os códigos nas três bases de dados do estudo. A avaliação foi realizada a partir dos casos previamente identificados de forma manual como plágio, da aplicação das ferramentas tradicionais MOSS e JPLAG e das três abordagens adotadas no estudo: análise no domínio do tempo, com Transformada de Fourier e Transformada de Wavelet.

5.1 Resultados obtidos na Base 1

Para o primeiro grupo de códigos, a análise manual identificou um caso de plágio entre os códigos 08 e 11. A base foi submetida às ferramentas JPLAG e MOSS, cujos resultados são apresentados nas Figuras 3 e 4 respectivamente.

Ambas as ferramentas identificaram um nível de mais de 50% de similaridade entre o par de códigos 08 e 11. O JPlag indicou uma similaridade de 63,6%, enquanto o MOSS apontou que 54% do código 08 é composto pelo código 11 e 72% do código 11 é derivado do código 8, com 109 linhas coincidentes entre eles. Além disso, o MOSS também detectou um elevado índice de similaridade para o par (código02, código06), em que 43% do código 02 é composto pelo código 06, com 170 linhas correspondentes.

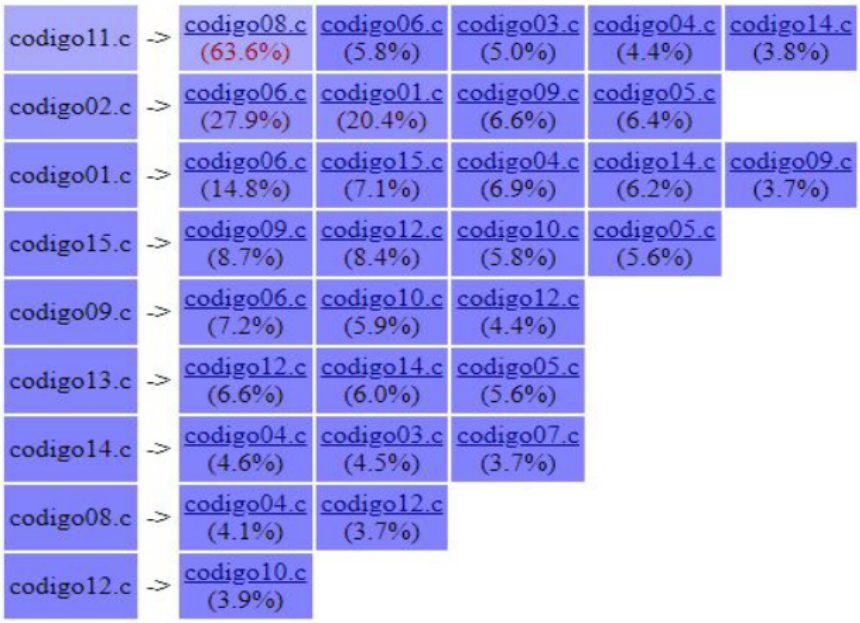

Figura 3 – Resultados da ferramenta JPLAG para a Base 1.

Fonte: O autor.

File 1	File 2	Lines Matched
codigo02.c (43%)	codigo06.c (23%)	170
codigo08.c (54%)	codigo11.c (72%)	109
codigo01.c (20%)	codigo06.c (11%)	70
codigo01.c (20%)	codigo02.c (20%)	84
codigo12.c (7%)	codigo15.c (5%)	52
codigo13.c (5%)	codigo14.c (9%)	54
codigo05.c (6%)	codigo06.c (4%)	29
codigo13.c (4%)	codigo15.c (3%)	31
codigo12.c (4%)	codigo13.c (3%)	15
codigo02.c (4%)	codigo05.c (2%)	16
codigo12.c (2%)	codigo14.c (4%)	19
codigo09.c (1%)	codigo12.c (2%)	20
codigo01.c (3%)	codigo13.c (2%)	12
codigo02.c (3%)	codigo09.c (1%)	23
codigo01.c (3%)	codigo05.c (2%)	6
codigo09.c (1%)	codigo15.c (1%)	25
codigo09.c (1%)	codigo10.c (2%)	23

Figura 4 – Resultados da ferramenta MOSS para a Base 1.

Fonte: O autor.

Para a análise do domínio do tempo, a métrica distância DTW identificou o par de códigos 08 e 11 como os mais similares, correspondendo com o caso de plágio da base. A Métrica de Correlação de Pearson com o ajuste temporal classificou esse como o quinto mais similar. Outras métricas não identificaram alta similaridade para o caso conhecido de plágio. A Figura 5 apresenta os cinco pares mais similares para cada métrica aplicada. Para as medidas de distância, os menores valores representam maior similaridade, enquanto, nas medidas de correlação, os maiores coeficientes indicam maior relação entre os códigos.

A análise no domínio da frequência, realizada por meio da Transformada de Fourier, também indicou o par de códigos 08 e 11 como os mais similares. A figura 6 apresenta a matriz de similaridade entre todos os pares de códigos, a partir das características extraídas do domínio da frequência, onde os valores mais altos indicam maior correspondência entre os sinais.

Por fim, na análise por Transformada de Wavelet, o índice de similaridade para o par (código08, código11) foi 0,68, com pares como (código03, código02) e (código06, código12) apresentando valores superiores. A Figura 7 apresenta a matriz de similaridade entre todos os códigos, baseada nas métricas extraídas dos coeficientes da decomposição realizada.

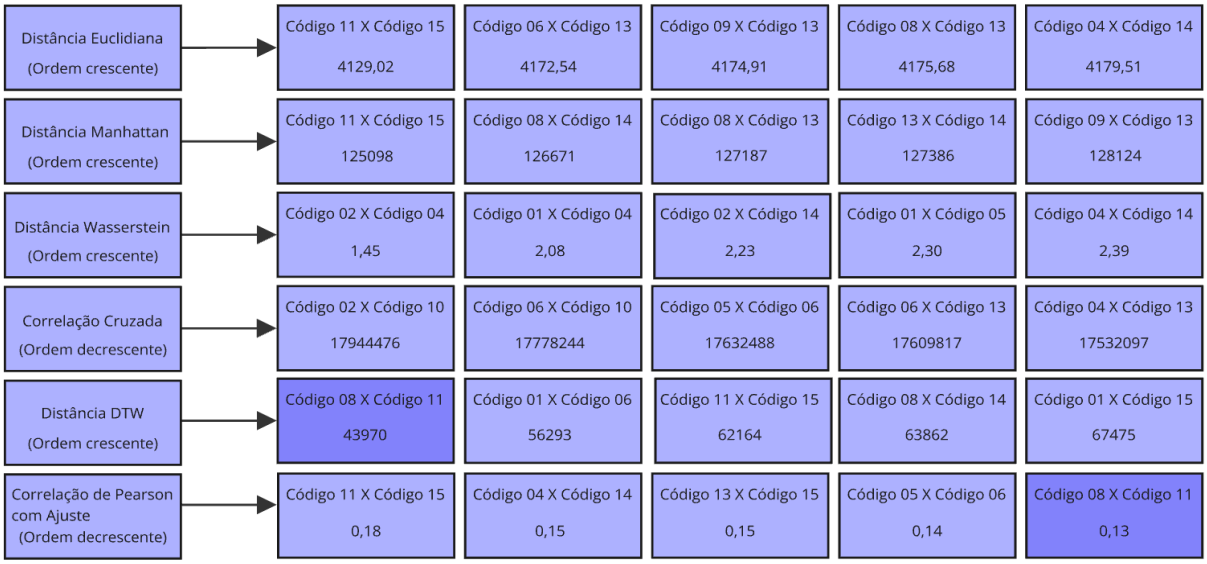

Figura 5 – Top 5 pares de códigos mais similares na Base 1 segundo as métricas do domínio do tempo.

Fonte: O autor

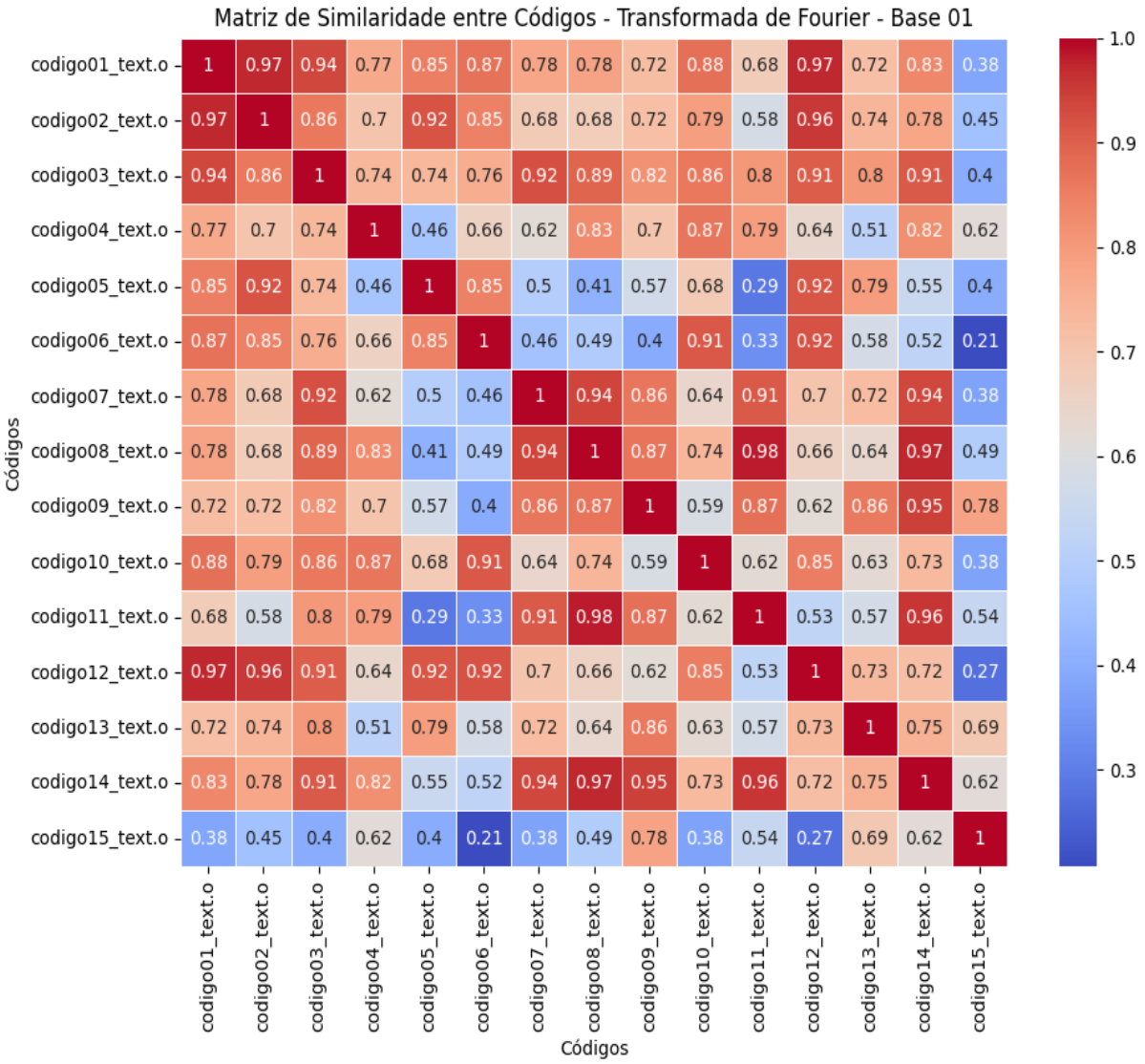

Figura 6 – Matriz de Similaridade entre os Códigos - Transformada de Fourier - Base 1.

Fonte: O próprio autor.

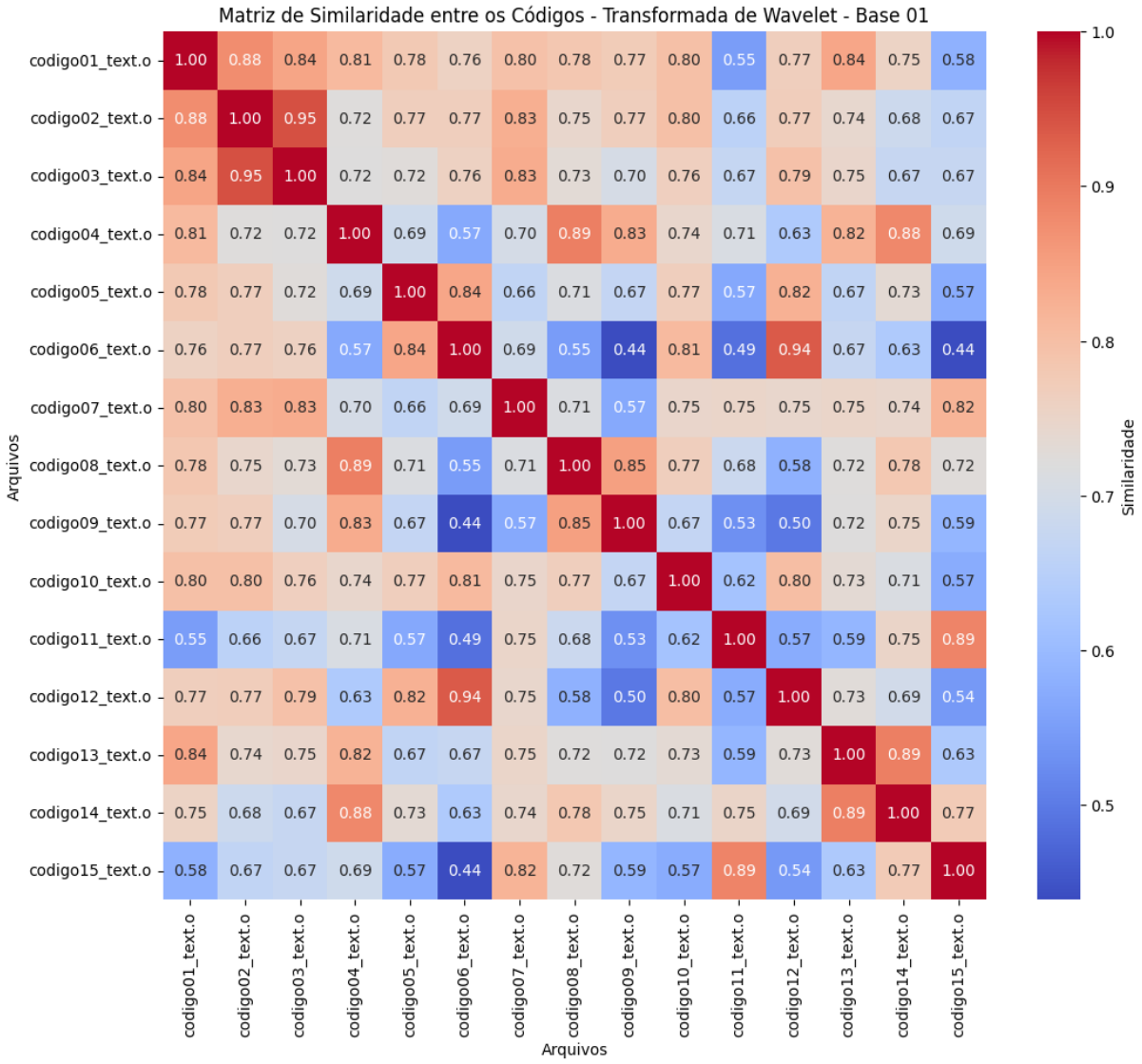

Figura 7 – Matriz de Similaridade entre os Códigos - Transformada de Wavelet - Base 1.

Fonte: O próprio autor.

Os resultados obtidos mostraram que a análise no domínio do tempo, com as métricas distância DTW e correlação de Pearson, e a análise espectral com a transformada de Fourier alcançaram resultados similares aos das ferramentas MOSS e JPLAG, também alinhadas com o caso identificado de plágio.

5.2 Resultados obtidos na Base 2

Para o segundo grupo de códigos, a análise manual identificou três casos de plágio entre os pares (código05, código10), (código07, código09) e (código09, código11). Os resultados obtidos com MOSS e JPlag são apresentados nas Figuras 8 e 9 respectivamente.

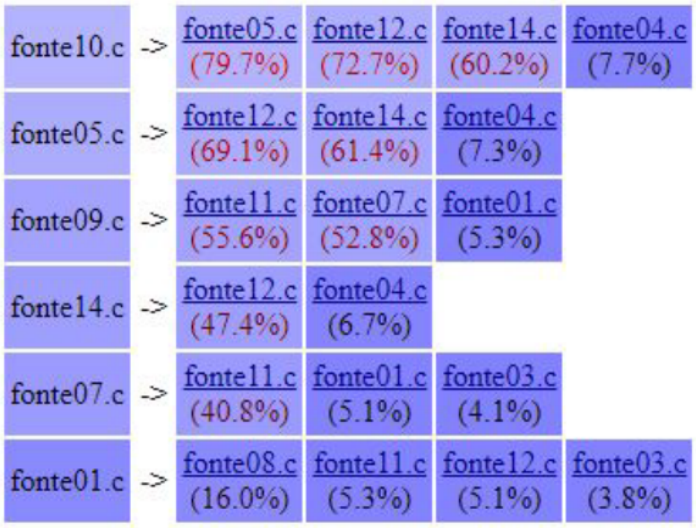

Figura 8 – Resultados da ferramenta JPLAG para a Base 2.

Fonte: O autor.

File 1	File 2	Lines Matched
fonte09.c (60%)	fonte11.c (56%)	126
fonte10.c (45%)	fonte14.c (38%)	75
fonte07.c (47%)	fonte09.c (55%)	96
fonte05.c (37%)	fonte10.c (40%)	46
fonte10.c (32%)	fonte12.c (25%)	40
fonte07.c (31%)	fonte11.c (34%)	63
fonte05.c (23%)	fonte14.c (21%)	42
fonte05.c (23%)	fonte12.c (19%)	35
fonte12.c (10%)	fonte14.c (11%)	29
fonte02.c (4%)	fonte08.c (5%)	12
fonte01.c (4%)	fonte09.c (7%)	8
fonte01.c (4%)	fonte07.c (6%)	8
fonte03.c (3%)	fonte07.c (6%)	8
fonte04.c (4%)	fonte05.c (4%)	5

Figura 9 – Resultados da ferraments MOSS para a Base 2.

Fonte: O autor.

O JPlag identificou um alto nível de similaridade para os três pares previamente identificados, enquanto o MOSS encontrou valores acima de 50% apenas para os pares (código09, código11) e (código07, código09). Outros pares, como (código10, código12), (código10, código14), (código05, código12) e (código05, código14) também apresentaram elevada similaridade no JPlag.

Na análise no domínio do tempo, a distância DTW identificou (código05, código10) como o mais similar, seguido de (código09, código11) como o quinto mais similar. A distância Euclidiana também posicionou (código09, código11) e (código07, código09) entre os menores valores. A Correlação de Pearson classificou os pares (código05, código10), (código09, código11) e (código07, código09) entre os cinco mais similares, corroborando os resultados obtidos pelas ferramentas JPLAG e MOSS. A Figura 10 apresenta os cinco pares mais similares identificados para cada métrica aplicada.

A análise por transformada de Fourier atribuiu índices de similaridade de 0,96 para (código05, código10), 0,71 para (código09, código11) e 0,83 para (código07, código09). Outros pares apresentaram índices superiores, incluindo (código02, código01), (código03, código02), (código10, código06), (código05, código14) e (código13, código02).A figura 11 apresenta a matriz de similaridade entre todos os pares de códigos, considerando as métricas extraídas do domínio da frequência.

Por fim, a Figura 12 exibe a matriz de similaridade gerada a partir das métricas extraídas dos coeficientes de decomposição wavelet. Os pares (código02, código01), (código03, código02), (código10, código06), (código05, código14) e (código13, código02) apresentaram valores mais elevados em comparação com os pares (código05, código10), (código07, código09) e (código09, código11).

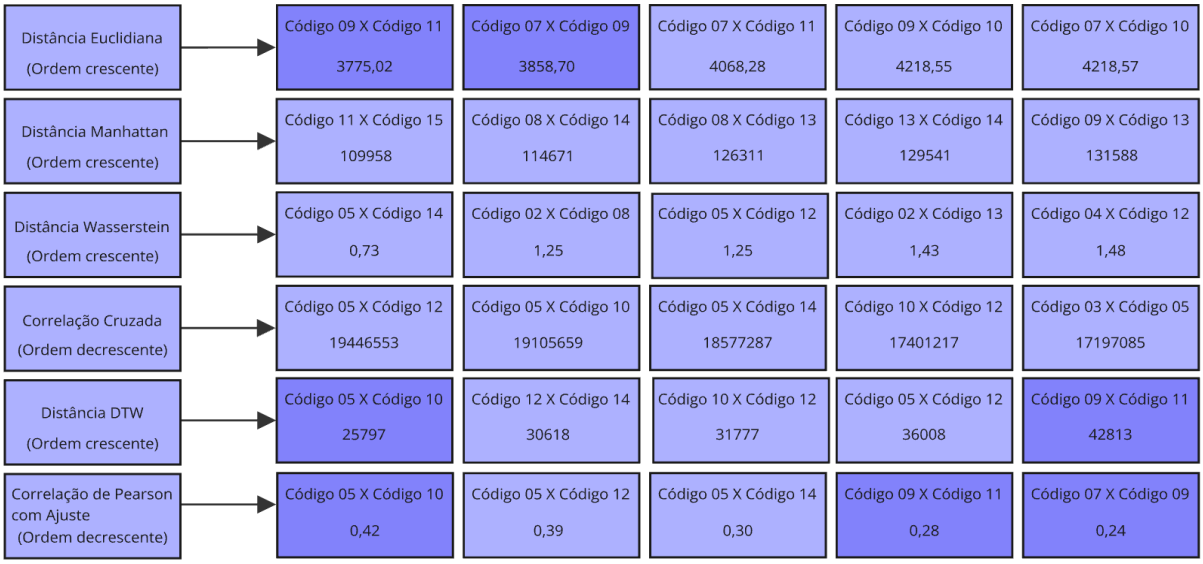

Figura 10 – Top 5 pares de códigos mais similares na Base 2 segundo as métricas do domínio do tempo.

Fonte: O autor

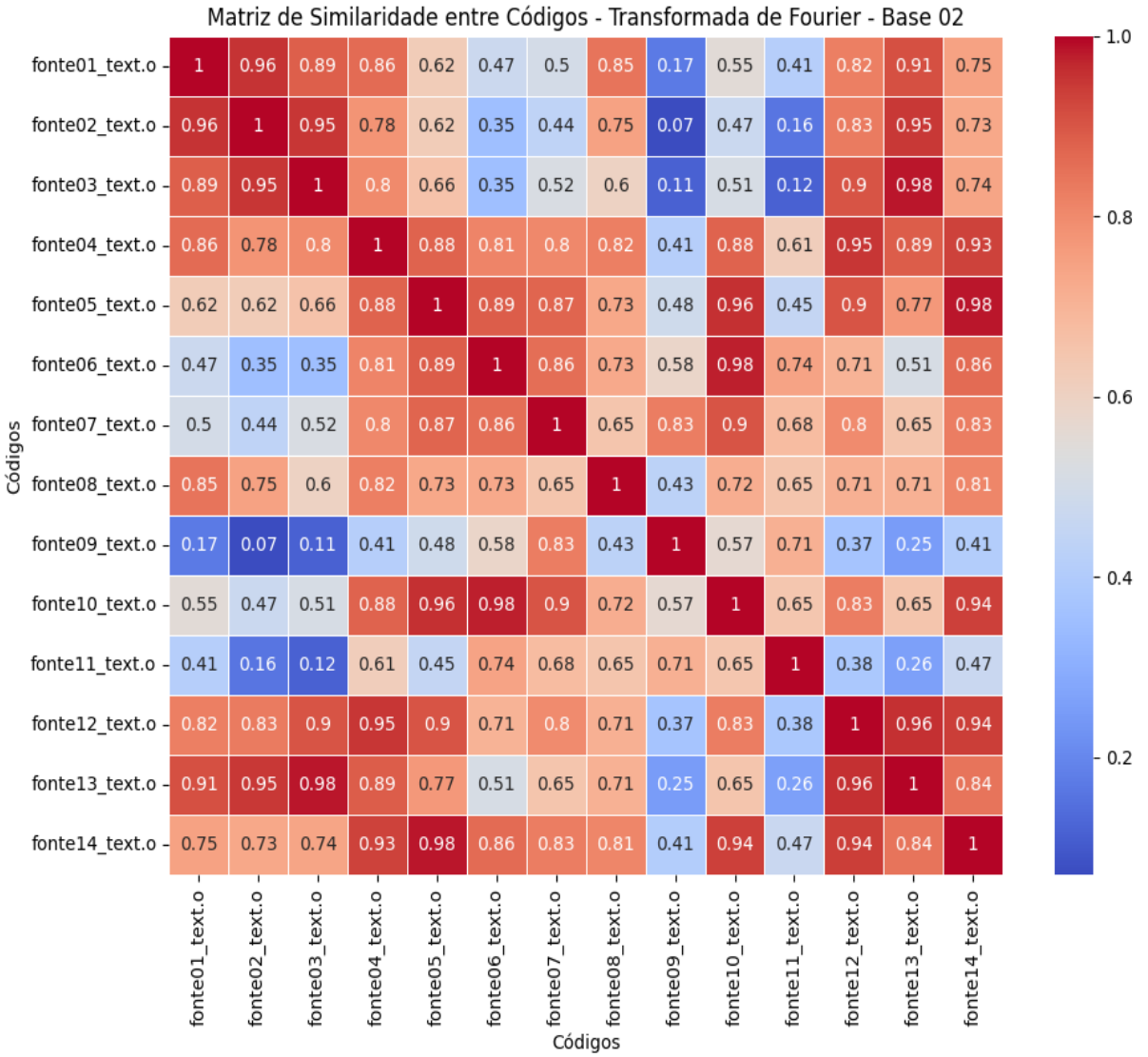

Figura 11 – Matriz de Similaridade entre os Códigos - Transformada de Fourier - Base 2.

Fonte: O próprio autor.

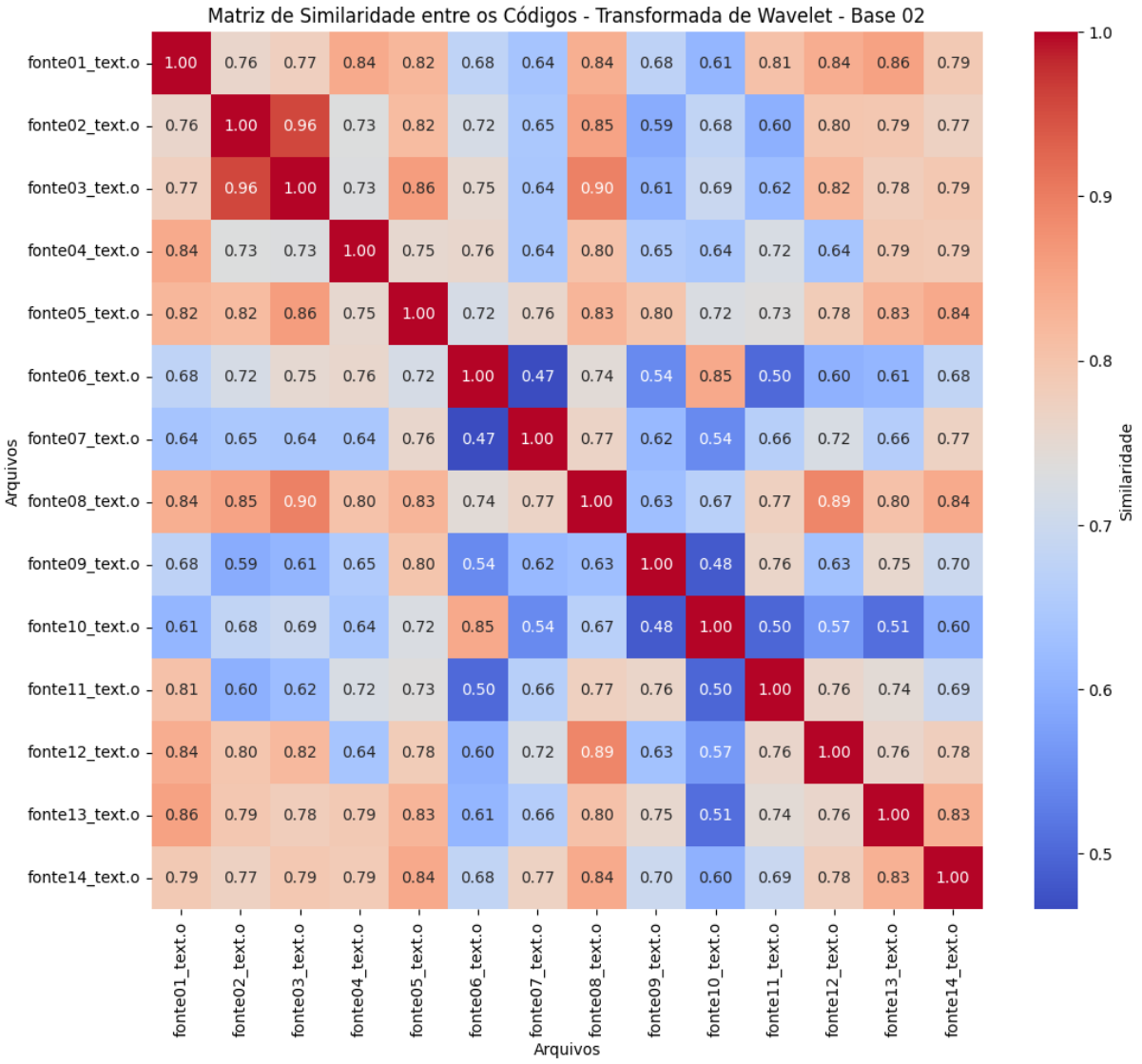

Figura 12 – Matriz de Similaridade entre os Códigos - Transformada de Wavelet - Base 2.

Fonte: O próprio autor.

Os resultados obtidos mostraram que a análise no domínio do tempo, com as métricas distância DTW, distância euclidiana e correlação de Pearson com ajuste temporal, apresentaram resultados similares aos das ferramentas MOSS e JPlag, também alinhadas com os casos identificados de plágio. Entretanto, as abordagens baseadas nas Transformadas de Fourier e Wavelet não posicionaram esses pares entre os mais similares, registrando índices mais elevados para outros pares de códigos.

5.3 Resultados obtidos na Base 3

Para o terceiro grupo de códigos, a análise manual identificou dois casos de plágio entre os pares de códigos (fonte02, fonte03) e (fonte09, fonte12). A base foi submetida às ferramentas MOSS e JPLAG, cujos resultados são apresentados nas figuras 13 e 14.

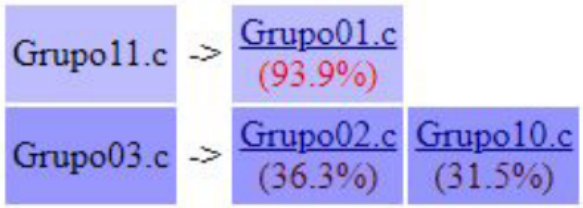

Figura 13 – Resultado da ferramenta JPLAG para a Base 3.

Fonte: O autor.

File 1	File 2	Lines Matched
Grupo01.c (69%)	Grupo11.c (72%)	118
Grupo02.c (34%)	Grupo03.c (27%)	48
Grupo07.c (17%)	Grupo08.c (16%)	14
Grupo10.c (17%)	Grupo11.c (16%)	8
Grupo01.c (15%)	Grupo10.c (17%)	8
Grupo05.c (16%)	Grupo12.c (18%)	6
Grupo08.c (7%)	Grupo12.c (9%)	5
Grupo07.c (7%)	Grupo12.c (9%)	5
Grupo07.c (7%)	Grupo11.c (7%)	13
Grupo01.c (7%)	Grupo07.c (7%)	13

Figura 14 – Resultado da ferramenta MOSS para a Base 3.

Fonte: O autor.

O JPlag e o MOSS identificaram o par (fonte11, fonte01) como o mais similar. O par (fonte02, fonte03) apresentou um nível de similaridade mais baixo em comparação, com 36,3% no JPlag e 34% e 27% no MOSS.

Na análise no domínio do tempo, as métricas Distância Euclidiana e Distância Manhattan identificaram o par (fonte02, fonte03) como o mais similar. A Correlação de Pearson com ajuste temporal classificou esse par como o segundo mais similar, enquanto a Correlação Cruzada o posicionou como o terceiro mais similar. As métricas Distância DTW e Correlação de Pearson com ajuste temporal identificaram o par (fonte01, fonte11) como o mais similar, em concordância com os resultados obtidos pelas ferramentas MOSS e JPlag. A Figura 15 apresenta os cinco pares mais similares identificados para cada métrica aplicada.

Na análise espectral da transformada de Fourier, os pares (fonte05, fonte08) e (fonte06, fonte08) apresentaram índices mais elevados em comparação com o caso de plágio (fonte02, fonte03). A Figura 16 apresenta a matriz de similaridade entre os pares de códigos, considerando as métricas extraídas do domínio da frequência.

Na análise por Transformada de Wavelet, a similaridade entre o par (fonte02, fonte03)

não esteve entre os maiores índices observados. A Figura 17 apresenta a matriz de similaridade gerada a partir das métricas extraídas dos coeficientes wavelet, no qual outros pares foram posicionados com valores mais altos.

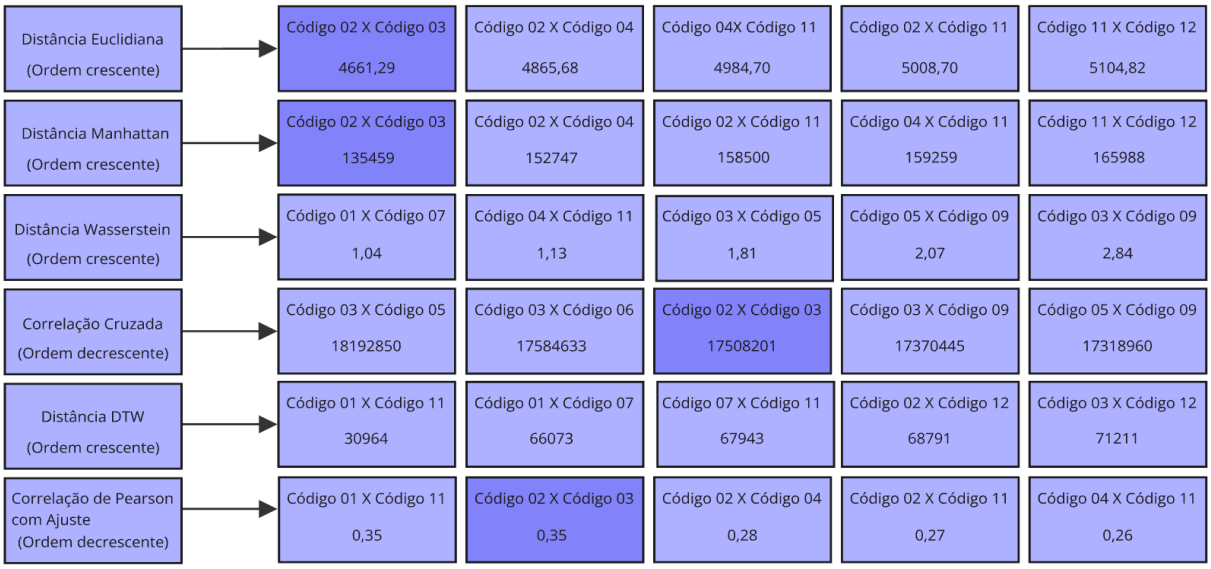

Figura 15 – Top 5 pares de códigos mais similares na Base 3 segundo as métricas do domínio do tempo.

Fonte: O autor

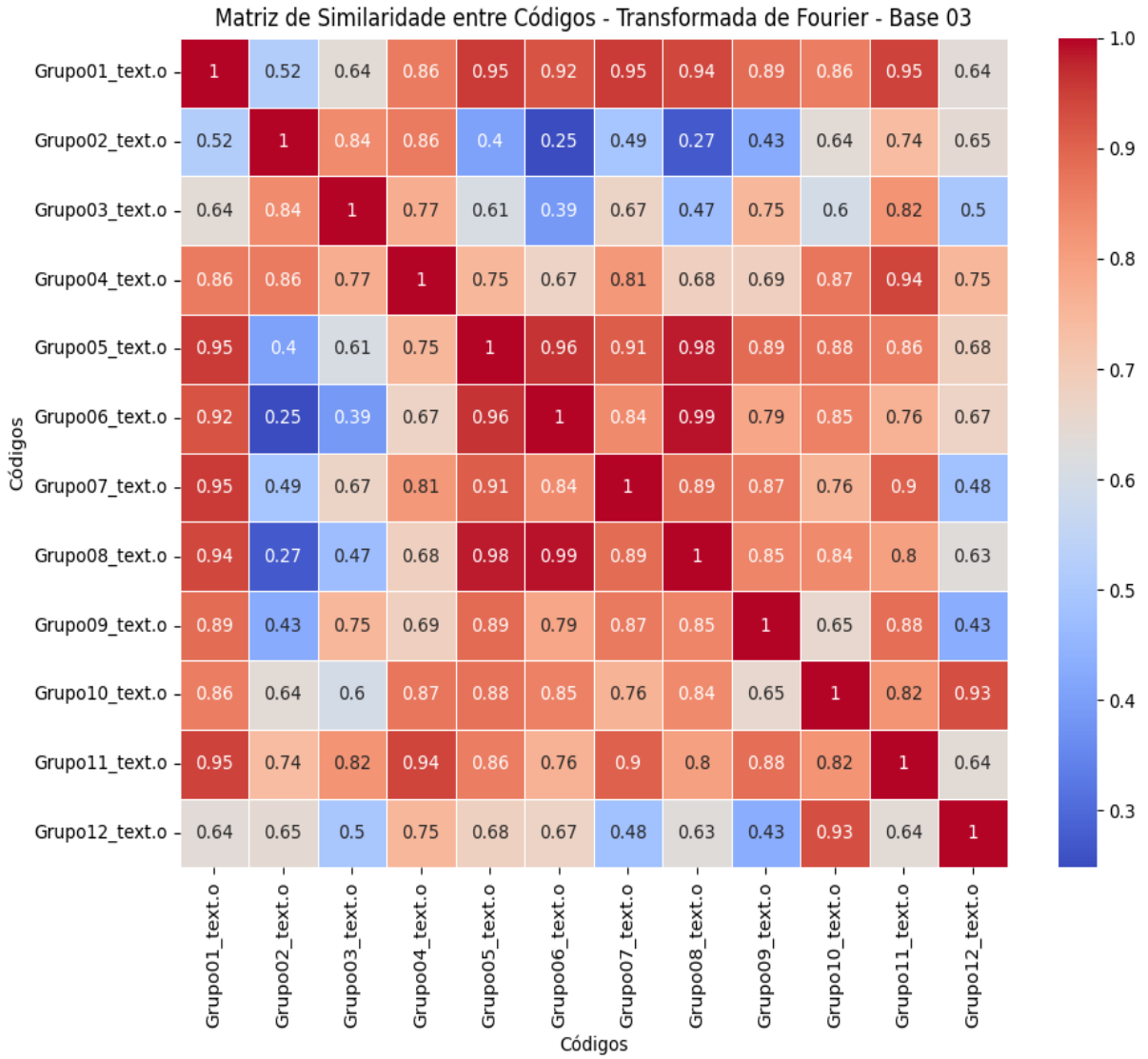

Figura 16 – Matriz de Similaridade entre os Códigos - Transformada de Fourier - Base 3.

Fonte: O autor.

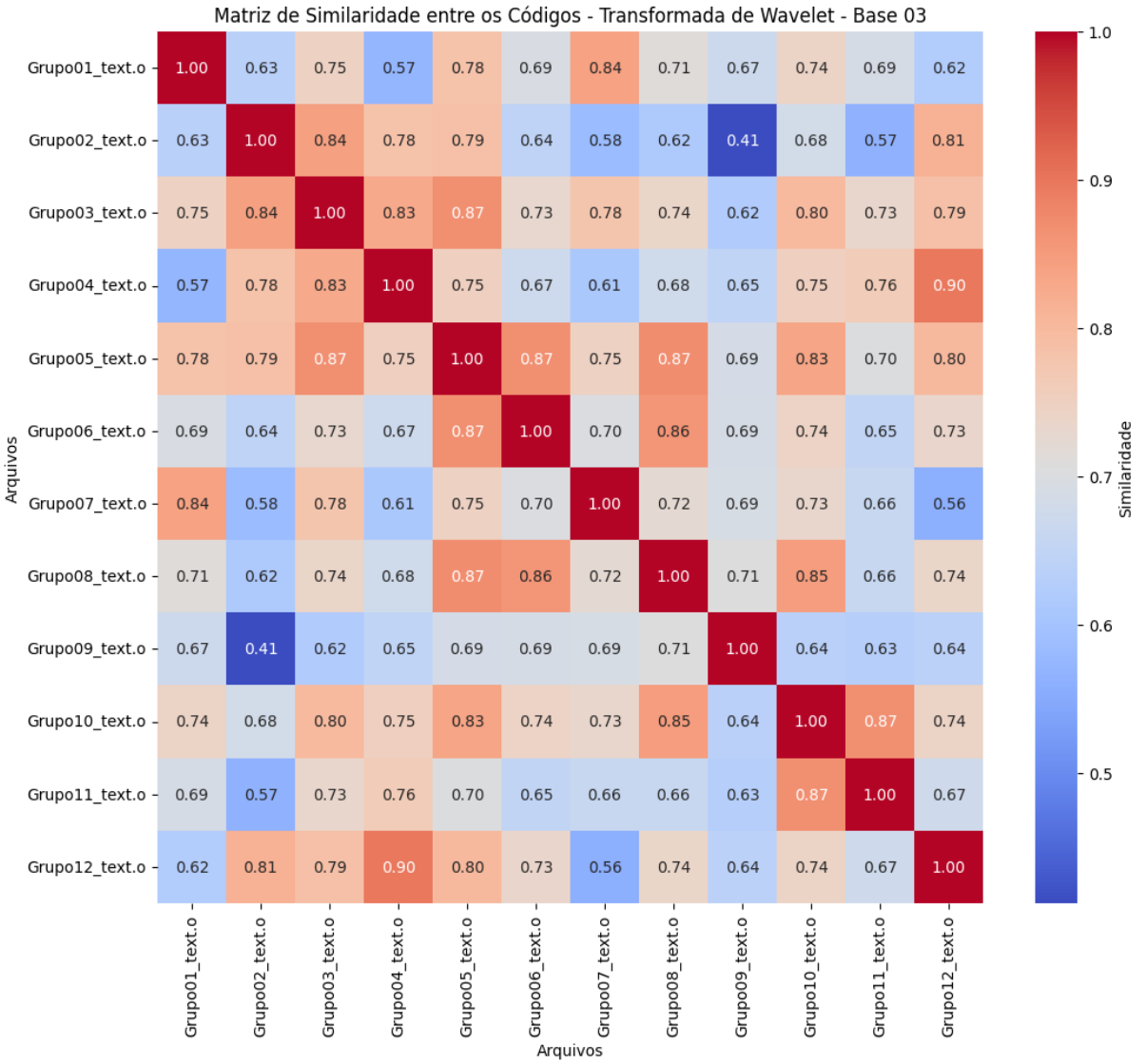

Figura 17 – Matriz de Similaridade entre os Códigos - Transformada de Wavelet - Base 3.

Fonte: O autor.

Os resultados da Base 3 mostram que as abordagens no domínio do tempo foram mais consistentes com os casos identificados manualmente. Por outro lado, as abordagens baseadas na Transformada de Fourier e na Transformada de Wavelet não posicionaram o par (fonte02, fonte03) entre os mais similares, atribuindo índices mais elevados a outros pares de códigos.

6 Conclusão

Neste trabalho foram avaliadas abordagens para a detecção de similaridade entre códigos-fonte a partir da análise de arquivos objeto utilizando técnicas de processamento de sinais. As análises foram realizadas nos domínios do tempo, Transformada de Fourier e Transformada de Wavelet, e comparadas com os resultados obtidos pelas ferramentas MOSS e JPLAG.

Os resultados indicam que as métricas baseadas no domínio do tempo, em especial a Distância DTW e a Correlação de Pearson com ajuste temporal, foram as mais eficazes na identificação de plágio entre códigos. Essas métricas identificaram corretamente casos de plágio previamente conhecidos e apresentaram desempenho próximo ao das ferramentas tradicionais utilizadas na área.

As abordagens baseadas na Transformada de Fourier e na Transformada de Wavelet não apresentaram a mesma efetividade das métricas no domínio do tempo. A matriz de similaridade obtida por essas técnicas não evidenciou padrões claros de correspondência entre os códigos analisados, sugerindo limitações da abordagem adotada neste estudo para a aplicação direta dessas transformadas na detecção de similaridade.

A análise no domínio do tempo, com destaque para a DTW e a Correlação de Pearson com ajuste temporal, mostrou-se a alternativa mais adequada para a detecção de similaridade entre códigos a partir da representação binária. Esse resultado sugere que técnicas de processamento de sinais podem ser empregadas como complemento ou alternativa aos métodos tradicionais baseados na estrutura sintática do código-fonte.

Para trabalhos futuros, seria promissor explorar um conjunto ampliado de métricas baseadas em séries temporais e investigar a combinação ponderada de medidas como DTW, Correlação de Pearson e Correlação Cruzada para a definição de um coeficiente unificado de similaridade. Além disso, a avaliação do impacto de diferentes estratégias de pré-processamento dos arquivos objeto e dos sinais obtidos pode contribuir para melhorar a representatividade dos sinais analisados e influenciar diretamente a precisão dos resultados obtidos.

Referências

AHTIAINEN, A.; SURAKKA, S.; RAHIKAINEN, M. Plaggie: Gnu-licensed source code plagiarism detection engine for java exercises. In: *Proceedings of the 6th Baltic Sea conference on Computing education research: Koli Calling 2006*. [S.l.: s.n.], 2006. p. 141–142. Citado na página 29.

ALFRED, V. A.; MONICA, S. L.; JEFFREY, D. U. *Compilers principles, techniques & tools*. [S.l.]: pearson Education, 2007. Citado na página 21.

BAZALUK, B. et al. Source code plagiarism in computer science courses: Facts and impressions. In: SBC. *Anais do XXXI Workshop sobre Educação em Computação*. [S.l.], 2023. p. 111–121. Citado na página 17.

BOULOS, P.; CAMARGO, I. de. Geometria analítica. *CEP*, v. 4533, n. 004, 1987. Citado na página 22.

BRACEWELL, R.; KAHN, P. B. The fourier transform and its applications. *American Journal of Physics*, American Association of Physics Teachers, v. 34, n. 8, p. 712–712, 1966. Citado na página 25.

BURROWS, S.; TAHAGHOGHI, S. M.; ZOBEL, J. Efficient plagiarism detection for large code repositories. *Software: Practice and Experience*, Wiley Online Library, v. 37, n. 2, p. 151–175, 2007. Citado na página 17.

CHUDA, D. et al. The issue of (software) plagiarism: A student view. *IEEE Transactions on Education*, IEEE, v. 55, n. 1, p. 22–28, 2011. Citado na página 17.

CHUN-LIN, L. A tutorial of the wavelet transform. *NTUEE, Taiwan*, v. 21, n. 22, p. 2, 2010. Citado na página 25.

COOLEY, J. W.; LEWIS, P. A.; WELCH, P. D. The fast fourier transform and its applications. *IEEE Transactions on Education*, IEEE, v. 12, n. 1, p. 27–34, 1969. Citado na página 25.

COOPER, K. D.; TORCZON, L. *Engineering a compiler*. [S.l.]: Morgan Kaufmann, 2022. Citado na página 21.

COSMA, G.; JOY, M. Towards a definition of source-code plagiarism. *IEEE Transactions on Education*, IEEE, v. 51, n. 2, p. 195–200, 2008. Citado na página 17.

COSMA, G.; JOY, M. An approach to source-code plagiarism detection and investigation using latent semantic analysis. *IEEE transactions on computers*, IEEE, v. 61, n. 3, p. 379–394, 2011. Citado na página 17.

DEVORE-MCDONALD, B.; BERGER, E. D. Mossad: Defeating software plagiarism detection. *Proceedings of the ACM on Programming Languages*, ACM New York, NY, USA, v. 4, n. OOPSLA, p. 1–28, 2020. Citado na página 27.

JI, J.-H.; WOO, G.; CHO, H.-G. A plagiarism detection technique for java program using bytecode analysis. In: IEEE. *2008 third international conference on convergence and hybrid information technology*. [S.l.], 2008. v. 1, p. 1092–1098. Citado 2 vezes nas páginas 18 e 30.

JURIČIĆ, V. Detecting source code similarity using low-level languages. In: IEEE. *Proceedings of the ITI 2011, 33rd International Conference on Information Technology Interfaces*. [S.l.], 2011. p. 597–602. Citado 2 vezes nas páginas 18 e 30.

JURIČIĆ, V.; JURIĆ, T.; TKALEC, M. Performance evaluation of plagiarism detection method based on the intermediate language. Department of Information Sciences, Faculty of Humanities and Social . . . , 2011. Citado na página 30.

KARNALIM, O. Detecting source code plagiarism on introductory programming course assignments using a bytecode approach. In: IEEE. *2016 International Conference on Information & Communication Technology and Systems (ICTS)*. [S.l.], 2016. p. 63–68. Citado 2 vezes nas páginas 18 e 30.

KARNALIM, O. Source code plagiarism detection with low-level structural representation and information retrieval. *International Journal of Computers and Applications*, Taylor & Francis, v. 43, n. 6, p. 566–576, 2021. Citado na página 18.

KARUS, S.; KILGI, K. Code clone detection using wavelets. In: IEEE. *2015 IEEE 9th International Workshop on Software Clones (IWSC)*. [S.l.], 2015. p. 8–14. Citado na página 25.

KRAUSE, E. F. Taxicab geometry. *The Mathematics Teacher*, National Council of Teachers of Mathematics, v. 66, n. 8, p. 695–706, 1973. Citado na página 23.

KUSTANTO, C.; LIEM, I. Automatic source code plagiarism detection. In: IEEE. *2009 10th ACIS International conference on software engineering, artificial intelligences, networking and parallel/distributed computing*. [S.l.], 2009. p. 481–486. Citado na página 17.

LUO, L. et al. Semantics-based obfuscation-resilient binary code similarity comparison with applications to software plagiarism detection. In: *Proceedings of the 22nd ACM SIGSOFT international symposium on foundations of software engineering*. [S.l.: s.n.], 2014. p. 389–400. Citado na página 17.

MATUSCHEK, M.; SCHLÜTER, T.; CONRAD, S. Measuring text similarity with dynamic time warping. In: *Proceedings of the 2008 international symposium on Database engineering & applications*. [S.l.: s.n.], 2008. p. 263–267. Citado na página 23.

MOZGOVOY, M.; KAKKONEN, T.; COSMA, G. Automatic student plagiarism detection: future perspectives. *Journal of Educational Computing Research*, SAGE Publications Sage CA: Los Angeles, CA, v. 43, n. 4, p. 511–531, 2010. Citado na página 17.

PRECHELT, L. et al. Finding plagiarisms among a set of programs with jplag. *J. Univers. Comput. Sci.*, Citeseer, v. 8, n. 11, p. 1016, 2002. Citado 2 vezes nas páginas 17 e 28.

PRESS, W. H. *Numerical recipes 3rd edition: The art of scientific computing*. [S.l.]: Cambridge university press, 2007. Citado na página 25.

RABBANI, F. S.; KARNALIM, O. Detecting source code plagiarism on. net programming languages using low-level representation and adaptive local alignment. *Journal of Information and Organizational Sciences*, Sveučilište u Zagrebu Fakultet organizacije i informatike, v. 41, n. 1, p. 105–123, 2017. Citado na página 18.

RAMDAS, A.; TRILLOS, N. G.; CUTURI, M. On wasserstein two-sample testing and related families of nonparametric tests. *Entropy*, MDPI, v. 19, n. 2, p. 47, 2017. Citado na página 24.

RUBNER, Y.; TOMASI, C.; GUIBAS, L. J. A metric for distributions with applications to image databases. In: IEEE. *Sixth international conference on computer vision (IEEE Cat. No. 98CH36271)*. [S.l.], 1998. p. 59–66. Citado na página 24.

SCHLEIMER, S.; WILKERSON, D. S.; AIKEN, A. Winnowing: local algorithms for document fingerprinting. In: *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. [S.l.: s.n.], 2003. p. 76–85. Citado 2 vezes nas páginas 17 e 27.

SRAKA, D.; KAUCIC, B. Source code plagiarism. In: IEEE. *Proceedings of the ITI 2009 31st international conference on information technology interfaces*. [S.l.], 2009. p. 461–466. Citado na página 17.

THOMPSON, V. U.; PANCHEV, C.; OAKES, M. Performance evaluation of similarity measures on similar and dissimilar text retrieval. In: IEEE. *2015 7th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K)*. [S.l.], 2015. v. 1, p. 577–584. Citado na página 22.

ĐURIĆ, Z.; GAŠEVIĆ, D. A source code similarity system for plagiarism detection. *The Computer Journal*, Oxford University Press, v. 56, n. 1, p. 70–86, 2013. Citado na página 17.

WANG, J.; DONG, Y. Measurement of text similarity: a survey. *Information*, MDPI, v. 11, n. 9, p. 421, 2020. Citado na página 24.

YADAV, M.; ALAM, M. A. Dynamic time warping (dtw) algorithm in speech: a review. *International Journal of Research in Electronics and Computer Engineering*, v. 6, n. 1, p. 524–528, 2018. Citado na página 23.