UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DIRETORIA DE PESQUISA E PÓS-GRADUAÇÃO
MESTRADO EM CIÊNCIA DA COMPUTAÇÃO

KAIO PABLO GOMES

BIOPLAG: ABORDAGEM DE DETECÇÃO DE PLÁGIO EM CÓDIGO-FONTE UTILIZANDO BIOINFORMÁTICA

DISSERTAÇÃO

PONTA GROSSA
2020

KAIO PABLO GOMES

BIOPLAG: ABORDAGEM DE DETECÇÃO DE PLÁGIO EM CÓDIGO-FONTE UTILIZANDO BIOINFORMÁTICA

Dissertação apresentada como requisito parcial à obtenção do título de Mestre em Ciência da Computação do programa de Pós-Graduação em Ciência da Computação da Universidade Tecnológica Federal do Paraná – Campus Ponta Grossa.

Área de Concentração: Sistemas e Métodos de Computação.

Orientadora: Profa. Dra. Simone Nasser Matos

PONTA GROSSA

2020

Ficha catalográfica elaborada pelo Departamento de Biblioteca
da Universidade Tecnológica Federal do Paraná, Campus Ponta Grossa
n.48/20

G633 Gomes, Kaio Pablo

BIOPLAG: abordagem de detecção de plágio em código-fonte utilizando bioinformática. / Kaio Pablo Gomes, 2020.
117 f. : il. ; 30 cm.

Orientadora: Prof. Dra. Simone Nasser Matos

Dissertação (Mestrado em Ciência da Computação) - Programa de Pós-Graduação em Ciência da Computação. Universidade Tecnológica Federal do Paraná, Ponta Grossa, 2020.

1. Plágio. 2. Programação (Computadores). 3. Bioinformática. I. Matos, Simone Nasser. II. Universidade Tecnológica Federal do Paraná. III. Título.

CDD 004

Elson Heraldo Ribeiro Junior. CRB-9/1413. 06/07/2020.

	Ministério da Educação **UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ** **Câmpus Ponta Grossa** Diretoria de Pesquisa e Pós-Graduação Programa de Pós-Graduação em Ciência da Computação	

FOLHA DE APROVAÇÃO

Título de Dissertação **n. 19/2020**

BIOPLAG: ABORDAGEM DE DETECÇÃO DE PLÁGIO EM CÓDIGO-FONTE UTILIZANDO BIOINFORMÁTICA

Por

Kaio Pablo Gomes

Esta dissertação foi apresentada às **16 horas** de **8 de junho de 2020**, à distância, por webconferência, como requisito parcial para a obtenção do título de MESTRE EM CIÊNCIA DA COMPUTAÇÃO, Programa de Pós-Graduação em Ciência da Computação. O candidato foi arguido pela Banca Examinadora, composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho APROVADO.

Prof. Dr. Elias Silva de Oliveira (UFES)

Prof. Dr. Alexandre Rossi Paschoal (UTFPR)

Profa. Dra. Simone Nasser Matos (UTFPR)
Orientador e presidente da banca

Visto da Coordenadora:

Prof. Dr. André Koscianski
Coordenadora do PPGCC
UTFPR – Câmpus Ponta Grossa

- A Folha de Aprovação assinada encontra-se na Coordenação do Programa -

AGRADECIMENTOS

À minha família, namorada e amigos pelo apoio incondicional a todos os momentos durante a realização dos meus estudos. Houve o incentivo e suporte em diferentes aspectos que me ajudaram na conclusão deste mestrado. Agradeço a Deus por tudo.

À orientadora Profa. Dra. Simone Nasser Matos, por me apoiar e auxiliar durante toda a jornada acadêmica que percorri. À Profa. Dra. Kate Cooper, ao Prof. Dr. Dhundy Bastola e ao pesquisador Dr. Jay Pedersen, por possibilitarem a minha participação em uma pesquisa que contribuiu para a origem do tema desta dissertação durante um intercâmbio acadêmico na graduação.

Aos membros da banca de defesa: Prof. Dr. Elias Silva de Oliveira e Prof. Dr. Alexandre Rossi Paschoal, que contribuíram com suas análises avaliativas. Ao Vinicius Schultz Garcia da Luz e todos os outros participantes voluntários dos experimentos realizados nesta dissertação.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Código de Financiamento 001.

RESUMO

GOMES, Kaio Pablo. **BIOPLAG:** abordagem de detecção de plágio em código-fonte utilizando bioinformática. 2020. 117 f. Dissertação (Mestrado em Ciência da Computação) - Universidade Tecnológica Federal do Paraná, Ponta Grossa, 2020.

A reutilização não autorizada de código-fonte caracteriza o plágio em programação, que pode afetar desde o desempenho de alunos em disciplinas de programação até a qualidade do desenvolvimento de software nas empresas. Por meio da realização de um estudo de mapeamento sistemático foram analisadas as abordagens de detecção automática de plágio em programação para identificar as técnicas utilizadas, os procedimentos de testes de avaliação e as linguagens de programação que possuem suporte. Constatou-se que as soluções não contemplam as diferentes técnicas utilizadas pelos plagiadores para alterar os códigos-fontes. Este trabalho cria uma abordagem, nomeada BIOPLAG, capaz de aprimorar a detecção automática de níveis de plágio em código-fonte. O funcionamento da abordagem criada é fundamentado em técnicas da Bioinformática e da Ciência da Computação: *tokens* de elementos de linguagem de programação, mapeamento de códigos-fontes em sequências biológicas sintéticas e alinhamento de sequências biológicas. A implementação da BIOPLAG foi avaliada por meio de sete cenários de testes contendo 336 códigos-fontes implementados em linguagem *C* utilizados em 168 testes diferentes, sendo considerado em cada cenário os parâmetros avaliativos de desempenho: precisão, revocação e medida *F*. Todos os exemplos de códigos-fontes plagiados foram produzidos a partir de três experimentos reais desenvolvidos com a participação de alunos de graduação, mestrado e programadores de uma empresa de desenvolvimento de software da região. Os resultados obtidos foram comparados com duas ferramentas consideradas de referência no estado da arte: MOSS e JPLAG. Como resultado, a BIOPLAG apresentou desempenho melhor em quatro e igual em três cenários de testes considerando os indicadores de precisão, revocação e medida *F*.

Palavras-chave: Plágio em programação. Detecção automática. Código-fonte. Bioinformática.

ABSTRACT

GOMES, Kaio Pablo. **BIOPLAG:** approach to detect plagiarism in source code by using bioinformatics. 2020. 117 p. Thesis (Master's Degree in Computer Science) – Universidade Tecnológica Federal do Paraná, Ponta Grossa, 2020.

The unauthorized reuse of source code characterizes plagiarism in programming, which can affect everything from the performance of students in programming courses to the quality of software development in companies. Through the realization of a systematic mapping study, the approaches of automatic detection of plagiarism in programming were analyzed to identify the used techniques, the evaluation test procedures, and the supported programming languages. It was found that the solutions do not include the different techniques used by plagiarists to change the source codes. This work created an approach, named BIOPLAG, capable of improving the automatic detection of plagiarism levels in source code. The functioning of the created approach is based on Bioinformatics and Computer Science techniques: tokens of programming language elements, mapping of source codes in synthetic biological sequences, and alignment of biological sequences. The implementation of BIOPLAG was evaluated through seven test scenarios containing 336 source codes implemented in C language used in 168 different tests, considering in each scenario the evaluative performance parameters: precision, recall and measure F. All examples of Plagiarized source codes were produced from three real experiments developed with the participation of students from undergraduate, graduate, and programmers from a software development company in the region. The results obtained were compared with two tools considered to be the reference in state of the art: MOSS and JPLAG. As a result, BIOPLAG performed better in four and equal in three test scenarios considering the indicators of precision, recall, and measure F.

Keywords: Plagiarism in Programming. Automatic Detection. Source Code. Bioinformatics.

LISTA DE FIGURAS

LISTA DE GRÁFICOS

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE ABREVIATURAS, SIGLAS E ACRÔNIMOS

3D	Três dimensões
A	Adenina
A	Alanina
Ala	Alanina
Arg	Arginina
ASCII	American Standard Code for Information Interchange
Asn	Asparagina
Asp	Asparato
BLAST	Basic Local Alignment Search Tool
C	Citosina
C	Cisteína
CPAN	Comprehensive Perl Archive Network
Cys	Cisteína
D	Asparato
DNA	Ácido Desoxirribonucleico
E	Glutamato
F	Fenilalanina
FASTA	Fast Adaptive Shrinkage Search Tool
FN	Falsos Negativos
FORTRAN	Formula Translator
FP	Falsos Positivos
FPDS	Fast Plagiarism Detection System
G	Guanina
G	Glicina
Gln	Glutamina
Glu	Glutamato
Gly	Glicina
GST	Greedy String Tiling
H	Histidina
HCL	HashiCorp Configuration Language
His	Histidina

I	Isoleucina
Ile	Isoleucina
K	Lisina
L	Leucina
LCS	Longest Common Subsequence Problem
Leu	Leucina
LISP	List Processing
Lys	Lisina
M	Metionina
MATLAB	Matrix Laboratory
Met	Metionina
MIPS	Microprocessor without Interlocked Pipeline Stages
ML	Meta Language
MOSS	Measure of Software Similarity
N	Asparagina
NCBI	National Center for Biotechnology
P	Prolina
Perl	Practical Extraction and Report Language
Phe	Fenilalanina
Pro	Prolina
Q	Glutamina
R	Arginina
RKR	Running Karp-Rabin
RNA	Ácido Ribonucleico
S	Serina
Sbjct	Subject
Ser	Serina
T	Timina
T	Treonina
TCL	Tool Command Language
Thr	Treonina
Trp	Triptofano
Tyr	Tirosina

V	Valina
Val	Valina
VHDL	Very High-Speed Integrated Circuit Hardware Description Language
VN	Verdadeiros Negativos
VP	Verdadeiros Positivos
W	Triptofano
Y	Tirosina
YAP	Yet Another Plague

SUMÁRIO

1 INTRODUÇÃO

O plágio em programação ou código-fonte pode ser definido como a reutilização não autorizada da implementação de um programa por meio da cópia de sua estrutura e sintaxe (BURROWS; TAHAGHOGHI; ZOBEL, 2007). Diferente do uso autorizado que permite a reutilização de código-fonte no desenvolvimento de software. Por exemplo, optar pelo reúso de implementações de testes de projetos de código aberto para contribuir com maior segurança e agilidade no desenvolvimento de um software (MAKADY; WALKER, 2017).

Acerca do problema do plágio em programação, um estudo desenvolvido por Sraka e Kaucic (2009) mostrou que no meio acadêmico uma taxa de 72,5% dos alunos admitiram plagiar pelo menos uma vez em disciplinas de laboratório de programação. Esse problema pode afetar a aprendizagem dos alunos em disciplinas de programação (GOMES; MATOS, 2019). No entanto, esse ato de desonestidade não se limita ao meio acadêmico (CHUDA *et al*., 2012). De acordo com Ullah *et al*. (2018), todo sistema de software apresenta em sua implementação até 20% de código plagiado.

Diferentes técnicas e ferramentas foram criadas para automatizar a detecção de similaridade entre códigos-fontes, tendo o seu uso para a detecção tanto da reutilização autorizada como não autorizada (RAGKHITWETSAGUL, 2017; MAKADY; WALKER, 2017). A realização manual dessa tarefa demanda muito esforço e pode se tornar inviável em determinados casos (PRADO; BISPO; ANDRADE, 2018). As soluções para a detecção do uso não autorizado de código-fonte propõem detectar os diferentes níveis ou tipos existentes de modificações em códigos-fontes plagiados, dois dos principais estudos que categorizam as modificações são: os 6 níveis de Faidhi e Robinson (1987) e os 4 tipos de Roy e Cordy (2007).

As técnicas comumente utilizadas nas soluções são fundamentadas em: grafos, árvores, *tokens*, métricas, comparação textual, dentre outras abordagens (OLIVEIRA *et al*., 2016; ARWIN; TAHAGHOGHI, 2006). Algumas das ferramentas desenvolvidas a partir do uso dessas técnicas existentes são: JPlag, *Measure of Software Similarity* (MOSS), *Sim*, *Plague*, *Yet Another Plague* (YAP), *Plaggie*, *Fast Plagiarism-Detection System* (FPDS), *Marble* (ĐURIĆ; GAŠEVIĆ, 2012).

Com o intuito de identificar o atual cenário dos estudos associados a detecção de plágio em código-fonte, realizou-se neste trabalho um mapeamento sistemático. O modelo de mapeamento proposto e aplicado é fundamentado no método de Rattan, Bhatia e Singh (2013) e o período de abrangência de estudos é entre 2013 e 2018.

Analisando os estudos da área de plágio em programação, diferentes propostas de soluções nesse domínio foram identificadas e informações relevantes foram extraídas tais como: quais são as principais técnicas, quais são as linguagens de programação suportadas pelas soluções e como é feito o processo de validação da solução proposta, dentre outras. A partir dos resultados encontrados pelo mapeamento contatou-se que a maioria das propostas utilizam uma combinação de técnicas e suportam mais de uma linguagem de programação. O processo de validação comumente utilizado pelas propostas é realizado por meio de experimentos com até 100 códigos-fonte para serem testados em busca de diferentes tipos ou níveis de plágio.

Muitas ferramentas de detecção foram propostas e desenvolvidas, as duas mais utilizadas como referências no estado da arte são: MOSS e JPLAG (AHADI; MATHIESON, 2019; ĐURIĆ; GAŠEVIĆ, 2012; LANCASTER; CULWIN, 2004). No entanto, essas ferramentas incluindo as mais utilizadas apresentam limitações em relação a capacidade de detectar diferentes níveis ou tipos de modificações em códigos-fontes. Um fator limitante principal para essas ferramentas é a associação da qualidade da detecção com a complexidade de tempo de execução para que não inviabilize a automação desta tarefa.

O presente trabalho cria uma abordagem, nomeada BIOPLAG, com o objetivo de realizar a detecção dos 6 (seis) níveis de plágio em código-fonte de Faidhi e Robinson (1987). A elaboração da BIOPLAG é fundamentada na combinação de técnicas da Bioinformática e da Ciência da Computação. As seguintes técnicas são utilizadas: *tokens* de elementos de uma linguagem de programação, mapeamento de *bits* em sequências biológicas sintéticas e alinhamento de sequências biológicas. Os conceitos aplicados da bioinformática são baseados no método de detecção de vírus de computadores desenvolvido por Pedersen *et al*. (2012).

A BIOPLAG tem como característica o suporte a qualquer linguagem de programação. Neste trabalho, a sua implementação utilizou um gerador de *tokens* para a linguagem C. Essa foi a linguagem de suporte escolhida para o processo

avaliativo. No entanto, pode ser utilizado outros geradores de *tokens* de qualquer linguagem de programação.

O processo avaliativo é realizado a partir da aplicação da BIOPLAG em 7 (sete) cenários de testes, dos quais 6 (seis) objetivam avaliar o desempenho na detecção dos níveis de plágio em código-fonte e 1 (um) é direcionado para encontrar falsos positivos/negativos. Os testes foram feitos a partir de implementações de códigos-fontes escritos em linguagem *C*, sendo que foram produzidos por meio de três experimentos reais.

Os experimentos realizados neste trabalho tiveram a participação de 25 (vinte e cinco) alunos de graduação, 3 (três) alunos de mestrado da Universidade Tecnológica Federal do Paraná campus Ponta Grossa e 6 (seis) programadores de uma empresa de desenvolvimento de software da região. Os voluntários a partir de uma base de códigos-fontes produziram exemplos de códigos-fontes plagiados para 6 (seis) cenários de testes.

Os cenários de testes utilizaram um total de 336 exemplos de códigos-fontes implementados em linguagem C para serem testados em pares, resultando em 168 testes para avaliar a detecção de 6 níveis de plágio em programação e casos de ausência de plágio. Para compor os casos de ausência de plágio, utilizou-se 60 exemplos não plagiados escolhidos aleatoriamente a partir da base de códigos-fontes utilizada nos experimentos.

A avaliação dos cenários de testes foi realizada por meio de os seguintes parâmetros avaliativos de desempenho: precisão, revocação e medida *F*. As ferramentas MOSS e JPLAG, consideradas como referências no estado da arte, foram testadas nesse processo avaliativo e comparadas com a BIOPLAG em relação ao desempenho na detecção de plágio em programação.

Os resultados obtidos pela BIOPLAG demonstraram que em nenhum cenário de teste o seu desempenho de precisão, revocação e medida *F* foi inferior em relação às ferramentas comparadas. Em suma, o seu desempenho foi superior em 4 (quatro) e igual em 3 (três) cenários de testes.

1.1 JUSTIFICATIVA

O propósito de automatizar a detecção de plágio em códigos-fontes é prover uma maior qualidade na busca por plágio, além de executar essa tarefa em tempo

hábil mesmo com uma elevada quantidade de códigos a serem avaliados (LANCASTER; CULWIN, 2004). A atividade de verificar exclusivamente de forma manual o plágio em programação demanda esforço e a eficácia da detecção é baixa (PRADO; BISPO; ANDRADE, 2018).

De acordo com Ullah *et al*. (2018), a detecção da similaridade entre códigos-fontes é uma tarefa crucial para o ambiente acadêmico e para a indústria de software. As soluções propostas nesse domínio podem ser aplicadas não apenas no campo de detecção de plágio em programação, mas em outros tais como: softwares de detecção de códigos duplicados utilizados na engenharia de software e ferramentas de produção de métricas de softwares (RAGKHITWETSAGUL, 2017).

A utilização dos detectores automáticos é essencial em ambientes de ensino para controlar as atividades de programação desenvolvidas por alunos. Muitas instituições de ensino já utilizam ferramentas para este propósito. Com o crescente desenvolvimento da tecnologia da informação qualquer informação pode ser acessada pela internet incluindo códigos de programação, e essa facilidade permite a condução de comportamentos de desonestidade em cursos de programação (ULLAH *et al*., 2018).

Para a indústria de software, a detecção de plágio nas implementações pode contribuir para o processo de manutenção do sistema (ROY; CORDY, 2007; SHANMUGHASUNDARAM; SUBRAMANI, 2015). Muitos engenheiros de software reutilizam ou copiam códigos prontos para acelerar o desenvolvimento de seus projetos (COSMA; JOY, 2012). No entanto, além de situações de plágio outros tipos de problemas podem surgir tais como a presença de erros e *bad smells* que podem afetar negativamente o ciclo de vida do software.

1.2 OBJETIVOS

Criar uma abordagem de detecção de níveis de plágio em códigos-fontes utilizando a técnica de *token*, mapeamento de artefato digital em sequência biológica sintética e alinhamento de sequências biológicas por meio da ferramenta BLAST.

Para proporcionar a condução do objetivo geral, os seguintes objetivos específicos foram definidos: i) a utilização de um gerador de *tokens* dos elementos da linguagem de programação escolhida para suporte; ii) o desenvolvimento de um algoritmo para mapear um código-fonte em uma sequências biológica sintética do tipo

escolhido; iii) implementação de um algoritmo de análise de relatório da ferramenta BLAST com a extração e processamento dos dados do alinhamento, e iv) a avaliação da implementação da abordagem criada por meio de 7 (sete) cenários de testes com 336 (trezentos e trinta e seis) códigos-fontes proporcionando 168 (cento e sessenta e oito) testes, considerando os seguintes parâmetros avaliativos de desempenho: precisão, revocação e medida *f*.

1.3 ORGANIZAÇÃO DA DISSERTAÇÃO

A estruturação deste trabalho está organizada em seis capítulos. O capítulo 1 é referente a contextualização do tema, assim como a justificativa, objetivos e organização do trabalho. O capítulo 2 apresenta a fundamentação teórica relevante para a compreensão dos principais conceitos abordados de bioinformática e plágio em programação.

No capítulo 3 é apresentada a condução do mapeamento sistemático adotado para identificar o estado da arte do plágio em códigos-fontes entre os anos de 2013 e 2018. Todas as etapas executadas nesse estudo são explicadas com maiores detalhes, indicando a forma como foram conduzidas, quais bases de dados utilizadas, dentre outros aspectos contemplados. O capítulo 4 apresenta a explicação da abordagem criada definindo os conceitos abordados, a diagramação, e a explicação de cada etapa do seu funcionamento.

O capítulo 5 mostra a implementação da BIOPLAG, além da definição dos cenários de testes aplicados no processo avaliativo e os resultados para cada parâmetro avaliado. Também, nesse capítulo apresenta a comparação de desempenho da BIOPLAG em relação às ferramentas: MOSS e JPLAG. Por fim, o capítulo 6 realiza a conclusão e indica os trabalhos futuros indicados para a continuidade desta pesquisa.

2 FUNDAMENTAÇÃO TEÓRICA

O plágio, segundo o dicionário Cambridge (CAMBRIDGE, 2019), pode ser definido como "ação de plagiar" ou "copiar o trabalho alheio". De acordo com (ULLAH *et al.*, 2018), considerando a área de programação, diversos estudos mostram que cada sistema de software contém de 5% até 20% de plágio em seu código-fonte.

Devido a este problema ético encontrado em diferentes meios e não apenas no acadêmico (CHUDA *et al.*, 2012), foram desenvolvidas abordagens de identificação de plágio. As soluções propostas são criadas a partir de estudos tanto da ciência da computação (PRECHELT; MALPOHL; PHILIPPSEN, 2002; FRANÇA *et al.*, 2018; KUO; CHENG; WANG, 2018; MEUSCHKE *et al.*, 2018), como de outras áreas: por exemplo a bioinformática (XU *et al.*, 2006; KANE; SPRINGER, 2007; REVETT, 2009; PEDERSEN *et al.*, 2012). Os detectores de plágio se concentram em dois domínios: textos e programação (POTTHAST *et al.*, 2010).

Este capítulo aborda as definições e contextualizações fundamentais para a compreensão do tema do presente trabalho. A seção 2.1 apresenta os conceitos da área de bioinformática tais como: alinhamento de sequências, método baseado em bioinformática aplicado em segurança da informação e a ferramenta BLAST (ALTSCHUL *et al.*, 1990). A seção 2.2 contempla as definições, técnicas e classificações relacionadas ao plágio em programação tais como: níveis de plágio em programação, técnica baseada em *token* e o detector JPLAG (PRECHELT; MALPOHL; PHLIPPSEN, 2002). Por fim, a seção 2.3 resume o capítulo apresentando as considerações finais.

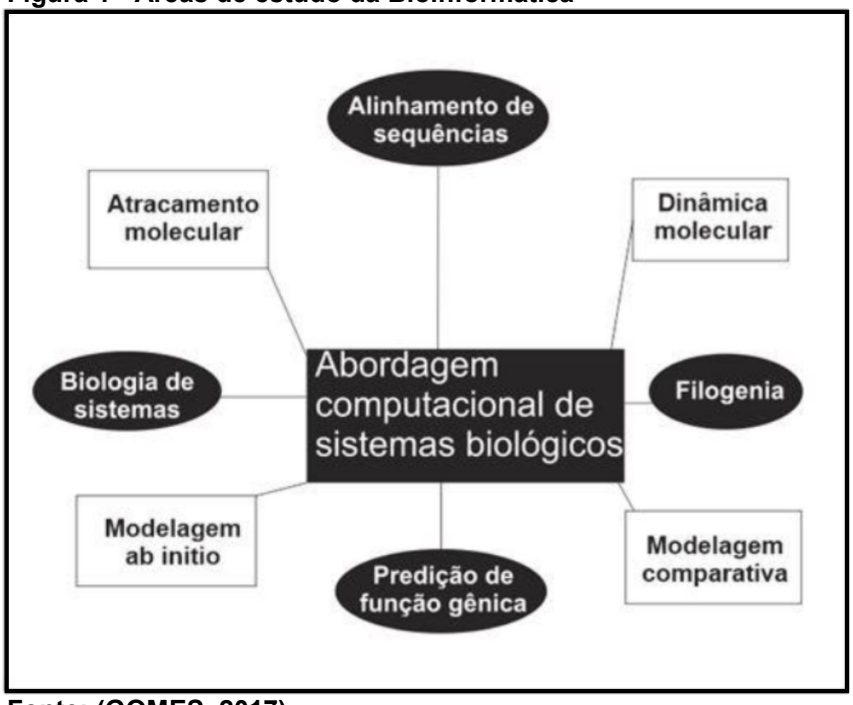
2.1 BIOINFORMÁTICA

No século XX, por meio de estudos de geneticistas e químicos, desenvolveram-se pesquisas científicas que obtiveram respostas para o entendimento do material genético. Em 1953, os estudos conduzidos por Watson e Crick possibilitou a compreensão da estrutura química de uma molécula de DNA (Ácido Desoxirribonucleico) (WATSON; CRICK *et al.*, 1953). Posteriormente, descobriram informações sobre o código genético e o fluxo de informação de polímeros como: ácidos nucléicos relacionados as proteínas (PROSDOCIMI *et al.*, 2002).

Em seguida, surgiram tecnologias para lidar com o sequenciamento dos polímeros, principalmente de DNA e proteínas. Uma nova área, chamada Biologia Molecular era responsável pelo estudo dessas estruturas. Com a utilização de sequenciadores automáticos, houve uma necessidade de manipular de maneira eficiente o armazenamento e processamento de sequências. Na década de 90, diante dos desafios computacionais impostos pelo avanço da Biologia Molecular, surgiu a Bioinformática (PROSDOCIMI *et al.*, 2002).

A bioinformática é caracterizada por associar o conhecimento de diferentes campos de estudos tais como: biologia molecular, matemática, estatística, biologia celular, engenharia de software, bioquímica, física e ciência da computação (ARAÚJO *et al*., 2008; PROSDOCIMI *et al*., 2002; WATERMAN, 1995). Dentre os seus diversos domínios de estudo, segundo Verli (2014), identifica-se duas áreas centrais de pesquisas: estrutura e sequências de biomoléculas. A Figura 1 mostra as linhas de pesquisas em cada área central, representando com formas geométricas circulares o campo de manipulação de sequências e com formas retangulares o campo de estruturas de biomoléculas.

Figura 1 - Áreas de estudo da Bioinformática



Fonte: (GOMES, 2017)

O estudo da estrutura de biomoléculas está associado ao entendimento de como funciona a interação de moléculas (atracamento). O relacionamento entre as

moléculas proporciona o desencadeamento de eventos, e para proporcionar a compreensão dessa dinâmica de eventos moleculares estuda-se modelos 3D de proteínas e outras biomoléculas. Por outro lado, a manipulação de sequências é fundamental para compreender os organismos em termos de complexidade biológica. Torna-se possível identificar a história evolutiva e relacionar indivíduos e populações. O foco de estudo dessa área incluem estudos em: alinhamento de sequências, identificação de padrões em sequências, construção de genomas e biologia de sistemas (VERLI, 2014).

Em problemas da computação, pode-se utilizar técnicas e ferramentas de Bioinformática para desenvolver soluções. Alguns exemplos de aplicação são: para computação de alta performance (KANE; SPRINGER, 2007), em segurança da informação (PEDERSEN *et al.*, 2012), em arquitetura SOA de *softwares* (XU *et al.*, 2006), verificação e autenticação de usuários no desenvolvimento de sistemas (REVETT, 2009), dentre outros.

Uma técnica comumente empregada nesses exemplos é a de alinhamento de sequências, detalhada na próxima seção. O alinhamento pode ser realizado em ferramentas de bioinformática tal como a BLAST, uma das mais utilizadas pela comunidade científica (ALTSCHUL *et al.*, 1990; ALTSCHUL *et al.*, 1997; PEDERSEN *et al.*, 2012; MCGINNIS; MADDEN, 2004). As seções a seguir apresentam os conceitos e detalhes associados à técnica de alinhamento de sequências, o método baseado em bioinformática de Pedersen (2012) que utiliza essa técnica e a ferramenta BLAST.

2.1.1 Alinhamento de Sequências Biológicas

As sequências utilizadas nos alinhamentos podem ser de diferentes tipos tais como: DNA, RNA e proteína (GOAD; KANEHISA, 1982). Dependendo da ferramenta de alinhamento pode-se utilizar qualquer tipo de sequência (TANG; XIAO; LU, 2009). Para a bioinformática o uso dessas ferramentas proporciona um apoio na investigação de funções de proteínas, relações evolucionárias entre diferentes organismos, dentre outras aplicações (JUNIOR; DIAS, 2019).

Um exemplo de aplicação de alinhamento de sequências de DNA em um problema da computação é o método desenvolvido por (PEDERSEN *et al.*, 2012), que

tem como objetivo identificar vírus de computadores. Nesse método, diferentemente do conceito biológico, uma estrutura de DNA representa qualquer artefato computacional. Nesse trabalho foi identificado que é possível mapear arquivos de computadores em sequências biológicas tais como de DNA, proteína, dentre outras.

O ácido desoxirribonucleico (DNA) é responsável por guardar a informação genética de um determinado organismo vivo. Em sua composição encontra-se quatro tipos diferentes de nucleotídeos, representados pelas seguintes bases nitrogenadas: Adenina (A), Timina (T), Guanina (G) e Citosina (C). A formulação de uma sequência de DNA é feita por meio de uma cadeia polinucleotídica (ALBERTS *et al.*, 2017).

A proteína está associada com o metabolismo dos organismos desempenhando diferentes funções. Em sua estrutura, define-se como um polímero formado por uma sequência linear de aminoácidos (HUNTER, 2009). Os aminoácidos são as moléculas que formam a macromolécula de proteína, e possuem 20 formas diferentes. O Quadro 1 mostra a identificação de cada uma dessas variações em uma sequência biológica por meio da abreviação de seus nomes.

Quadro 1 - Os 20 aminoácidos que compõem as proteínas

Nome	Abreviatura (3 letras)	Abreviatura (1 letra)
Alanina	Ala	A
Cisteína	Cys	C
Asparato	Asp	D
Glutamato	Glu	E
Fenilalanina	Phe	F
Glicina	Gly	G
Histidina	His	H
Isoleucina	Ile	I
Lisina	Lys	K
Leucina	Leu	L
Metionina	Met	M
Asparagina	Asn	N
Prolina	Pro	P
Glutamina	Gln	Q
Arginina	Arg	R
Serina	Ser	S
Treonina	Thr	T
Valina	Val	V

Triptofano	Trp	W
Tirosina	Tyr	Y

Fonte: Autoria Própria

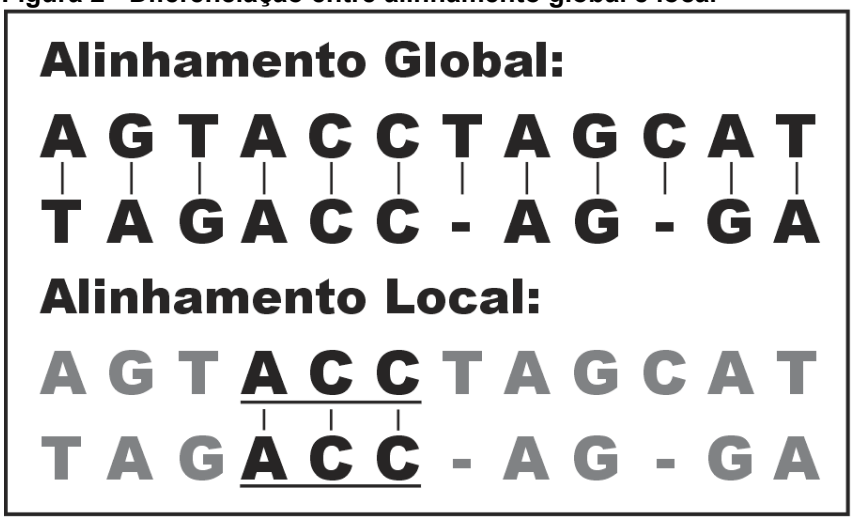
Para a computação, o DNA pode ser compreendido como uma cadeia de caracteres dos quais pertencem a um conjunto finito de símbolos (ARYA *et al.*, 2018; ORABI *et al.*, 2014). O alfabeto ou conjunto é representado por {A, C, G, T}, que são referências aos nucleotídeos por meio da inicial de cada base nitrogenada. Analogamente, as proteínas são representadas por meio de seus aminoácidos que podem ser identificados com as suas respectivas abreviaturas, conforme o Quadro 1.

O alinhamento dessas sequências biológicas consiste em comparar duas ou mais sequências em busca de uma série de caracteres ou padrões de caracteres que estão em uma mesma ordem na comparação (DAVID, 2001). Este método objetiva analisar o nível de similaridade entre sequências. O seu funcionamento pertence a mesma classe de problemas de encontrar a maior subsequência comum (*Longest common subsequence problem - LCS*) (COULL *et al.*, 2004). A figura 2 mostra como funciona um alinhamento tanto do tipo global como local, que nesse exemplo são para sequências de DNA.

Um alinhamento é global quando a comparação de caractere a caractere é feita considerando o tamanho inteiro das sequências, buscando obter a maior quantidade de comparações possíveis. Ao contrário do global, o tipo local busca apenas regiões com maior grau de similaridade ou igualdade de caracteres. Não é feito o alinhamento de todos os caracteres de uma sequência em relação a todos os caracteres da outra, como é feito necessariamente no global independente de as comparações serem iguais ou diferentes (DAVID, 2001).

Por exemplo, a Figura 2 ao considerar duas sequências de DNA, mostra que no alinhamento global realiza-se a comparação de cada nucleotídeo. Por outro lado, no local realiza-se apenas em uma região específica, percebe-se que nesse exemplo desconsiderou regiões pertencentes as extremidades. Essa desconsideração foi guiada pela busca por regiões que apresentam alto grau de similaridade, nesse caso aqueles nucleotídeos que fossem iguais nas comparações.

Figura 2 - Diferenciação entre alinhamento global e local



Fonte: Adaptado de (COULL *et al.*, 2004)

Ressalta-se que os caracteres comparados em uma região com alto grau de similaridade não precisam necessariamente serem iguais, o grau de similaridade depende de um sistema de pontuação. O nível de similaridade calculado por um alinhamento depende de um critério de pontuação. Um valor positivo é atribuído em casos de igualdade ou *match* de caracteres, situações em que ambos os caracteres são exatamente os mesmos. Por outro lado, um valor negativo é atribuído em casos de desigualdade. Esse cenário desigual pode surgir a partir de dois casos especiais: espaçamento (*gap*) ou desigualdade (*mismatch*).

O caso do espaçamento é quando existe a ausência de caracteres na comparação, enquanto na desigualdade existe a presença de dois caracteres diferentes. Para cada um dos casos, atribui-se um valor negativo diferente para penalizar a pontuação final do alinhamento (COULL *et al.*, 2004). Por exemplo, considerando a Figura 2, no alinhamento global a quarta comparação (adotando o início na extremidade esquerda) é um caso de igualdade entre as bases: "A" e "A". A primeira comparação é um caso de desigualdade entre as bases: "A" e "T". Por outro lado, na sétima comparação existe um caso de espaçamento entre a base "T" e a ausência de uma outra base denotada como "-".

A pontuação ou *score* que representa o resultado final do alinhamento de sequências é a soma das pontuações individuais calculadas em cada comparação a partir da análise dos três casos possíveis (igualdade, desigualdade e espaçamento). As ferramentas que executam esses alinhamentos apresentam não apenas esse resultado, mas outros parâmetros avaliativos para complementar a análise. As seções

seguintes abordam a aplicação do conceito de alinhamento de sequência biológica e a sua execução por uma ferramenta da Bioinformática.

2.1.2 Método Baseado em Bioinformática de Pedersen

O método proposto por Pedersen *et al.* (2012) é fundamentado em conceitos da Bioinformática para resolver um problema da computação, especificamente na área de segurança da informação. A solução contempla uma forma de identificar *malwares* em computadores. A abordagem proposta por este autor pode ser aplicada em outros campos da computação como: identificação de plágio em códigos-fonte e de tipos de arquivos.

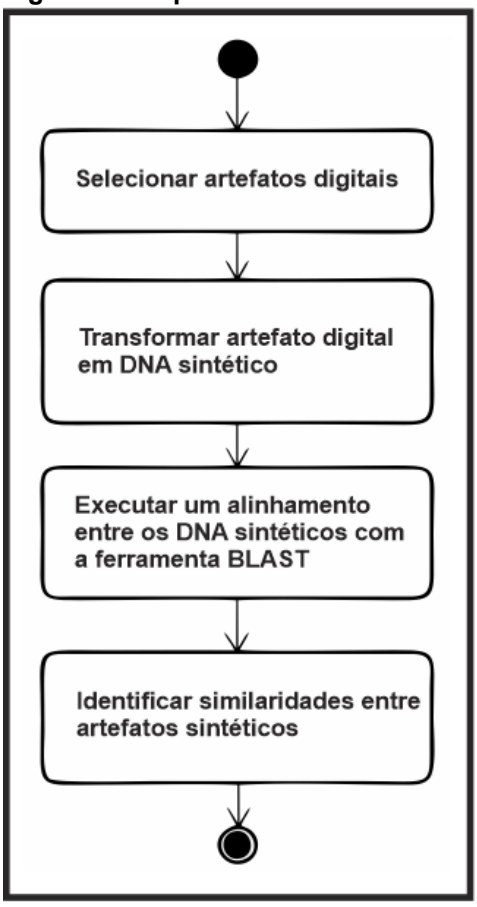
O funcionamento do método consiste em quatro etapas sequenciais: I, II, III e IV ilustradas na Figura 3. Para verificar se um arquivo apresenta ou não um determinado *malware* escondido ou camuflado aplica-se conceitos e ferramentas da bioinformática.

Em suma, o arquivo a ser verificado é convertido em uma sequência de DNA. Também, uma amostra ou um arquivo infectado com o determinado *malware* avaliado é convertido da mesma forma. Essa conversão depende de um mapeamento de *bits* em bases nitrogenadas, de modo a gerar arquivos do tipo *FASTA* que representam um DNA sintético. Esses arquivos de DNA sintético são entradas para uma ferramenta de alinhamento, neste caso o BLAST, para identificar a similaridade entre o arquivo verificado e o determinado *malware* contemplado. Caso exista uma alta similaridade no alinhamento, identifica-se a presença do vírus de computador.

A etapa I é responsável por selecionar os arquivos de interesse, que são chamados de artefatos digitais. Nesse contexto da aplicação, escolhe-se um arquivo qualquer suspeito de conter um vírus e o arquivo do próprio *malware* em questão.

Os artefatos digitais escolhidos são entradas para a etapa II, que realiza a transformação deles em DNA sintético. Um DNA sintético é a representação desses arquivos na forma de uma estrutura biológica. A criação dessas estruturas é feita por meio do mapeamento de *bits* em bases nitrogenadas. Essa conversão é realizada a partir da representação binária dos arquivos selecionados, que é feita por meio da tabela ASCII completa. O formato do arquivo da sequência de DNA sintético é do tipo *FASTA* desenvolvido por (PEARSON; LIPMAN, 1988).

Figura 3 - Etapas do método de Pedersen

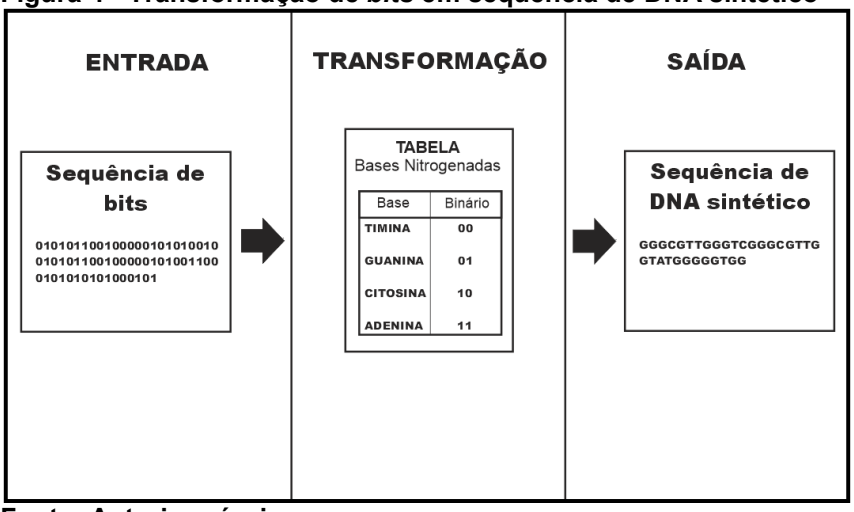


Fonte: Autoria própria

A Figura 4 ilustra a etapa de geração de DNA sintético. Esse processo é feito transformando uma sequência de *bits*, que é a representação binária de um artefato digital, em bases nitrogenadas. O mapeamento é feito com auxílio da tabela e o resultado ou saída dessa etapa é uma sequência de DNA sintético. Esse procedimento é feito para cada arquivo de interesse.

Uma vez que as sequências de DNA sintético estão geradas, realiza-se a etapa III. Nessa etapa, um alinhamento com as sequências por meio da ferramenta BLAST é obtido. O alinhamento realizado é do tipo local e o resultado produzido pela ferramenta é um relatório com diversas medidas de comparação assim como as regiões de alta similaridade identificadas.

Figura 4 - Transformação de *bits* em sequência de DNA sintético



Fonte: Autoria própria

Após realizar o alinhamento, analisa-se o relatório do mesmo na etapa IV. O objetivo principal é identificar o nível de similaridade entre as sequências. Considerando o grau da similaridade estabelecido é possível identificar se existe ou não um *malware* no arquivo suspeito. Devido à natureza do alinhamento local, no relatório identificam-se regiões de alta similaridade, o que ajuda na identificação de arquivos camuflados com o vírus. Pode acontecer de apenas uma pequena porção do arquivo apresentar o vírus e o restante não estar infectado.

A seguir na seção 2.1.3 é apresentada a ferramenta BLAST utilizada neste método de Pedersen (2012) para realizar o alinhamento. Dentre as diversas ferramentas de Bioinformática utilizadas para realizar o alinhamento de sequências: BLAST, ClustalW, Muscle, dentre outras (AGARWAL *et al.*, 2018; SAITOU, 2018), a ferramenta BLAST foi a escolhida na elaboração da criação da abordagem apresentada no Capítulo 4. A sua escolha é justificada pela possibilidade de utilização de diferentes módulos integrados capazes de lidar com os principais tipos de sequências biológicas. Além de apresentar as características de: criação de base de sequências e execução de alinhamento do tipo local por meio de um conjunto de parâmetros.

2.1.3 Ferramenta de Bioinformática: BLAST

A ferramenta *Basic Local Alignment Search Tool* (BLAST) é uma abordagem de comparação de sequências desenvolvida em 1990. A aplicação do seu algoritmo permite funcionalidades tais como: busca de bases de dados de sequências de DNA

e proteína, busca de padrões de aminoácidos e nucleotídeos, busca de identificação de gene, análise de múltiplas regiões de similaridade ao longo de uma sequência de DNA, dentre outras. Essa abordagem tem como característica a execução de um alinhamento local e a possibilidade de ser implementada para diferentes contextos (ALTSCHUL *et al.*, 1990).

Uma das implementações da abordagem foi feita pela *National Center for Biotechnology Information* (NCBI), que acrescentou características específicas para o alinhamento de diferentes tipos de sequências (MCGINNIS; MADDEN, 2004). O Quadro 2 mostra quais são os tipos suportados de sequências e os seus respectivos sistemas de pontuações adotados no alinhamento. A ferramenta NCBI BLAST possui diversas versões, uma delas é apresentada em (ZHANG *et al.*, 2004). No presente trabalho, essa versão foi utilizada na elaboração da abordagem de detecção de plágio em programação apresentada no Capítulo 4.

O Quadro 2 apresenta os módulos da ferramenta que são divididos por tipo de sequências. As categorias são: nucleotídeo, proteína, traduzido e páginas especiais. A categoria de nucleotídeo é composta pelos seguintes módulos: *MegaBLAST* descontínuo, *MegaBLAST*, *BLASTN* Padrão e sequência curta de nucleotídeo. Para a categoria de proteína: *Blastp* Padrão, *Psi-BLAST*, *Phi-BLAST*, *Sequência curta de proteína* e *RPS-BLAST*. Para os traduzidos são: *Blastx*, *Tblastn* e *Tblastx*. Por fim, a categoria de páginas especiais: *BLAST* Genoma. No quadro são apresentados os respectivos valores de parâmetros padrões por Módulos utilizados pelas ferramentas NCBI BLAST. Para cada situação de alinhamento é atribuído um valor específico de pontuação que varia de acordo com o tipo de sequência, com exceção de alguns casos em que o sistema de pontuação não se aplica (N/A).

Além de realizar o alinhamento, a ferramenta NCBI BLAST fornece um relatório com dados e estatísticas para avaliar a similaridade encontrada. De acordo com a versão 2.2.28+, os parâmetros avaliativos são: porcentagem de identidade, o valor esperado (*expect value*), porcentagem de espaçamento (*gap*) e a pontuação (*score*) (MCGINNIS; MADDEN, 2004). Outras informações como a visualização do alinhamento e a ordem de leitura (*score*Strand) são fornecidas para completar o resultado produzido pela ferramenta.

Um exemplo do relatório gerado é ilustrado pela Figura 5. O valor de pontuação (*score*) é calculado a partir do sistema de pontuação adotado, de modo a somar todos os valores individuais gerados por cada situação de *match*, *mismatch* e

gap. Quanto maior o valor *score* obtido, maior é a similaridade entre as sequências alinhadas. O valor esperado (*E-value* ou *Expect*) é a taxa de falso-positivo e representa a chance de encontrar alinhamentos diferentes com pontuação igual ou superior na base de dados de sequências avaliada. Quanto menor o valor *E-value*, maior será a significância do alinhamento e a sua respectiva pontuação.

Quadro 2 - Parâmetros padrões do sistema de pontuação da ferramenta NCBI BLAST

Tipo de Sequência (Módulos NCBI BLAST)	Igualdade (*Match*)	Desigualdade (*Mismatch*)	Existência de Espaçamento (*Gap existence*)	Extensão de Espaçamento (Gap extension)
MegaBLAST Descontínuo	1	-2	0	2,5
MegaBLAST	1	-2	0	2,5
BLASTN Padrão	1	-3	5	2
Sequência curta de nucleotídeo	1	-3	5	2
Blastp Padrão	N/A	N/A	11	1
Psi-BLAST	N/A	N/A	11	1
Phi-BLAST	N/A	N/A	11	1
Sequência curta de proteína	N/A	N/A	11	1
RPS-BLAST	N/A	N/A	11	1
Blastx	N/A	N/A	11	1
Tblastn	N/A	N/A	11	1
Tblastx	N/A	N/A	11	1
BLAST Genoma	1	-2	0	0

Fonte: Adaptado de (MCGINNIS; MADDEN, 2004)

A porcentagem de identidade (*Identities*) é a porcentagem de resíduos (nucleotídeos para DNA, aminoácidos para Proteína) idênticos considerando as mesmas posições nas sequências comparadas. O valor do espaçamento (*gap*) é a porcentagem de espaços em um alinhamento para compensar a inserção ou deleção de resíduos de uma sequência em relação a outra. Quanto maior o valor de *gap*, menor será o *score*. Por fim, o parâmetro *strand* indica o sentido em que os resíduos são alinhados. Em uma sequência de DNA cada extremidade, conhecida por posição: 5' e 3', é utilizada para indicar o sentido da realização do alinhamento. A visualização do alinhamento é apresentada nomeando a sequência de busca como *"Query"* e a sequência contida na base de dados como *"Sbjct"* (FASSLER; COOPER, 2011; NILSSON *et al.*, 2012).

O alinhamento realizado pela abordagem BLAST é mais eficiente em termos de complexidade de tempo de execução em relação a outros métodos consolidados neste campo de estudo como: *Needleman-Wunsch* e *Smith-Waterman* (TATUSOVA; MADDEN, 1999). A complexidade da abordagem considerando o tempo de execução no pior caso é $O(MN)$, sendo M o tamanho da sequência de entrada ou query, e N o tamanho do banco de dados de sequências buscado (KAUR; KAUR; SOOD, 2018). A sua maior eficiência em tempo de execução é conquistada por meio de uma heurística em seu algoritmo que otimiza a identificação da similaridade. Esse detalhe na abordagem permite que seja feita uma escolha entre velocidade e sensibilidade no alinhamento. A partir de uma limiarização na abordagem é possível aumentar a velocidade e diminuir a sensibilidade ou diminuir a velocidade e aumentar a sensibilidade. A velocidade está relacionada com o tempo de execução e a sensibilidade com a probabilidade de encontrar qualquer tipo de similaridade na comparação (ALTSCHUL *et al*., 1997).

Figura 5 - Relatório gerado pela ferramenta NCBI BLAST

```
 Score = 1136 bits (615),  Expect = 0.0
 Identities = 633/641 (99%), Gaps = 6/641 (1%)
 Strand=Plus/Minus

Query  1    TATTGATATGCTTAAGTTCAGCGGGTATTCCTACCTGATCCGAGGTCAACATTTGCATGA  60
            ||||||||||||||||||||||||||||||||||||||||||||||||||||| || ||
Sbjct  635  TATTGATATGCTTAAGTTCAGCGGGTATTCCTACCTGATCCGAGGTCAACATTT-CA-GA  578

Query  61   AGTTGGGTGTTTTACGGACGTGGACGCGCCGCGCTCCCGGTGCGAGTTGTGCAAACTACT  120
            ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
Sbjct  577  AGTTGGGTGTTTTACGGACGTGGACGCGCCGCGCTCCCGGTGCGAGTTGTGCAAACTACT  518

Query  121  GCGCATGAGAGGCTGCGGCGAGACCGCCACTGTATTTCGGGGCCGGGATCCCGTCTTAGG  180
            ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
Sbjct  517  GCGCATGAGAGGCTGCGGCGAGACCGCCACTGTATTTCGGGGCCGGGATCCCGTCTTAGG  458

Query  181  GGTTCCCGAAGTCCCCAACGCCGACCCCCCGGAGGAGGGGTTCGAGGGTTGAAATGACGC  240
            |||||||||||||||||||||||||||||    ||||||||||||||||||||||||||||
Sbjct  457  GGTTCCCGAAGTCCCCAACGCCGACCCCCC---GGAGGGGTTCGAGGGTTGAAATGACGC  401

Query  241  TCGGACAGGCATGCCCGCCAGAATACTGGCGGGCGCAATGTGCGTTCAAAGATTCGATGA  300
            ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
Sbjct  400  TCGGACAGGCATGCCCGCCAGAATACTGGCGGGCGCAATGTGCGTTCAAAGATTCGATGA  341

Query  301  TTCACTGAATTCTGCAATTCACATTACTTATCGCATTTCGCTGCGTTCTTCATCGATGCC  360
            ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
Sbjct  340  TTCACTGAATTCTGCAATTCACATTACTTATCGCATTTCGCTGCGTTCTTCATCGATGCC  281
```

Fonte: Adaptado de (NILSSON *et al*., 2012)

As seções a seguir contemplam as definições, técnicas e categorização do plágio em programação. Para a compreensão desse tipo de plágio são abordados os temas: classificação de plágio de código-fonte, funcionamento da técnica de *tokens* de detecção de plágio, e a ferramenta JPLAG de detecção de plágio em programação. Tanto a técnica como a ferramenta apresentada são utilizadas na elaboração da abordagem criada no Capítulo 4.

2.2 PLÁGIO EM PROGRAMAÇÃO

O plágio é um comportamento antiético que está associado com o reuso do trabalho de outros autores sem referenciá-los (ĐURIĆ; GAŠEVIĆ, 2012). Um exemplo é o plágio de programação, do qual códigos-fonte são modificados ou camuflados com pedaços parciais ou integrais da implementação de outros autores. Esse é um problema crescente no meio acadêmico, em especial nos cursos que envolvem laboratórios de programação (MAURER; KAPPE; ZAKA, 2006).

Os plagiadores na programação realizam diferentes tipos de modificações no código-fonte para que os detectores de plágio não consigam identificar que a implementação foi gerada a partir da cópia parcial ou até mesmo integral de outros trechos de códigos. Neste domínio de estudo, alguns autores criaram uma categorização das prováveis ações que os plagiadores possam realizar em termos de mudanças na implementação. Por exemplo, a categorização dos 4 níveis de (ROY; CORDY, 2007) e os 6 níveis de (ROY; CORDY, 2007; FAIDHI; ROBINSON, 1987).

Para realizar a detecção de plágio neste domínio, algumas técnicas da própria computação foram desenvolvidas. A baseada em métricas realiza a contabilização de dados referentes ao código. Por exemplo, o número de caracteres por linha, linhas comentadas, linhas indentadas, palavras-chaves, dentre outras. A baseada em textos computa a similaridade analisando um código-fonte como um conjunto de linhas de caracteres. A baseada em *tokens* que transforma um código em unidades de generalização de elementos como: variáveis, operadores matemáticos, palavras-chave, comentários, dentre outros. A fundamentada em árvores, conhecida como *Abstract Syntax Trees*, realiza a representação abstrata de cada elemento de um código por meio de uma estrutura hierárquica. A baseada em grafos utiliza o conceito de grafos para modelar um código-fonte, em que as declarações são os vértices e as dependências de elementos e controle entre declarações são as arestas. Outras técnicas são consideradas híbridas, das quais associam a uma ou mais abordagem diferente na composição da solução proposta (OLIVEIRA *et al.*, 2016; ARWIN; TAHAGHOGHI, 2006).

As ferramentas de detecção de plágio em programação são soluções que automatizam o processo de identificação do nível de similaridade entre dois ou mais códigos-fonte. Alguns exemplos de ferramentas são: JPLAG, *Measure of Software*

Similarity (MOSS), *Sim*, *Plague*, *Yet Another Plague* (YAP), *Plaggie*, *Fast Plagiarism-Detection System* (FPDS), *Marble*, dentre outras.

De acordo com o estado da arte e a literatura, as duas principais ferramentas utilizadas para comparar e validar outras propostas de abordagem de detecção de plágio em códigos-fonte são: JPLAG e MOSS. Algumas das características avaliadas em uma ferramenta de detecção de plágio em programação é o número de linguagens de programação suportadas, usabilidade, apresentação do resultado, tipo de licença de software, complexidade de tempo de execução, dentre outras (ĐURIĆ; GAŠEVIĆ, 2012).

A seção a seguir apresenta os diferentes níveis ou tipos de modificações em códigos de programação que uma solução de detecção de plágio em programação pode conseguir contemplar. Na literatura encontra-se diferentes categorias de modificações em códigos, por exemplo: os 4 tipos de plágio segundo Roy e Cordy (2007), e os 6 níveis de modificações elaborado por Faidhi e Robinson (1987). Os 6 níveis de (FAIDHI; ROBINSON, 1987; ROY; CORDY, 2007) são detalhados a seguir, uma vez que são utilizados na abordagem criada deste trabalho por abrangerem mais tipos de modificações em códigos-fonte em relação ao modelo de Roy e Cordy (2007).

2.2.1 Níveis de Plágio em Programação

Na literatura de detecção de plágio em programação, encontra-se diferentes abordagens para categorizar as modificações aplicadas em um código-fonte plagiado (FAIDHI; ROBINSON, 1987; ROY; CORDY, 2007). Por exemplo, os 6 níveis de modificações elaborado por Faidhi e Robinson (1987).

A abordagem dos 6 níveis de plágio é comumente utilizada por trabalhos que procuram validar uma nova solução para esse domínio de estudo. A intenção de utilizar uma abordagem de classificação está relacionada com a possibilidade de validar e identificar a eficiência das ferramentas diante de cada cenário específico (PARKER; HAMBLEN, 1989; VERCO; WISE, 1996; NOH, 2003; ZEIDMAN, 2006; MENAI; AL-HASSOUN, 2010; BIN-HABTOOR; ZAHER, 2012; FLORES *et al.*, 2012; INOUE; WADA, 2012; CRUZ *et al.*, 2014; LI *et al.*, 2015; KARNALIM, 2017a).

De acordo com a classificação dos 6 níveis de Faidhi e Robinson (1987), o nível 1 representa as mudanças em comentários e indentações. A Figura 6 apresenta

um exemplo dessas modificações. No exemplo, percebe-se que apenas a indentação do conteúdo da função "*main()"* e os comentários existentes são alterados.

Figura 6 - Exemplo de nível 1 de plágio

Código-fonte Original	**Código-fonte plagiado (nível 1)**

int main()
{
int X = 1; // Comentário-1
int Y = 2; // Comentário-2
if (X > Y) {
X = Y;
}
return 0;
}
	int main()
	{
	int X = 1; // Comentário-1'
	int Y = 2; // Comentário-2'
	if (X > Y) {
	X = Y;
	}
	return 0;
	}

Fonte: Autoria própria

O nível 2 representa as mudanças em identificadores, além das provocadas pelo nível anterior. A Figura 7 ilustra um exemplo com mudanças no nome das variáveis declaradas, comentários e indentação.

O nível 3 representa as alterações de nível 2 e em declarações de variáveis, como: mudar a posição das variáveis e declarar outras variáveis adicionais, e posicionamento de funções/procedimentos. A Figura 8 apresenta um exemplo de modificação por posicionamento de variáveis, nome das variáveis declaradas, comentários e indentação.

Figura 7 - Exemplo de nível 2 de plágio

Código-fonte Original	**Código-fonte plagiado (nível 2)**

int main()
{
int X = 1; // Comentário-1
int Y = 2; // Comentário-2
if (X > Y) {
X = Y;
}
return 0;
}
	int main()
	{
	int a = 1; // Comentário-1'
	int b = 2; // Comentário-2'
	if (a > b) {
	a = b;
	}
	return 0;
	}

Fonte: Autoria própria

O nível 4 são modificações de nível 3 e nas funções/procedimentos, por exemplo: criar uma função a partir da junção de duas ou mais existentes, dividir uma função entre duas ou mais, alterar os parâmetros, dentre outras manipulações associadas. A Figura 9 ilustra esse nível por meio da criação de uma função que insere integralmente um fragmento de código de outra função, além de modificações de nível 1, 2 e 3.

Figura 8 - Exemplo de nível 3 de plágio

Código-fonte Original	**Código-fonte plagiado (nível 3)**

int main()
{
int X = 1; // Comentário-1
int Y = 2; // Comentário-2
if (X > Y) {
X = Y;
}
return 0;
}
	int main()
	{
	int b = 2; // Comentário-2'
	int a = 1; // Comentário-1'
	if (a > b) {
	a = b;
	}
	return 0;
	}

Fonte: Autoria própria

O nível 5 indica modificações de nível 4 e em estruturas de repetições. Por exemplo, a alteração de um laço "*for*" por outro equivalente como o "*while*". A Figura 10 apresenta um exemplo de nível 5 de modificação, no qual acontece uma mudança de um laço "*for*" para um "*while*".

Figura 9 - Exemplo de nível 4 de plágio

Código-fonte Original	**Código-fonte plagiado (nível 4)**

int main()
{
int X = 1; // Comentário-1
int Y = 2; // Comentário-2
if (X > Y) {
X = Y;
}
return 0;
}
	int func()
	{
	int b = 2; // Comentário-2'
	int a = 1; // Comentário-1'
	if (a > b) {
	a = b;
	}
	return 0;
	}
	void main() {
	func();
	}

Fonte: Autoria própria

Figura 10 - Exemplo de nível 5 de plágio

Código-fonte Original	Código-fonte plagiado (nível 5)

int main()
{
int cont; //
for (cont = 1; cont<11;cont++)
{
printf("%d",cont);
}
return 0;
}
	int main()
	{
	int cont = 1; //
	while (cont < 11)
	{
	printf("%d",cont);
	cont++;
	}
	return 0;
	}

Fonte: Autoria própria

O nível 6 corresponde as mudanças de nível 5 e nas decisões lógicas de estruturas de controle tais como os laços de repetição, condicionais, dentre outras. A Figura 11 apresenta um código plagiado a partir da modificação de um "*for*" para um "*while*", e da expressão lógica de decisão da estrutura de repetição "*while*".

Figura 11 - Exemplo de nível 6 de plágio

Código-fonte Original	Código-fonte plagiado (nível 6)

int main()
{
int cont; //
for (cont = 1; cont<11;cont++)
{
printf("%d",cont);
}
return 0;
}
	int main()
	{
	int cont = 1; //
	while (cont <= 10)
	{
	printf("%d",cont);
	cont = cont + 1;
	}
	return 0;
	}

Fonte: Autoria própria

A seção a seguir apresenta uma das técnicas da computação utilizada na detecção das modificações em códigos de programação de modo a identificar o plágio de programação. Algumas técnicas elaboradas pela computação são: grafos, árvores, *tokens*, métricas, comparação textual, dentre outras abordagens. A técnica de *token* foi escolhida por ser capaz de detectar diferentes níveis de modificações em códigos-fonte tais como: mudanças de nome de variáveis e funções, indentação, comentário, dentre outras.

2.2.2 Técnica Baseada em *Token*

De acordo com Roy e Cordy (2007), a técnica baseada em *token* consiste em transformar o código-fonte em uma sequência de *tokens*. A partir dessas sequências produzidas, executa-se uma análise em busca de subsequências em comum. A vantagem de se utilizar essa técnica é a capacidade de lidar com mudanças em códigos em termos de indentação, comentários, nomes de identificadores, tipos de estruturas de repetição, espaçamentos e formatação visual (ROY; CORDY, 2007; MOZGOVOY; KARAKOVSKIYZ; KLYUEV, 2008).

Os algoritmos fundamentados nessa técnica utilizam analisadores de conjuntos de caracteres para identificar os elementos de um código-fonte e os substituírem por *tokens*. Cada *token* é uma generalização de algum componente contido na implementação (POON *et al.*, 2012). Por exemplo, as variáveis são substituídas por VAR que representa qualquer variável, as estruturas de decisão do tipo "*if* " são representadas por CONDICIONAL, abre e fecha colchetes por abre-bloco e fecha-bloco respectivamente, dentre outras possíveis representações. A Figura 12 ilustra um exemplo de como seria um código convertido com esses rótulos de *tokens*.

O exemplo da Figura 12 apresenta um código contendo declarações de variáveis, comentários e condicional do tipo "*if* " com abertura e fechamento de um bloco de instrução. Todos esses elementos são convertidos em suas respectivas generalizações: VAR, Comentário, Condicional, abre-bloco e fecha-bloco. As modificações tais como as mudanças de variáveis, dentre outras, não são eficazes para camuflar uma situação de plágio diante dessa técnica. Os detectores de plágio que utilizam essa abordagem analisam o que a variável declarada representa em um modelo de código-fonte, ao contrário de outros métodos que representam a variável apenas pelo seu identificador (OLIVEIRA *et al.*, 2016).

Figura 12 - Código-fonte transformado em sequência de *token*

Código-fonte	Sequência de token

int X = 0
int Y = 1; // Comentário-1
int Z = 2; // Comentário-2
// Comentário-3
if (X > Y) {
  int W = 4;
}
	<VAR>
	<VAR> <Comentário>
	<VAR> <Comentário>
	<Comentário>
	<Condicional><abre-bloco>
	<VAR>
	<fecha-bloco>

Fonte: Autoria própria

Muitas ferramentas utilizam variações da técnica de *token* por meio de combinações com outros métodos para aprimorar o resultado da detecção de plágio em códigos-fonte. Uma das ferramentas que utilizam técnica de *token* é o JPLAG (ĐURIĆ; GAŠEVIĆ, 2012). Na seção a seguir detalhes dessa ferramenta são apresentados, uma vez que a abordagem criada neste trabalho a utiliza como parâmetro de comparação no processo de validação conforme apresentado no Capítulo 4.

2.2.3 Detector de Plágio em Programação: JPLAG

A ferramenta JPLAG desenvolvida por Prechelt, Malpohl e Philippsen (2002) é um detector automatizado de plágio em programação. A solução contempla a análise de códigos-fonte implementados nas seguintes linguagens: *Java*, *Scheme*, *C*, C# e *C++* (AHADI; MATHIESON, 2019).

O seu objetivo é fornecer uma medida de similaridade entre os códigos comparados, conforme as Figuras 13 e 14 é possível verificar como a ferramenta apresenta o seu resultado. Nesse exemplo, dois códigos-fontes implementados em linguagem *C* nomeados como "*exemploA.c*" e "*exemploB.c*" são analisados.

Figura 13 - Modelo de resultado fornecido pelo JPLAG parte 1

JPlag

Search Results

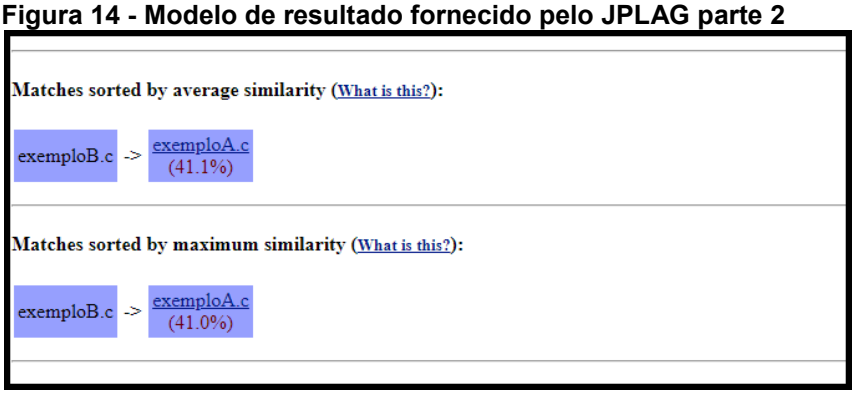
Title:	
Programs:	exemploA.c - exemploB.c
Language:	C/C++ Scanner [basic markup]
Submissions:	2
Matches displayed:	1 (Treshold: 41.1%) (average similarity) 1 (Treshold: 41.0%) (maximum similarity)
Date:	2020-04-11
Minimum Match Length (sensitivity):	12
Suffixes:	.cpp, .CPP, c++, C++, .c, .C, .h, .H, .hpp, .HPP

90% - 100%	0 .
80% - 90%	0 .
70% - 80%	0 .
60% - 70%	0 .
50% - 60%	0 .
40% - 50%	1 ##
30% - 40%	0 .
20% - 30%	0 .
10% - 20%	0 .
0% - 10%	0 .

Fonte: Autoria própria

O funcionamento do JPLAG depende de duas fases. A fase I se aplica a técnica baseada em *token*, nessa ocorre a transformação de cada elemento do código-fonte em uma sequência de *tokens*. A fase II se executa uma versão própria do algoritmo *Running Karp-Rabin Greedy String Tiling* (RKR-GST) para realizar a comparação par-a-par entre as sequências criadas na fase anterior (NOH, 2003; OLIVEIRA *et al.*, 2016). A execução desse processo de comparação de *tokens*, em seu pior caso, tem complexidade de tempo de execução O(n^3).

Figura 14 - Modelo de resultado fornecido pelo JPLAG parte 2



Matches sorted by average similarity (What is this?):

exemploB.c -> exemploA.c (41.1%)

Matches sorted by maximum similarity (What is this?):

exemploB.c -> exemploA.c (41.0%)

Fonte: Autoria própria

Para compor o resultado em porcentagem, o JPLAG considera o tamanho de cada código-fonte comparado e fornece duas opções: a similaridade média e a

máxima. No caso da primeira, realiza-se a média das proporções dos códigos-fontes cobertos por similaridade e seus respectivos tamanhos totais. Na segunda opção, o resultado é definido como sendo a maior dentre as proporções encontradas em cada código-fonte comparado.

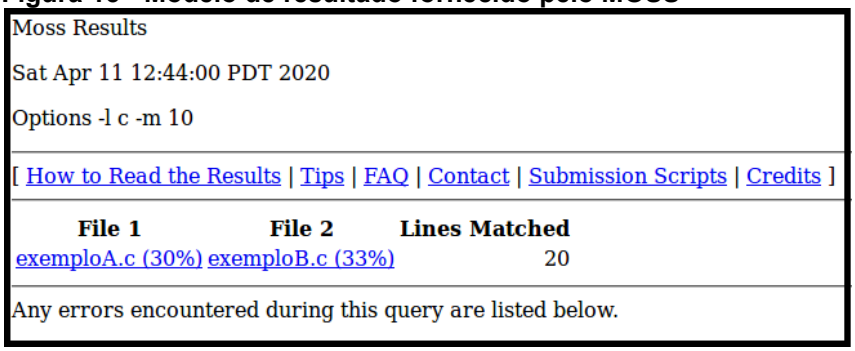
2.2.4 Detector de Plágio em Programação: MOSS

A ferramenta *Measure Of Software Similarity* (*MOSS*) é um detector automático de plágio em programação desenvolvido em 1994, seu uso segundo Araujo e Kyrilov (2020) tem sido direcionado na educação para avaliar a autenticidade dos códigos-fontes desenvolvidos por alunos em atividades de programação. As linguagens de programação suportadas para a sua análise de plágio são: *C, C++, Java, C#, Python, Visual Basic, Javascript, FORTRAN, ML, Haskell, Lisp, Scheme, Pascal, Modula2, Ada, Perl, TCL, Matlab, VHDL, Verilog, Spice, MIPS assembly, a8086 assembly* e *HCL2* (MOSS, 2020).

O funcionamento do MOSS depende da técnica de *tokens* (FRANÇA, 2019). Os códigos-fontes a serem comparados são convertidos em *tokens* e com o uso do algoritmo *Winnowing* desevolvido por Schleimer, Wilkerson e Aiken (2003) cria-se assinaturas digitais ou *fingerprinting* para representar cada *token* previamente gerado. O cálculo da distância entre essas estruturas de representação indica o nível de similaridade entre os códigos-fontes comparados.

A complexidade de tempo de execução dessa ferramenta no pior caso é O(n^3) devido as manipulações e comparações a partir de estruturas *hashes* (DILI *et al*., 2011). Para a execução do MOSS é possível utilizar diferentes opções de configuração que buscam otimizar o manuseio e os próprios resultados da busca por plágio nos códigos-fontes a serem comparados. A Figura 15 mostra o resultado a partir de duas entradas implementadas em linguagem *C*, nomeadas: “exemploA.c” e “exemploB.c”.

Figura 15 - Modelo de resultado fornecido pelo MOSS


Moss Results

Sat Apr 11 12:44:00 PDT 2020

Options -l c -m 10

[How to Read the Results | Tips | FAQ | Contact | Submission Scripts | Credits]

File 1	File 2	Lines Matched
exemploA.c (30%)	exemploB.c (33%)	20

Any errors encountered during this query are listed below.

Fonte: Autoria própria

Para facilitar o seu uso, o MOSS permite o acesso aos códigos-fontes por meio de seus respectivos diretórios com a opção "*d*". Para otimizar os resultados, a ferramenta permite o uso de códigos-fontes bases por meio da opção "*b*". Quando são fornecidos indica que todos os seus conteúdos podem aparecer nos códigos-fontes a serem comparados e não será considerado plágio. No entanto, existe a opção "*m*" que delimita a quantidade máxima que um determinado trecho de código-fonte qualquer comparado pode aparecer até que seja considerado como sendo base.

2.3 CONSIDERAÇÕES FINAIS DO CAPÍTULO

Este capítulo apresentou a fundamentação teórica necessária para a compreensão da abordagem criada no Capítulo 4. Para isso, apresentou a bioinformática levando em consideração que parte do modelo da abordagem utiliza conceitos, técnicas e ferramentas dessa área de estudo. Além disso, introduziu o campo de estudo do plágio em programação com: técnicas de detecção, níveis de plágio e ferramentas de detecção de plágio em códigos-fontes como: JPLAG e MOSS.

Com relação a bioinformática, explicou os conceitos de sequência de DNA, modelo de DNA sintético, alinhamento de sequências, e a execução do alinhamento com a ferramenta BLAST. Tanto os conceitos, a técnica de alinhamento e a BLAST são utilizados em um método baseado em bioinformática desenvolvido por Pedersen (2012) que serviu como modelo na elaboração da abordagem deste trabalho.

As técnicas da computação utilizadas na solução do problema de detecção de plágio em programação foram introduzidas e a técnica de token foi explicada em maiores detalhes por ser utilizada de forma combinada com a abordagem que é baseada em bioinformática. A combinação de uma ou mais técnicas, dentre outras

características presentes na abordagem criada são fundamentadas no mapeamento sistemático descrito no próximo capítulo.

3 ESTADO DA ARTE

Ao considerar o domínio de estudo associado ao plágio de código-fonte, segundo (NOVAK, 2016), percebe-se a existência de diferentes soluções propostas como em (DONALDSON; LANCASTER; SPOSATO, 1981; JOY; LUCK, 1999; LEACH, 1995; OTTENSTEIN, 1976; SCHLEIMER; WILKERSON; AIKEN, 2003; WISE, 1996). Por meio de um mapeamento sistemático foi identificado qual o cenário atual de soluções propostas neste campo de estudo. O objetivo de um estudo de mapeamento de modo sistemático é reconhecer e categorizar as principais técnicas, modelos e ferramentas apresentados até o momento para este domínio. A partir de uma análise dos resultados encontrados, foi possível identificar ideias de pesquisas futuras a serem realizadas (MUSHTAQ; RASOOL; SHEHZAD, 2017).

Com o intuito de levantar o cenário atual de 2013 a 2018 da área de aplicação deste trabalho, um mapeamento sistemático é proposto baseado na adaptação do modelo de (RATTAN; BHATIA; SINGH, 2013). Neste modelo, os autores fundamentaram as suas etapas no guia de revisão sistemática de literatura definido por (KITCHENHAM *et al.*, 2009; BRERETON *et al.*, 2007). A área de abrangência do estudo foi a de detecção de clonagem de software. Como é uma área correlata ao presente trabalho, utilizou-se este como parâmetro para a elaboração do mapeamento proposto na área de plágio de código-fonte.

A elaboração do mapeamento sistemático proposto foi realizada a partir de 8 etapas: planejamento do mapeamento, questões de pesquisa, bases de informação, *string* de busca, critérios de seleção, avaliação da qualidade, extração de dados e resultados. A seção 3.1 explica o método adotado de modo a mostrar quais adaptações foram definidas para o modelo de referência escolhido. A seção 3.2 apresenta a aplicação do método de mapeamento adotado. Ao final do capítulo, a seção 3.3 apresenta as considerações finais a respeito do mapeamento realizado.

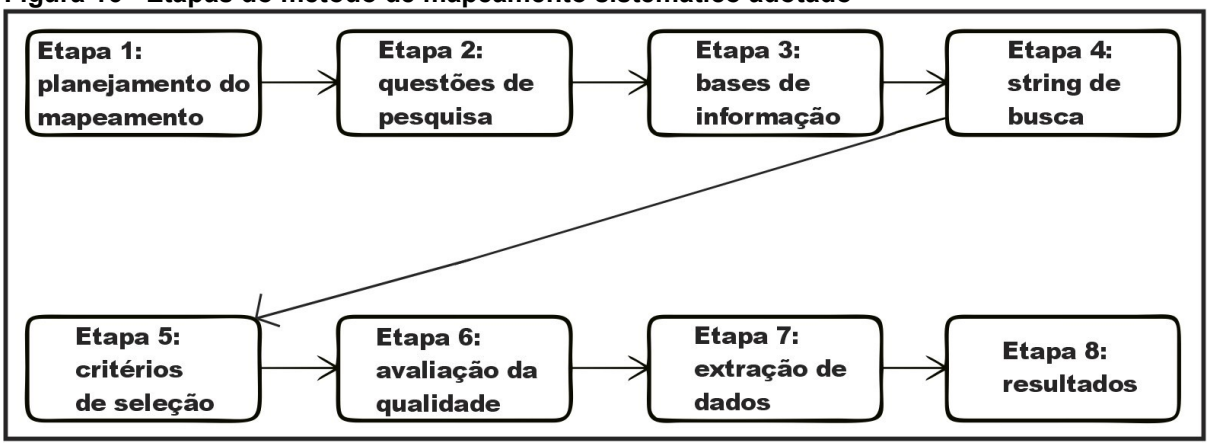
3.1 MÉTODO DE MAPEAMENTO SISTEMÁTICA ADOTADO

O método adotado apresenta todas as etapas definidas no modelo de (RATTAN; BHATIA; SINGH, 2013). No entanto, as seguintes etapas foram conduzidas com modificações: planejamento do mapeamento, critérios de seleção, avaliação da

qualidade, e extração de dados. As etapas que permaneceram sem alterações são: questões de pesquisa, bases de informação e *string* de busca. A ordem e a própria definição de cada etapa são ilustradas pela Figura 16.

A etapa de planejamento do mapeamento foi modificada tendo em vista que era destinada a revisão sistemática e não mapeamentos sistemáticos. No planejamento foi definido o protocolo de condução do estudo assim como todas as etapas a serem realizadas. Por exemplo, as questões de pesquisas devem ser criadas respeitando motivações associadas ao trabalho. Além da definição de cada fase, o processo de revisão de cada uma foi definido para ser conduzido por mais de um pesquisador em apenas algumas etapas específicas: questões de pesquisa, bases de dados, e *string* de busca.

Figura 16 - Etapas do método de mapeamento sistemático adotado



Fonte: Autoria própria

As questões de pesquisa são definidas a partir da motivação do estudo do mapeamento e refletem as indagações que guiam o objetivo geral estabelecido para o trabalho. Qualquer informação extraída durante o mapeamento deve estar relacionada a uma das questões levantadas. Cada pergunta é elaborada para considerar uma ou mais motivação específica.

Na etapa de identificação de bases de informação são escolhidas as fontes de dados utilizadas na seleção de trabalhos relevantes para responder as questões de pesquisa. Para que um estudo de mapeamento identifique relevantes trabalhos da área em questão estudada, deve-se utilizar bases de dados apropriadas no momento da pesquisa (BRERETON *et al.*, 2007). Estas bases de informação escolhidas são as fontes de todos os estudos identificados na execução do procedimento de seleção de trabalhos, que acontece nas próximas etapas. Também, é possível utilizar bases

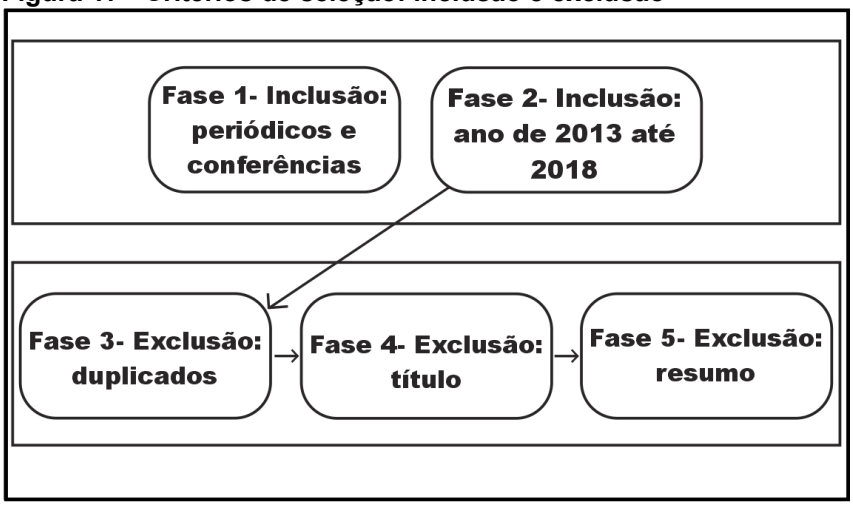
adicionais para abranger trabalhos importantes para o estudo se as bases definidas não contemplarem como esperado. De modo similar, o modelo de referência seguido neste mapeamento utilizou bases adicionais.

A elaboração de uma *string* de busca é uma etapa que depende das bases de informação escolhidas. A *string* de busca é formada por um conjunto de palavras-chaves, que abrangem o título e o resumo dos principais trabalhos relacionados às questões de pesquisas definidas na etapa 2. Estes trabalhos relacionados são procurados nas bases de informação definidas na etapa 3.

O conjunto de palavras-chaves pode variar de acordo com a base de informação escolhida, assim pode haver conjuntos específicos para cada base de dados utilizada ou até mesmo utilizar as mesmas palavras-chaves em cada base.

A etapa 5, referente aos critérios de seleção, é definida de uma forma diferente em relação ao que foi proposto no modelo de referência, pois foram definidos diferentes critérios de inclusão e exclusão. A Figura 17 apresenta quais passos foram considerados na seleção e quais critérios adotados para a filtragem dos trabalhos relevantes a partir do resultado da aplicação da *string* de busca em uma das bases de informação.

Figura 17 - Critérios de seleção: inclusão e exclusão



Fonte: Autoria própria

A Figura 17 ilustra a sequência de fases de filtragem no processo de seleção, que é composto de 5 (cinco) fases divididas em 2 (dois) tipos de critérios. Os tipos de critérios são: inclusão e exclusão. As etapas 1 e 2 pertencem a categoria de inclusão. A fase 1 define que apenas trabalhos publicados em conferências e periódicos podem

ser considerados. A fase seguinte, fase 2, verifica o ano em que os trabalhos foram publicados. Neste caso, apenas trabalhos de 2013 até 2018 podem ser aceitos.

As fases 3, 4 e 5 pertencem a categoria de exclusão. A fase 3 faz a exclusão de trabalhos duplicados devido a possibilidade de existirem em mais de uma base de dados. Na fase 4, é feita uma filtragem com base na leitura do título de cada artigo. A última fase realiza a exclusão dos trabalhos que não possuem um resumo compatível com o tema em questão.

Todas as fases apresentam um fluxo linear, em que cada uma começa mediante ao término da anterior. O resultado da filtragem proporcionada por cada fase é acumulativo, ou seja, cada fase trabalha com o resultado encontrado na sua etapa anterior.

Na avaliação de qualidade, referente a etapa 6 do modelo adotado, busca-se analisar a qualidade dos resultados encontrados a partir da aplicação de todas as fases de filtragem da etapa 5. A qualidade atribuída aos resultados é fundamentada na relevância de cada trabalho identificado no resultado. De acordo com (PAGANI; KOVALESKI; RESENDE, 2015), um dos possíveis critérios para avaliar a relevância de um artigo científico é quantidade de citação do mesmo. Diferente do método de mapeamento seguido, a etapa de avaliação de qualidade foi adaptada para classificar os trabalhos encontrados por ordem decrescente de citação. A quantidade de citação é de acordo com os dados do *Google Scholar* (GOOGLE, 2019).

A penúltima etapa, de extração de dados, é importante para orientar a análise de cada artigo contido nos resultados encontrados. Cada artigo encontrado após as filtragens deve passar por uma análise a fim de identificar como ele se relaciona com as questões de perguntas elaboradas na etapa 2. Diferente do modelo de referência seguido, a análise de cada trabalho não passou por uma etapa de verificação por mais de um pesquisador.

Por fim, a etapa de resultados apresenta todas as respostas e análises pertinentes a cada questão de pesquisa levantada na etapa 2. Todas as perguntas devem ser respondidas e informações devem ser inferidas com base nas extrações de dados da sétima etapa.

3.2 APLICAÇÃO DO MODELO DE MAPEAMENTO SISTEMÁTICO ADOTADO

A aplicação do método de mapeamento sistemático é apresentada nesta seção de modo a mostrar os resultados das etapas contempladas. As etapas foram executadas sequencialmente respeitando a ordem proposta, ilustrada pela Figura 16.

3.2.1 Etapa 1: Planejamento do Mapeamento

O mapeamento realizado teve participação de mais de um pesquisador para realizar a revisão das etapas iniciais: 2, 3 e 4. Os critérios de inclusão e exclusão foram planejados para serem executados imediatamente após a etapa de *string* de busca. Após todas as fases da etapa de seleção serem executadas, foi realizada uma classificação dos artigos filtrados por quantidade de citações para identificar os principais artigos mapeados neste estudo.

3.2.2 Etapa 2: Questões de Pesquisa

As questões de pesquisa contribuíram para a identificação do cenário atual da área de plágio em código-fonte e foram elaboradas a partir de motivações específicas tais como: identificar quais os tipos de técnicas empregadas nas soluções, avaliar como as soluções são validadas e quais os tipos de linguagens de programação são suportadas. O Quadro 3 explica o motivo de criação de cada pergunta ao todo foram criadas 3 perguntas de pesquisa.

As perguntas elaboradas foram as seguintes: 1) quais são as abordagens utilizadas na identificação de plágio em código-fonte; 2) quais abordagens de identificação de plágio foram avaliadas por meio de conjuntos de testes e quantos testes foram conduzidos; e 3) quais as linguagens de programação suportadas nas abordagens de identificação de plágio.

Quadro 3 - Elaboração das questões de pesquisa

Questões de Pesquisa	Descrição
1. Quais são as abordagens utilizadas na identificação de plágio em código-fonte?	A principal motivação é identificar qual é a sistemática utilizada na identificação de plágio em código-fonte. De modo a identificar os métodos ou técnicas ou metodologias mais utilizados e se ocorre a combinação de mais de uma abordagem para resolver o problema de identificação de plágio em código-fonte.

2. Quais abordagens de identificação de plágio foram avaliadas por meio de conjuntos de testes e quantos testes foram conduzidos?	Para identificar a aplicabilidade de uma solução é importante avaliar por meio de testes se os resultados esperados são similares aos obtidos. Também, identificar o nível de avaliação das abordagens para sugerir qual é a quantidade ideal de testes que uma abordagem neste domínio precisa apresentar.
3. Quais são as linguagens de programação suportadas nas abordagens de identificação de plágio?	Avaliar para quais linguagens de programação as abordagens existentes dão suporte. Se existem soluções capazes de contemplar qualquer linguagem de programação. Quais as linguagens mais suportadas pelas abordagens de identificação de plágio.

Fonte: Autoria própria

Neste estudo de mapeamento é feita a investigação de soluções gerais propostas para o domínio de plágio em código-fonte. Também, busca-se identificar estudos relacionados envolvendo sequências biológicas.

3.2.3 Etapa 3: Bases de informação

A condução deste mapeamento sistemático foi feita a partir dos principais mecanismos de busca acadêmica. De acordo com (BUCHINGER; CAVALCANTI; HOUNSELL, 2014), uma análise qualitativa identificou as 7 melhores bases de informação para buscas de conteúdos científicos na computação. Dentre essas melhores bases identificadas, considerou-se aquelas presentes no modelo de Rattan, Bhatia e Singh (2013): IEEE, Xplore e ACM.

3.2.4 Etapa 4: *String* de busca

Para identificar os trabalhos que podem responder as questões de pesquisa é utilizada uma combinação de palavras-chaves nas bases de informação. O Quadro 4 identifica quais palavras-chaves formam a *string* utilizada em cada mecanismo de busca no mapeamento realizado. Além disto, neste mesmo quadro é possível identificar quais componentes booleanos são empregados na busca. Todas as bases de informação escolhidas empregaram a mesma estrutura de busca definida.

Quadro 4 - Palavras-chaves utilizadas em cada base de informação

***String* de busca**	**Base de Informação**
1. ((Source*) OR (Software*) OR (Pro-gram*)) AND (Plagiari*)	IEEE Xplore e ACM

Fonte: Autoria própria

A escolha de cada termo da *string* de busca foi feita para resultar em buscas mais inclusivas do que exclusivas, com o objetivo de identificar diferentes abordagens no campo de estudo da identificação de plágio em códigos-fontes. Desta forma, contribui-se diretamente e indiretamente para a busca por respostas frente as questões de pesquisa. No entanto, uma restrição no idioma foi colocada em pauta para viabilizar a busca por resultados nas bases de informação. Como as bases escolhidas utilizam a língua inglesa, as palavras-chaves foram elaboradas em inglês.

3.2.5 Etapa 5: Critérios de Seleção

Ao executar a etapa 4, foram identificados 97 trabalhos relacionados com a *string* de busca nas bases de dados escolhidas. Este resultado é gerado sem utilizar nenhum critério de seleção. Nesta etapa de critérios de seleção é aplicada as 5 fases de filtragem e os resultados obtidos pela seleção são apresentadas no Quadro 5.

Quadro 5 - Resultado da etapa de seleção após a fase de filtragem

Fase de filtragem	**Quantidade total de artigos por fase de seleção**	**Quantidade de artigos reprovados**	**Quantidade de artigos aprovados**
Fase 1 – Inclusão: periódicos e conferências	97	0	97 (Todos: periódicos ou conferências)
Fase 2 – Inclusão: ano de 2013 até 2018	97	50 (artigos com data fora do intervalo)	47
Fase 3 – Exclusão: duplicados	47	0	47 (Nenhum duplicado)
Fase 4 – Exclusão: título	47	9 (artigos eliminados devido ao título)	39
Fase 5 – Exclusão: resumo	39	6 (artigos eliminados devido ao resumo)	33

Fonte: Autoria própria

Após a aplicação de todos os critérios de inclusão e exclusão, a quantidade de artigos foi reduzida de 97 para 33. A partir deste resultado filtrado foi feita a aplicação da próxima etapa.

3.2.6 Etapa 6: Avaliação da Qualidade

Com o intuito de identificar os trabalhos mais relevantes no mapeamento sistemático, utilizou-se uma avaliação da qualidade por ordem de importância conforme a quantidade de citações que o estudo possui. O Quadro 6 apresenta a classificação geral dos artigos.

Quadro 6 - Classificação final dos artigos

Classificação	Autor	Ano	Citação
1º	(CHAN; HUI; YIU, 2013)	2017	32
2º	(TIAN *et al*., 2015)	2015	28
3º	(ZHANG *et al*., 2014)	2014	26
4º	(SOH *et al*., 2015)	2015	25
5º	(JHI *et al*., 2015)	2015	22
6º	(LUO *et al*., 2017)	2017	19
7º	(TIAN *et al*., 2013)	2013	18
8º	(MING *et al*., 2016)	2016	16
9º	(KARNALIM, 2017a)	2017	16
10º	(LAZAR; BANIAS, 2014)	2014	14
11º	(ACAMPORA; COSMA, 2015)	2015	9
12º	(AJMAL *et al*., 2014)	2014	9
13º	(KIKUCHI *et al*., 2014)	2014	7
14º	(KIM *et al*., 2013)	2013	6
15º	(SHARMA; SHARMA; TYAGI, 2015)	2015	6
16º	(ZHANG; LIU, 2013)	2013	5
17º	(POHUBA; DULIK; JANKU, 2014)	2014	5
18º	(KARNALIM, 2018)	2018	5
19º	(CHOI *et al.*, 2013)	2013	4
20º	(BABY *et al.*, 2014)	2014	3
21º	(LIU; ZHENG; WEI, 2014)	2014	2
22º	(MIŠIĆ; PROTIĆ; TOMAŠEVIĆ, 2017)	2017	1
23º	(WANG; ZHON; ZHANG, 2015)	2015	1
24º	(SUDHAMANI; RANGARAJAN, 2017)	2017	1
25º	(MIRZA; JOY; COSMA, 2017)	2017	1

26º	(STRILEȚCHI *et al.*, 2016)	2016	1
27º	(JAIN *et al.*, 2017)	2017	1
28º	(OPRIŞA; IGNAT, 2015)	2015	0
29º	(AGRAWAL; SHARMA, 2017)	2017	0
30º	(SCHNEIDER *et al.*, 2017)	2017	0
31º	(DUTTA, 2015)	2015	0
32º	(ROOPAM; SINGH, 2018)	2018	0
33º	(KARGÉN; SHAHMEHRI, 2017)	2017	0

Fonte: Autoria própria

Com a classificação realizada, identifica-se que alguns estudos ainda não obtiveram citações. De uma forma geral, a taxa média de citação que cada trabalho possui nessa classificação é de 5 citações. No entanto, uma taxa de 76,32% das citações se concentram entre os dez primeiros trabalhos classificados.

3.2.7 Etapa 7: Extração de Dados

A extração dos dados relevantes de cada artigo científico foi proposta seguindo um modelo de ficha de leitura apresentado no Quadro 7. Neste modelo, buscou-se extrair sucintamente informações necessárias para responder as questões de pesquisa.

Quadro 7 – Exemplo de ficha de leitura para extração de dados

Tópico	Descrição
1. Título:	*Heap Graph Based Software Theft Detection*
2. Primeiro autor:	(CHAN; HUI; YIU, 2013)
3. Ano:	2017
4. Tipo de artigo:	*Journal*
5. Procedimento de teste:	200 *websites*
6. Linguagens suportadas:	*JavaScript*
7. Técnica utilizada:	*Birthmark* e grafos

Fonte: Autoria própria

O processo de leitura completa do artigo foi feito apenas para aqueles encontrados na classificação final de artigos. A leitura parcial ocorreu nas fases de exclusão 4 e 5 da etapa de critérios de seleção.

3.2.8 Etapa 8: Resultados

O mapeamento sistemático proposto apresentou três questões de pesquisas. A primeira questão de pesquisa busca avaliar as técnicas existentes e empregadas nas soluções de detecção de plágio em código-fonte. A segunda questão analisa como as soluções propostas estão validando as suas ideias e qual o conjunto de teste utilizado. Por fim, a última questão procura identificar quais linguagens de programação estão sendo atendidas pelas ferramentas de plágio em programação.

O Quadro 8 apresenta todos os resultados extraídos pelo mapeamento de modo a mostrar respostas para as questões de pesquisas. Para cada pesquisa identificada no resultado as seguintes informações foram extraídas: primeiro autor, abordagem, teste e linguagem. As últimas três informações respondem às perguntas: 1, 2 e 3 elaboradas na etapa 2, respectivamente.

Quadro 8 - Resultado do mapeamento sistemático realizado

Primeiro Autor	**Abordagem**	**Teste**	**Linguagem**
(MIŠIĆ; PROTIĆ; TOMAŠEVIĆ, 2017)	RKR-GST algoritmo; Programação paralela; Técnica baseada em *token*.	Coletou exemplos de códigos de alunos e criou uma cópia plagiada por meio de 20 tipos de modificações	C e C++
(TIAN *et al.*, 2015)	Técnica de *birthmark*	324 versões de 28 códigos-fontes implementados em C e Java	C e Java
(ZHANG *et al.*, 2014)	Método baseado em semântica; Método baseado em lógica.	168 pares de testes	Independente da linguagem (Não precisa do código-fonte)
(SOH *et al.*, 2015)	Análise de interface de usuário; Técnica de *birthmark*.	521 aplicativos *Android*	Parcialmente dependente de linguagem (Android apps)
(JHI *et al.*, 2015)	Método baseado em sequências; Método baseado em valores críticos.	34 tipos de ofuscação	Independente da linGuagem
(TIAN *et al.*, 2013)	Baseado em sequências; Método baseado em valores críticos.	34 versões de ofuscação	Independente da linguagem
(MING *et al.*, 2016)	Método baseado em semântica; Método baseado em lógica.	10 programas testados	Independente da linguagem
(LUO *et al.*, 2017)	Método baseado em semântica; Método baseado em sequências.	13 funções de um programa	Independente da linguagem
(LAZAR; BANIAS, 2014)	Árvore sintática abstrata; Método baseado em sequências.	10 códigos testatos	C

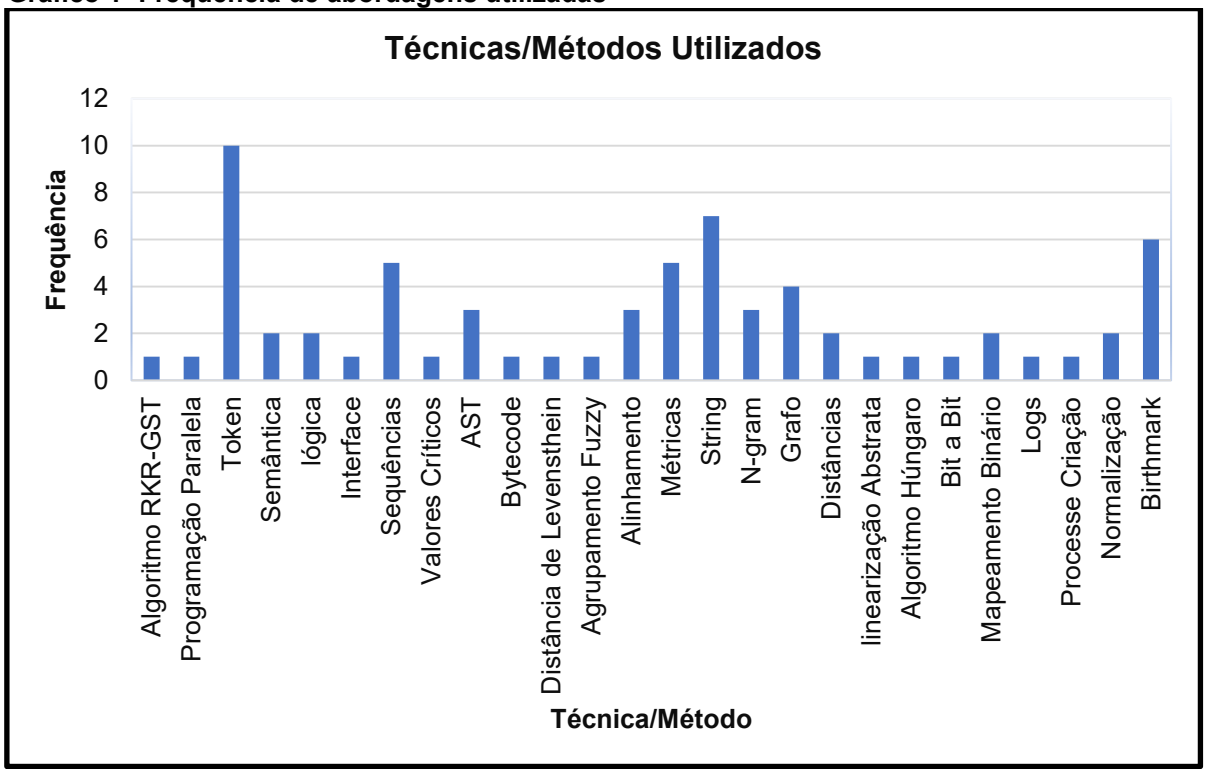
(KARNALIM, 2017a)	Técnica baseda em *bytecode*; Método da distância de *Levensthein*.	378 códigos-fontes plagiados	Java
(ACAMPORA; COSMA, 2015)	Abordagem baseada em agrupamento *Fuzzy*	40 cenários de testes	Independente da linguagem
(KIKUCHI *et al*., 2014)	Técnica de alinhamento; Técnica baseada em *token*; Árvore sintática abstrata.	4 testes de similaridade de códigos-fontes	C e C++
(AJMAL *et al.*, 2014)	Método beseado em métricas; Técnica baseada em *token*; Método baseado em *string*.	40 códigos-fontes	Java
(KIM *et al.*, 2013)	Técnica de *birthmark*; Método *N-gram*; Método baseado em *string*.	14 programas testados	Independente da linguagem
(ZHANG; LIU, 2013)	Árvore sintática abstrata; Alinhamento de sequências biológicas.	36 códigos escritos em linguagem C	C e Java
(POHUBA; DULIK; JANKU, 2014)	Técnica baseada em *token*; Método baseado em *string*.		Independente da linguagem
(SHARMA; SHARMA; TYAGI, 2015)	Técnica baseada em *token*; Método *N-gram*; Método baseado em *string*.	50 códigos-fontes com 75 linhas cada um	C, C++, Java, Perl, Python e Php
(CHOI *et al*., 2013)	Técnica de *birthmark*; Técnica baseada em grafo.	4 códigos-fontes testados no experimento	Independente da linguagem
(BABY *et al.*, 2014)	Método baseado em métricas; Técnicas de medidas de distâncias; Técnica baseada em *token*.	26 códigos-fontes escritos em C	C
(KARNALIM, 2018)	Método de linearização abstrata; Técnica baseada em *token*.	23 pares de códigos-fontes	Java
(WANG; ZHON; ZHANG, 2015)	Método baseado em sequências; Método baseado em *string*.	800 códigos selecionados	Python
(SUDHAMANI; RANGARAJAN, 2017)	Método baseado em métricas; Técnicas de medidas de distâncias.	24 programas e 103 variações	Independente da linguagem
(CHAN; HUI; YIU, 2013)	Técnica de birthmark; Técnica baseada em grafo.	200 *websites* avaliados	JavaScript
(MIRZA; JOY; COSMA, 2017)	Método baseado em métricas; Técnica baseada em *token*.	4 cenários com 250 arquivos java cada um. Arquivos retirados da base de dados BlackBox.	Java
(STRILEȚCHI *et al*., 2016)	Técnica baseada em *token*; Método baseado em *string*.	47040 códigos-fontes	C, C++, Java, Php e JavaScript
(OPRIŞA; IGNAT, 2015)	Método *N-gram*; Algoritmo Húngaro.	7000 métodos em java	Java
(AGRAWAL; SHARMA, 2017)	Técnica de normalização; Método baseado	15 códigos-fontes java	Java

	em *string*.		
(LIU; ZHENG; WEI, 2014)	Técnica de *birthmark*; Técnica baseada em grafo.	2 códigos de teste	Independente da linguagem
(JAIN *et al.*, 2017)	Método baseado *bit* a *bit*; Método de mapeamento binário.	12 diferentes códigos-fontes em C	C
(SCHNEIDER *et al.*, 2017)	Técnica baseada em arquivos *log*; Técnica ba- seada no processo de criação.	180 códigos-fontes	Independente da linguagem
(DUTTA, 2015)	Técnica baseada em *token*; Técnica de normalização.	3 cenários com 2 códigos-fontes cada	C
(ROOPAM; SINGH, 2018)	Técnica baseada em grafo; Método baseado em métricas.	4 diferentes repositórios de códigos em Java	Java
(KARGÉN; SHAHMEHRI, 2017);	Método baseado em alinhamento; Método de mapeamento binário.	Testado em 11 programas linux	Independente da linguagem

Fonte: Autoria própria

Diversas técnicas foram utilizadas ao analisar os 33 artigos. Como resposta para a questão de pesquisa 1 da etapa 2, foram 26 abordagens diferentes identificadas. O Gráfico 1 exibe a frequência de utilização de tais abordagens, nota-se que algumas técnicas são comumente utilizadas por diferentes trabalhos. A técnica de *token* é a mais utilizada entre todas identificadas. No entanto, mais de uma técnica apresentou o menor valor de frequência de utilização, por exemplo: o algoritmo Húngaro.

Gráfico 1- Frequência de abordagens utilizadas

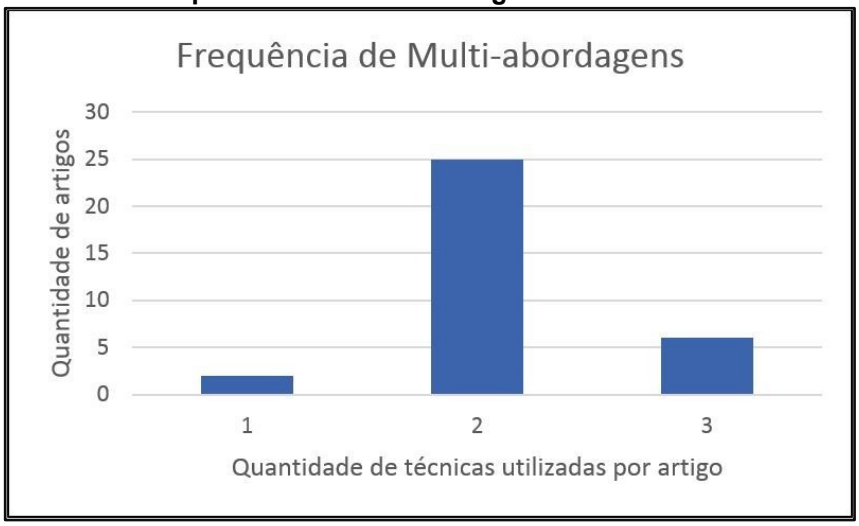


Fonte: Autoria própria

O Gráfico 2 por meio de um histograma identifica-se a frequência de utilização de combinação de diferentes técnicas. A partir do histograma visualiza-se que 25 artigos utilizaram pelo menos duas abordagens diferentes: (AGRAWAL; SHARMA, 2017; AJMAL *et al.*, 2014; BABY *et al.*, 2014; CHAN; HUI; YIU, 2013; CHOI *et al.*, 2013; DUTTA, 2015; JAIN *et al.*, 2017; JHI *et al.*, 2015; KARGÉN; SHAHMEHRI, 2017; KARNALIM, 2017a; KARNALIM, 2018; KIKUCHI *et al.*, 2014; KIM *et al.*, 2013; LAZAR; BANIAS, 2014; LIU; ZHENG; WEI, 2014; LUO *et al.*, 2017; MING *et al.*, 2016; MIRZA; JOY; COSMA, 2017; MIŠIĆ; PROTIĆ; TOMAŠEVIĆ, 2017; ZHANG; LIU, 2013; POHUBA; DULIK; JANKU, 2014). Enquanto apenas 2 artigos utilizaram uma técnica específica: abordagem baseada em agrupamento *Fuzzy* (ACAMPORA; COSMA, 2015) e técnica de *birthmark* (TIAN *et al*., 2015).

Com relação a resposta da questão de pesquisa 2 da etapa 2, a maioria dos trabalhos utilizou procedimentos de teste para validar os resultados encontrados. Apenas um trabalho utilizou uma abordagem sem especificar o procedimento de teste. O Gráfico 3 apresenta os intervalos das quantidades de códigos-fontes de testes que foram executados nos procedimentos de avaliação de resultados dos artigos.

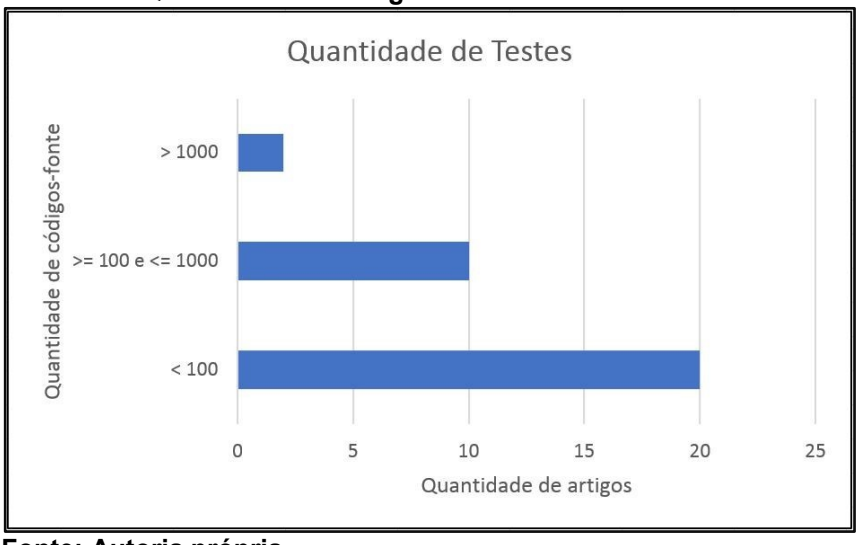
Gráfico 2 - Frequência de multi-abordagens identificadas



Fonte: Autoria própria

Os procedimentos de testes apresentaram comumente menos de 100 códigos-fontes de testes para avaliar a abordagem proposta. A segunda maior frequência de intervalo foi a utilização entre 100 e 1000 códigos-fontes por trabalho, e o procedimento menos frequente foi utilizar mais do que 1000 em testes na avaliação da abordagem. Apenas o trabalho de (POHUBA; DULIK; JANKU, 2014) não apresentou algum procedimento de avaliação ou validação por meio de testes.

Gráfico 3 - Quantidade de códigos-fontes utilizados em testes



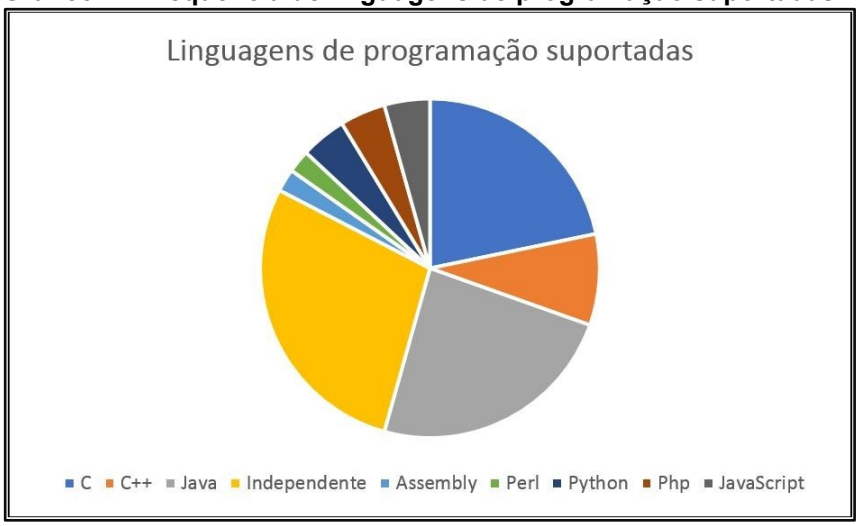
Fonte: Autoria própria

Nota-se que não é comum a utilização de bases de dados de códigos-fonte compartilhadas entre as pesquisas para testes, apenas a pesquisa de (MIRZA; JOY; COSMA, 2017) fez uso de uma base de dados independente na validação. O restante dos trabalhos utiliza códigos-fonte desenvolvidos por alunos em disciplinas de

laboratórios de programação e criam os seus próprios códigos-fonte de forma manual ou automática.

As linguagens de programação suportadas pelas abordagens ou técnicas propostas pelos trabalhos identificados variaram, como é mostrado no Gráfico 4. No entanto, a maioria dos trabalhos, exatamente 13 dos 33 artigos, estão criando soluções que são independentes da linguagem de programação. O que mostra uma tendência entre as pesquisas mais relevantes na área de identificação de plágio de programação.

Gráfico 4 - Frequência de linguagens de programação suportadas



Fonte: Autoria própria

Acerca das abordagens que foram dependentes da linguagem de programação, destaca-se o uso de *Java* contabilizando 11 de 33 trabalhos, logo em seguida é a linguagem *C* que é suportada em 10 dos 33 trabalhos. Identifica-se que esses tipos de abordagens dependentes tendem a oferecer suporte a mais de uma linguagem.

3.3 CONSIDERAÇÕES FINAIS DO CAPÍTULO

O presente capítulo realizou um mapeamento sistemático seguindo um modelo proposto fundamentado no trabalho de (RATTAN; BHATIA; SINGH, 2013). Nesse modelo foram aplicadas 8 etapas para a geração de resultados referentes a um estudo do cenário atual das pesquisas sobre plágio em programação. Tais

resultados apresentam informações a respeito das principais técnicas, modelos e ferramentas desenvolvidas nessa área, dentre outras.

Em suma, aproximadamente 94% dos 33 artigos contemplados neste mapeamento utilizam uma combinação de técnicas para propor uma solução neste domínio. Dentre as diversas técnicas utilizadas, a mais usada é a baseada em *tokens*. A maioria dos estudos desenvolvem abordagens com suporte a mais de uma linguagem de programação diferente. Inclusive, algumas propostas são independentes da linguagem de programação que o programa ou código-fonte é criado.

Apenas o estudo de Pohuba, Dulik e Janku (2014) não especificou o procedimento de teste para avaliar o desempenho da solução que foi proposta. Desta forma, mostra-se que é fundamental a avaliação da solução por meio de procedimentos de testes, a maioria dos artigos apresentou avaliações com até 100 diferentes códigos-fonte.

Com a aplicação do mapeamento sistemático foi possível identificar a oportunidade de desenvolver a combinação da técnica de *token* com técnicas da bioinformática, uma vez que ambas não são utilizadas de forma combinada entre os trabalhos analisados, apenas são aplicadas com outras técnicas ou de forma isolada. Inclusive, um dos trabalhos utiliza *token* com alinhamento, mas não considera uma abordagem de bioinformática. Sendo assim, o presente trabalho criou uma abordagem de detecção de plágio em código-fonte utilizando bioinformática no próximo capítulo.

4 BIOPLAG: UMA ABORDAGEM DE DETECÇÃO DE PLÁGIO EM CÓDIGO-FONTE

Por meio do mapeamento sistemático realizado no Capítulo 3, percebe-se a necessidade das soluções propostas em serem validadas com testes, utilizarem suporte a diferentes linguagens de programação e a tendência das soluções em utilizarem mais de uma técnica para aprimorar a capacidade de detecção de modificações em códigos-fonte. As validações aplicadas nas soluções propostas objetivam demonstrar os seus respectivos desempenhos na detecção de plágio em programação.

Conforme a fundamentação teórica do Capítulo 2, o código-fonte pode sofrer diferentes tipos de modificações. Nesse domínio de estudo, o desempenho pode ser representado pela capacidade em lidar com diferentes níveis de plágio em códigos-fonte. Também, a eficiência é considerada para obter soluções viáveis em termos de tempo de execução em relação a quantidade de processamento exigido.

Este capítulo cria uma abordagem de detecção de plágio capaz de lidar com diferentes linguagens de programação, mas neste trabalho a abordagem foi avaliada apenas para códigos-fonte em linguagem C.

A abordagem criada, denominada de BIOPLAG, fundamenta-se na combinação da técnica de computação de *tokens* com a utilização de conceitos, técnicas e ferramentas da bioinformática. Essa combinação pretende possibilitar a identificação dos 6 (seis) níveis de Faidhi e Robinson (1987) de plágio em códigos-fonte. Para explicar o seu funcionamento, a Seção 4.1 mostra uma visão geral da BIOPLAG e as seções 4.2, 4.3, 4.4 e 4.5 abordam os seus detalhes. A seção 4.2 apresenta as considerações finais do capítulo.

4.1 VISÃO GERAL

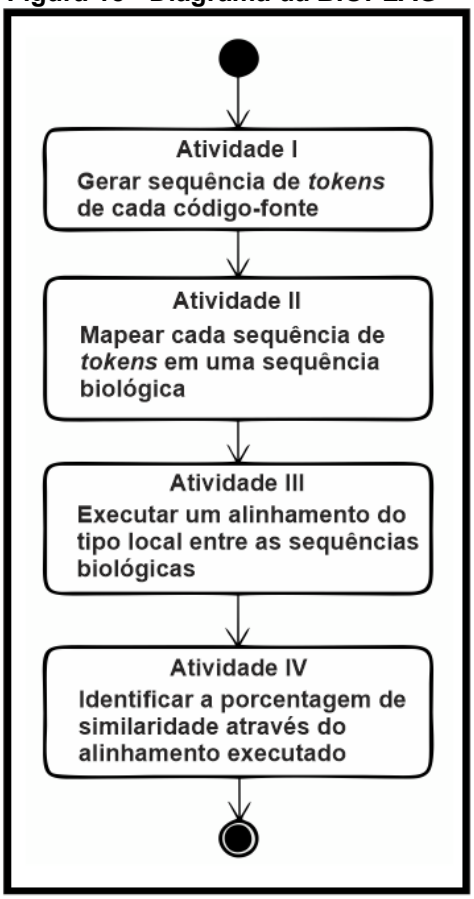
A BIOPLAG utiliza conceitos da bioinformática e da computação com o objetivo de detectar diferentes tipos de plágio em código-fonte. Esta abordagem busca ser capaz de identificar diferentes níveis de plágio em programação, seguindo a categorização proposta em 6 níveis por (FAIDHI; ROBINSON, 1987).

O funcionamento da solução proposta depende de três diferentes técnicas tais como: geração de *tokens*, mapeamento em sequências biológicas sintéticas e

alinhamento local de sequências biológicas. A aplicação da bioinformática na abordagem é baseada no trabalho de (PEDERSEN *et al.*, 2012), conforme apresentado no Capítulo 2. A técnica de computação utilizada é a geração de *tokens*, da qual é comumente aplicada em diferentes domínios, inclusive na área de detecção de plágio em programação.

Uma visão geral da forma como as técnicas são utilizadas é apresentada por meio de um diagrama de atividades ilustrado pela Figura 18. As principais atividades da abordagem são: geração de sequência de *tokens* para cada código-fonte de entrada, mapeamento de cada sequência de *tokens* em suas respectivas sequências biológicas sintéticas, execução de alinhamento local entre as sequências biológicas sintéticas, e a identificação da taxa de similaridade entre as sequências a partir do resultado do alinhamento.

Figura 18 - Diagrama da BIOPLAG



Fonte: Autoria própria

A atividade I é responsável por receber como entrada os códigos de programação a serem analisados, os quais são convertidos em sequências de *tokens*. Cada sequência gerada representa um determinado código-fonte, e a partir delas

possibilita a execução da etapa seguinte referente ao mapeamento em sequência biológica sintética.

O mapeamento de sequências de *tokens* em sequências biológicas sintéticas é feita na atividade II. A conversão em estruturas biológicas é necessária para possibilitar a utilização de ferramentas de bioinformática de alinhamento na atividade seguinte. Essa conversão é feita por um algoritmo gerador de sequência biológica que mapeia caracteres ASCII em unidades biológicas que dependem do tipo de sequência adotada.

A execução de um alinhamento local de sequências é realizada na atividade III e seu objetivo é identificar similaridades presentes entre duas ou mais estruturas de sequência biológica. A partir das regiões identificadas, é possível calcular uma taxa de similaridade entre os códigos-fontes.

A última atividade, representada pela atividade IV, realiza a geração de uma porcentagem de similaridade presente entre as sequências comparadas. Como cada sequência é uma representação de um código de programação, o valor calculado demonstra o quanto códigos-fontes são iguais ou diferentes. As seções a seguir apresentam detalhes do funcionamento de cada uma das atividades.

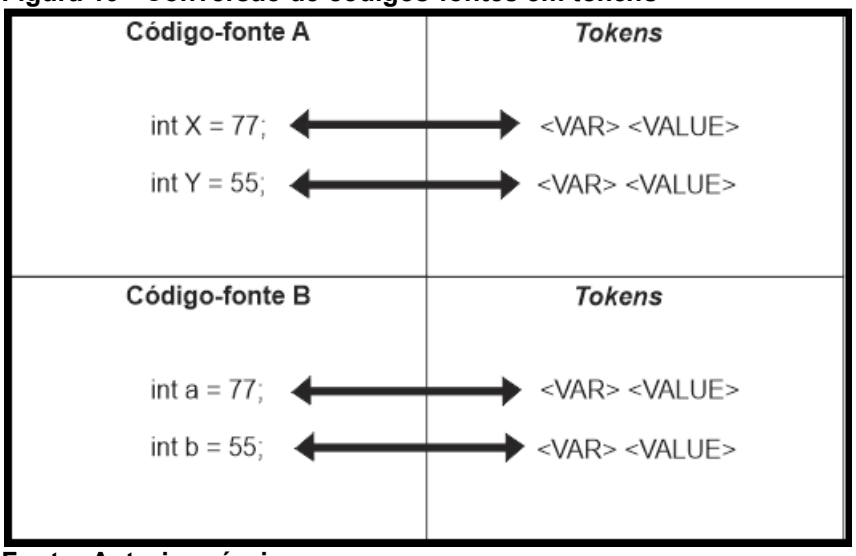
4.2 GERAÇÃO DE *TOKENS*

De acordo com (MOZGOVOY; KARAKOVSKIYZ; KLYUEV, 2008), os sistemas modernos de detecção de plágio em programação utilizam técnicas de *tokens*, devido a sua eficiência em analisar diversos tipos ou níveis de plágio. A tendência de os sistemas utilizarem essa técnica foi apresentada no mapeamento sistemático do capítulo 3. Por meio do mapeamento sistemático foi identificado que a técnica com maior frequência de utilização é a de *tokens*, dentre as pesquisas desenvolvidas no período entre o ano de 2013 até 2018.

A principal vantagem de utilizar essa técnica é a possibilidade de representação de elementos de um código-fonte por meio de unidades de informação. Conforme explicado no Capítulo 2, cada unidade representa uma categoria de elementos dos códigos de programação, por exemplo: as variáveis podem pertencer a um determinado grupo nomeado VAR, os valores das variáveis pertencem a um outro grupo que pode ser nomeado como VALUE. Sendo assim, considerando esse exemplo, os grupos: VAR e VALUE são diferentes *tokens*.

A partir do exemplo da Figura 19, percebe-se a característica da técnica em identificar elementos como as variáveis e seus valores de forma generalizada. Não importa se uma variável é declarada como "X" ou como "a" pois ambas são identificadas como VAR. Esse comportamento permite superar diversas ações de plagiadores a fim de modificar um código original. Alguns exemplos de ataques utilizados pelos plagiadores são: mudar a indentação, nome de variáveis, valores literais, comentários, estruturas de repetição, palavras-chave, dentre outros elementos.

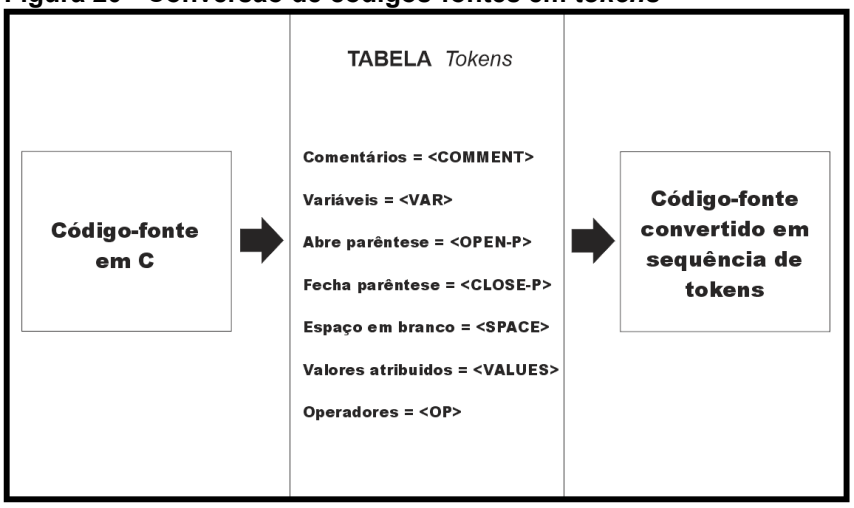
Figura 19 - Conversão de códigos-fontes em *tokens*



Fonte: Autoria própria

O funcionamento da atividade de geração de *tokens* na abordagem criada é apresentado pela Figura 20. A entrada de dados desse processo é um código-fonte escrito em uma determinada linguagem de programação suportada. Após ter acesso a entrada, realiza-se um mapeamento dos elementos pertencentes aos códigos fornecidos em sequências de *tokens* por meio de uma tabela ou regras definidas por meio de um analisador léxico. A saída do processo é a conversão do código-fonte em uma sequência de *tokens*.

Figura 20 - Conversão de códigos-fontes em *tokens*



Fonte: Autoria própria

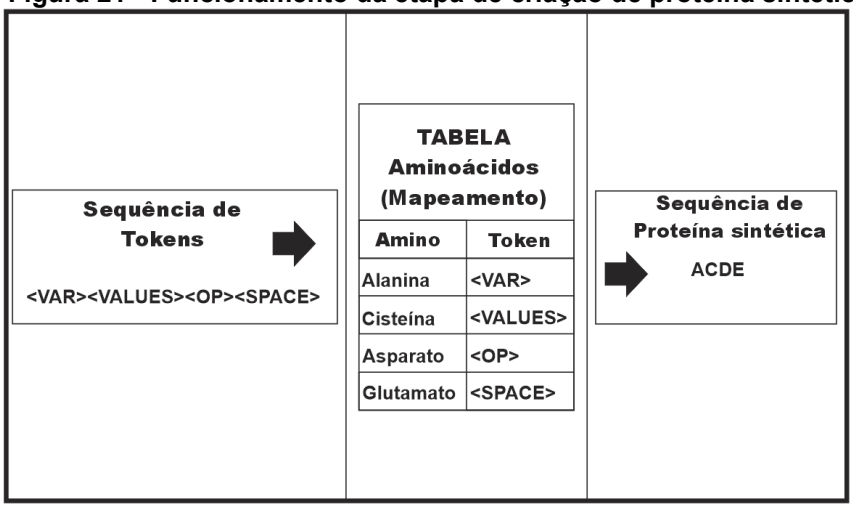
Neste trabalho, a linguagem C de programação é a suportada, no entanto, outras linguagens podem ser incorporadas nessa abordagem. A incorporação de novas linguagens depende apenas de um analisador léxico específico para tal linguagem a ser suportada. A importância de um detector de plágio ser multilinguagem é ressaltada por (ARWIN; TAHAGHOGHI, 2006). Essa característica permite abranger um tipo especial de plágio em que é feita uma transcrição de um código-fonte de uma linguagem para outra.

4.3 MAPEAMENTO EM SEQUÊNCIA BIOLÓGICA

A atividade II da BIOPLAG realiza o mapeamento das sequências de *tokens* em sequências biológicas. Conforme apresentado no capítulo 2, os artefatos computacionais podem ser mapeados em diferentes tipos de sequências. Nesta abordagem, os artefatos computacionais de interesse são os códigos-fontes e o tipo de sequência adotada para representá-los são as proteínas.

O mapeamento funciona a partir da conversão direta de cada *token* em um dos aminoácidos que compõem a estrutura linear de uma sequência de proteína. Para ilustrar como funciona esse processo, a Figura 21 apresenta um exemplo de conversão considerando sequências contendo 4 *tokens* diferentes e os seguintes aminoácidos: alanina ("*Ala*" ou "*A*"), cisteína ("*Cys*" ou "*C*"), asparato ("*Asp*" ou "*D*") e glutamato ("*Glu*" ou "*E*").

Figura 21 - Funcionamento da etapa de criação de proteína sintética



Fonte: Autoria própria

No exemplo da Figura 21, um determinado código-fonte é convertido em uma sequência de *tokens* definidos de acordo com a Figura 20. Por meio de uma tabela de mapeamento em aminoácidos, definiu-se que a alanina seria representada pelo *token* " *VAR*", cisteína pelo " *VALUES*", asparato pelo " *OP*" e glutamato pelo " *SPACE*". A partir da seguinte sequência como entrada: " VAR VALUES OP SPACE", gerou-se como saída desse processo a sequência de proteína sintética: "*ACDE*".

O funcionamento dessa forma de mapeamento tem como requisito a utilização de sequências biológicas que possuam uma quantidade de unidades de composição igual ou superior a quantidade de *tokens* diferentes gerados pela atividade anterior. Por exemplo, se utilizar um gerador capaz de produzir até 5 *tokens* diferentes na atividade anterior, não será possível mapear em uma sequência como a de DNA que pode conter apenas até 4 unidades diferentes em sua estrutura: adenina, guanina, timina e citosina.

Para viabilizar a utilização de qualquer sequência biológica de interesse nessas situações de não atender ao requisito de tamanho, pode-se mapear a representação binária em ASCII dos nomes dos *tokens* em uma das possíveis unidades da sequência escolhida. A conversão passa a ser por meio de *bits* contíguos que representam um determinado caractere do nome de um *token*.

Essa forma alternativa de mapeamento é ilustrada pela Figura 22 utilizando como exemplo o uso de sequência de DNA sintético. A entrada é uma sequência de *tokens* e cada um possui uma nomeação diferente. Considerando seus respectivos nomes, é feita uma consulta na tabela ASCII completa (incluída a versão estendida)

para identificar a codificação binária de cada caractere que pertença a formulação da palavra ou nome de identificação.

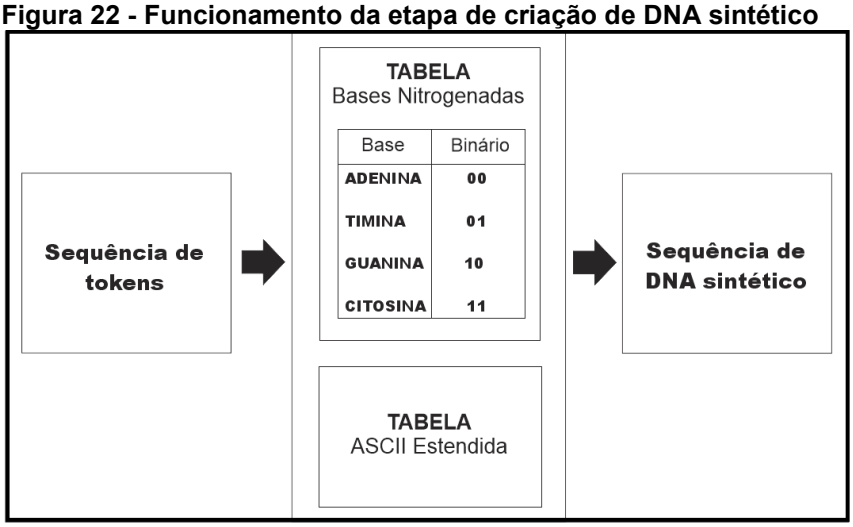
Por exemplo, o VAR possui três caracteres: "V", "A" e "R". Após identificar o código binário de VAR: 010101100100000101010010, é feita uma conversão de seus *bits* contíguos em bases nitrogenadas seguindo uma tabela de mapeamento, como é demonstrado no método de Pedersen (PEDERSEN *et al*., 2012). O critério de conversão é baseado na quantidade de *bits* necessários para representar 4 informações diferentes: adenina, citosina, guanina e timina.

A quantidade de *bits* pode ser calculada conforme a Fórmula 1, onde *N* é quantidade de possibilidades ou informações diferentes a serem representadas. Nesse contexto adotando o valor de *N* como 4, o resultado adotado como *T* indica que a quantidade mínima necessária é de 2 bits para representar as bases nitrogenadas.

$$T = \log_2 N \tag{1}$$

Por meio da tabela de bases nitrogenadas, ilustrada na Figura 22, uma das possíveis combinações de 2 *bits* foi utilizada. Os bits contíguos "00" são mapeados em adenina (*A*), os bits "01" em timina (*T*), os bits "10" em guanina (*G*), e os bits "11" em citosina (*C*). A saída é uma sequência com as bases nitrogenadas que foram mapeadas a partir dos códigos binários que representam os nomes dos *tokens.*

Figura 22 - Funcionamento da etapa de criação de DNA sintético



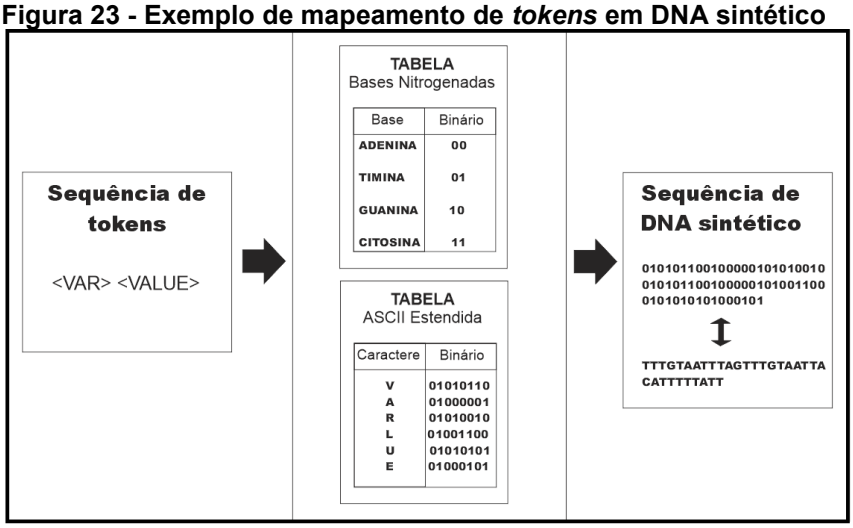
Fonte: Autoria própria

Um exemplo da execução da etapa de criação de um DNA sintético é mostrado pela Figura 23. A entrada é um código-fonte que foi convertido em dois *tokens*: VAR e VALUE. Após saber o nome de cada *token*, busca-se por meio da tabela ASCII a

codificação binária de cada caractere que forma essa nomeação. Nesse exemplo, as letras ou caracteres a serem buscados são: V, A, R, L, U e E.

Com as sequências de bits definida por meio da concatenação desses códigos, mapeia-se a cada dois *bits* uma base nitrogenada. Por exemplo, os dois primeiros *bits* do código binário da letra maiúscula "V" é "01", uma vez que o código para tal caractere é "01010110". Os *bits* "01" são convertidos em uma base nitrogenada: a timina (T). Esse mapeamento é repetido para todos os pares de bits contíguos concatenados respeitando a tabela de bases nitrogenadas, que nesse exemplo segue conforme a Figura 23.

Figura 23 - Exemplo de mapeamento de *tokens* em DNA sintético



Fonte: Autoria própria

Na BIOPLAG essa etapa de mapeamento de um artefato digital em uma sequência biológica sintética é fundamental para gerar a entrada para um método de alinhamento de sequências. O alinhamento pode ser executado por ferramentas de bioinformática capazes de encontrar regiões de similaridades presentes entre as sequências de entrada. A próxima atividade da abordagem é responsável por lidar com a execução destas ferramentas.

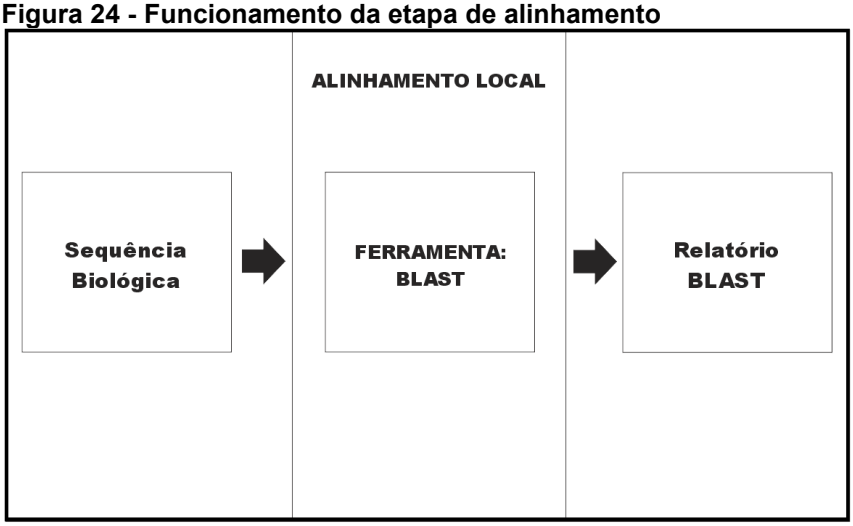
4.4 ALINHAMENTO DE SEQUÊNCIAS BIOLÓGICAS

O alinhamento é um processo responsável por identificar o nível de similaridade entre duas ou mais sequências, conforme apresentado no Capítulo 2. Para a BIOPLAG a aplicação desse processo é feita com o intuito de analisar duas ou mais sequências biológicas, que neste caso utiliza-se o tipo de proteína. O objetivo é

utilizar os relatórios produzidos por ferramentas de bioinformática de alinhamento para identificar uma taxa de similaridade entre os códigos-fonte analisados.

A atividade de alinhamento contida na abordagem é ilustrada pela Figura 24. A entrada desse processo é uma ou mais sequências de proteína, no entanto, esta abordagem é capaz de utilizar outros tipos como de DNA. Cada estrutura biológica definida representa um determinado código-fonte, uma vez que nesse caso são aminoácidos mapeados a partir de *tokens* gerados pela atividade II. O alinhamento é executado considerando esse mapeamento como entrada para a execução de uma ferramenta de bioinformática específica para essa tarefa.

Figura 24 - Funcionamento da etapa de alinhamento



Fonte: Autoria própria

A abordagem utiliza a ferramenta BLAST para a execução de um tipo específico de alinhamento: o local. A ferramenta BLAST foi escolhida por apresentar as seguintes características: criação de base de sequências própria e execução do alinhamento tipo local. Além disso, a complexidade de tempo de execução foi levada em consideração para otimizar a eficiência geral da detecção de plágio da BIOPLAG.

A vantagem de se utilizar alinhamento local é poder identificar diversas regiões similares entre as sequências analisadas. Em plágio de programação, os plagiadores inserem modificações em diferentes regiões ao longo do código inteiro. A aplicação do BLAST em proteína sintética pode ser capaz de identificar tais modificações em diferentes partes ou regiões de um código de programação, levando em consideração que a representatividade dos *tokens* são suficientes para deixar as modificações indiferentes em relação ao conteúdo original modificado. A saída do processo de alinhamento executado pelo BLAST é um relatório com parâmetros

avaliativos de similaridade, ilustrados no Capítulo 2. Esse relatório é lido na atividade seguinte da BIOPLAG a fim de obter uma taxa de similaridade unificada em porcentagem.

Ao realizar o alinhamento local com uma entrada ou *query* de busca, torna-se possível compará-la com todas as outras sequências presentes na base de dados de sequências gerada. Essa característica presente na ferramenta se ajusta ao cenário de aplicação da abordagem criada, tendo em vista que um código-fonte pode ser comparado com outros suspeitos podendo ser um ou mais. Outras ferramentas não possuem essa característica, e acrescentar essa funcionalidade a elas poderia afetar a complexidade computacional de tempo e espaço das mesmas.

Ao analisar a complexidade computacional de tempo da BIOPLAG, verifica-se que ela é limitada pelo custo de execução do processo de alinhamento. As outras etapas ou processos como: geração de *tokens* e mapeamento em sequências biológicas possuem uma complexidade O(n^2), sendo n o tamanho da entrada. Conforme apresentado no capítulo 2, o BLAST possui uma complexidade no pior caso de O(n^2), e o seu alinhamento produziria uma complexidade O(n^3) para a abordagem criada considerando todos os n códigos-fontes de entrada.

4.5 IDENTIFICAÇÃO DA TAXA DE SIMILARIDADE

A taxa de similaridade de um código-fonte é um valor numérico variando de 0 até 1, que indica o quanto um determinado código de programação é similar a outro. Valores próximos a zero indicam uma menor similaridade entre os códigos analisados, enquanto os valores próximos a 1 indicam uma maior similaridade. Esse valor é apresentado em forma de porcentagem, sendo necessária a multiplicação da taxa calculada por 100.

Para a BIOPLAG esse valor é calculado a partir de dados gerados pelo relatório da ferramenta de alinhamento BLAST. A Figura 25 apresenta um exemplo de relatório parcial gerado a partir de um alinhamento de sequências de proteína sintética. Dados informados nesse resultado fazem parte das variáveis que compõem a formulação matemática do cálculo da similaridade.

Figura 25 - Exemplo de relatório BLAST

```
 Score = 49.5 bits (130),  Expect(5) = 9e-57
 Identities = 16/25 (64%), Positives = 16/25 (64%), Gaps = 0/25 (0%)

Query  1   REGCCCEGCGCGCQCCGCQCCGCGC  25
           REGCCCEGC C   C     CGCGC
Sbjct  2   REGCCCEGCQCCEGCGCGCGCGCGC  26
```

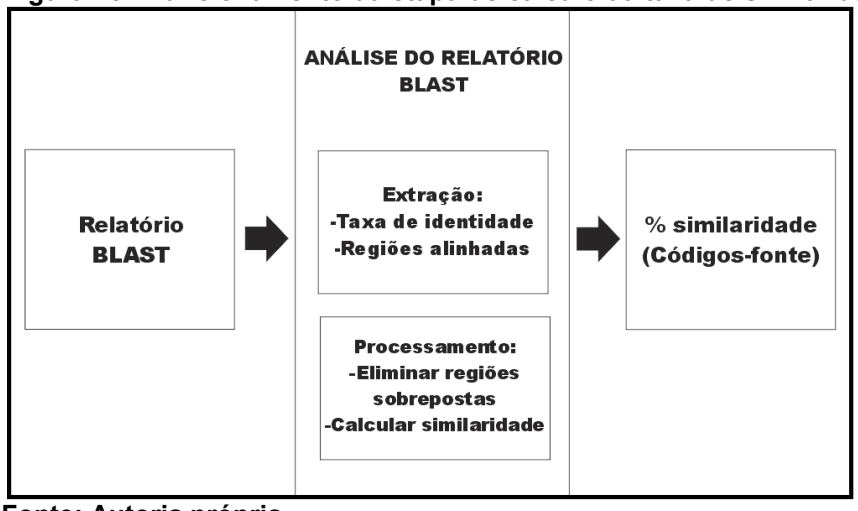
Fonte: Autoria própria

Considerando o relatório do alinhamento, o numerador da relação: "*Identities*" é utilizado para representar a variável n_i que representa quantas unidades de composição da sequência biológica são iguais na região alinhada i. Um relatório pode contar até k regiões alinhadas. Pode acontecer de uma ou mais regiões estarem sobrepostas, e essa sobreposição deve ser eliminada para não contabilizar repetições.

A sequência biológica sintética a ser analisada ou "*Query*" possui um tamanho h, que representa a quantidade de unidades de composição de sua estrutura. A outra sequência denominada "*Sbjct*" é referente a um dos outros códigos-fontes suspeitos utilizados para comparação, que pertence a base de sequências biológicas sintéticas previamente geradas por meio das atividades anteriores e armazenadas na ferramenta BLAST.

O funcionamento dessa atividade do cálculo da taxa de similaridade para a abordagem criada é ilustrado na Figura 26. A entrada para esse processo é o próprio relatório BLAST gerado a partir do alinhamento realizado na atividade III. Após fazer a leitura do relatório, ocorre a fase de análise do mesmo em duas etapas: extração e processamento.

Figura 26 - Funcionamento da etapa de cálculo da taxa de similaridade



Fonte: Autoria própria

A fase de extração dos dados é responsável por identificar os seguintes parâmetros: taxa de identidade e regiões alinhadas. Esses parâmetros são utilizados por serem capazes de definir o nível de igualdade entre regiões com alta similaridade nas sequências comparadas. Após a extração dos dados, executa-se um procedimento de eliminação de regiões alinhadas sobrepostas.

Com a análise do relatório realizada, atribui-se os dados extraídos e processados às variáveis que compõem a Fórmula 2 proposta para o cálculo da taxa de similaridade. O resultado encontrado com a sua aplicação é a porcentagem de similaridade entre os códigos-fontes analisados, ao multiplicar o valor de $sim(Query, Sbjct)$ por 100.

$$sim(Query, Sbjct) = \left(\frac{\sum_{i=1}^{k} n_i}{h}\right), \in [0, 1] \quad (2)$$

Um exemplo de funcionamento da identificação da taxa de similaridade é apresentado na Figura 27. Considerando o seguinte cenário: o tamanho da sequência de proteína sintética "*Query*" é de 100 aminoácidos, e possui apenas uma única sequência "*Sbjct*" na base de sequências potencialmente suspeitas da ferramenta BLAST.

Ao analisar o relatório, percebe-se que duas regiões alinhadas foram encontradas pela ferramenta. No entanto, trata-se de regiões sobrepostas, uma vez que a 1ª e a 2ª região consideram uma mesma sequência a partir do 11ª até o 35ª aminoácido da *Query*.

A eliminação da sobreposição é realizada decidindo por manter apenas uma das regiões. Aquela que será mantida deve corresponder a uma região em que a posição do primeiro aminoácido é mais próxima do início ou 1ª posição da sequência *Query*.

Considerando que o primeiro aminoácido da 1ª e 2ª região estão na posição 10ª e 11ª, respectivamente, será eliminada a 2ª região, uma vez que, o seu início está mais distante do começo da sequência *Query* de entrada. Após eliminar as regiões sobrepostas, calcula-se a taxa de similaridade pela Fórmula 2.

$$sim(Query, Sbjct) = \left(\frac{\sum_{i=1}^{k} n_i}{h}\right) = \left(\frac{16}{100}\right) = 0{,}16 \quad (2)$$

Figura 27 - Exemplo de alinhamento com sobreposição

1ª Região Alinhada:

```
Score = 49.5 bits (130), Expect(5) = 9e-57
Identities = 16/25 (64%), Positives = 16/25 (64%), Gaps = 0/25 (0%)

Query  10   REGCCCEGCGCGCQCCGCQCCGCGC  35
            REGCCCEGC C   C     CGCGC
Sbjct  12   REGCCCEGCQCCEGCGCGCGCGCGC  36
```

2ª Região Alinhada:

```
Score = 69.5 bits (134), Expect(5) = 1e-52
Identities = 16/25 (64%), Positives = 16/25 (64%), Gaps = 0/25 (0%)

Query  11   EGCCCEGCGCGCQCCGCQCCGCGCR  36
            EGCCCEGC C   C     CGCGCR
Sbjct  12   EGCCCEGCQCCEGCGCGCGCGCGCR  36
```

Fonte: Autoria própria

O resultado encontrado com a aplicação da Fórmula 2 foi 0,16, indicando por meio da conversão em porcentagem, que o código-fonte representado pela sequência *Query* é 16% similar em relação a um outro da *Sbjct*. Nota-se que nesse exemplo houve apenas uma região de alinhamento válida, mas dependendo das sequências de entrada poderia conter mais de uma região.

4.6 CONSIDERAÇÕES FINAIS DO CAPÍTULO

Este capítulo apresentou a abordagem de detecção de plágio em códigos-fontes, denominada BIOPLAG. A abordagem busca contemplar a detecção de diferentes tipos de plágio em programação, sendo fornecido como resultado da BIOPLAG um valor em porcentagem para indicar o nível de similaridade entre os códigos-fontes comparados.

A abordagem criada é fundamentada em conceitos da bioinformática e da computação. O seu funcionamento depende da técnica de alinhamento de sequências biológicas com a utilização de uma ferramenta de bioinformática para esse propósito: BLAST. Além dessa técnica, utiliza-se o conceito de *tokens* da computação para mapear códigos-fontes em sequências biológicas (DNA, Proteína, dentre outras) tornando possível o uso de conceitos, técnicas e ferramentas da bioinformática.

Dentre as melhorias propostas por meio da BIOPLAG, têm-se: contemplar os níveis de modificações em códigos-fonte categorizados por Faidhi e Robinson (1987); permitir que qualquer linguagem de programação seja suportada, proporcionar melhor qualidade na detecção de plágio em códigos-fontes sem desconsiderar a complexidade de tempo de execução da solução.

5 RESULTADOS

De acordo com o mapeamento sistemático realizado no capítulo 3, identificou a necessidade de validar ferramentas de detecção de plágio em código-fonte por meio de diferentes cenários de testes. Os testes buscam contemplar uma validação decorrente das técnicas utilizadas pelos plagiadores.

A validação adotada pelas pesquisas identificadas no mapeamento sistemático utiliza com maior frequência até 100 (cem) códigos-fontes na composição de seus testes. Esses códigos são produzidos de forma manual ou extraídos de uma base. Uma outra forma de produção é por meio da coleta de atividades de programação em disciplinas de computação de ensino superior.

Para avaliar a BIOPLAG foram realizados 2 (dois) experimentos com alunos de ensino superior da Universidade Tecnológica Federal do Paraná campus Ponta Grossa, e 1 (um) experimento com programadores de uma empresa da região focada em desenvolvimento de produtos de software. A partir dos 3 (três) experimentos foram produzidos 336 (trezentos e trinta e seis) códigos-fontes utilizados para validar 7 (sete) cenários de testes.

O processo de avaliação é descrito conforme a aplicação da BIOPLAG para detectar plágio por meio de 168 testes. A seção 5.1 descreve quais técnicas de plágio em programação cada cenário propõe validar e apresenta os experimentos realizados para produzir os seus testes. A seção 5.2 relata a implementação da BIOPLAG, além dos resultados encontrados com a sua aplicação. A Seção 5.3 apresenta um comparativo dos resultados obtidos em relação as ferramentas consideradas referências nesta área: MOSS e JPLAG.

5.1 CENÁRIOS DE TESTES

Os níveis de plágio em programação elaborados por Faidhi e Robinson (1987) foram utilizados como referência para a elaboração de 7 (sete) cenários de testes nomeados por letras de A até G. O detalhamento e a identificação de quais técnicas de plagiadores foram utilizadas para produzir os códigos-fontes em cada cenário pode ser encontrado no Quadro 9.

Os cenários A, B, C, D, E, F, e G buscam avaliar, respectivamente, a capacidade de detecção de plágio dos níveis I, II, III, IV, V, VI e falsos negativos/positivos. O cenário G é composto por códigos-fontes distintos não plagiados o seu objetivo é detectar a presença de falsos positivos e falsos negativos.

Quadro 9 - Técnicas de plágio em programação avaliadas em cenários de A até F

Número de Identificação	Nível de Faidhi e Robison (1987)	Descrição da Técnica de Plágio em Código-Fonte
1	Nível I	Inserir, remover e alterar comentários
2	Nível I	Modificar a indentação do código (Espaços em branco e *layout*).
3	Nível II	Alterar o nome de variáveis
4	Nível II	Alterar o nome de constantes
5	Nível II	Alterar o nome de funções
6	Nível II	Alterar o nome de procedimentos
7	Nível III	Alterar o posicionamento de uma variável no código
8	Nível III	Declarar variáveis extras
9	Nível III	Declarar constantes extras
10	Nível III	Alterar a declaração de uma função: o nome, tipo dos parâmetros e o retorno
11	Nível III	Alterar o posicionamento de funções no código
12	Nível III	Alterar a declaração de um procedimento: o nome e o tipo dos parâmetros
13	Nível III	Alterar o posicionamento de procedimentos no código
14	Nível III	Alterar o tipo das variáveis
15	Nível IV	Converter um procedimento em função
16	Nível IV	Converter uma função em procedimento
17	Nível IV	Unir dois ou mais procedimentos em apenas um
18	Nível IV	Unir duas ou mais funções em apenas uma
19	Nível IV	Criar novos procedimentos
20	Nível IV	Criar novas funções
21	Nível IV	Substituir a chamada de uma função com o seu respectivo conteúdo
22	Nível IV	Substituir a chamada de um procedimento com o seu respectivo conteúdo
23	Nível V	Alterar instrução “*for*” por “*while*”
24	Nível V	Alterar instrução “*if*” por “*switch-case*”
25	Nível V	Alterar instruções de incremento: *x = x + 1* por *x += 1* ou *x++* ou *++x*
26	Nível V	Alterar instruções de decremento: *x = x – 1* por *x -= 1* ou *x--* ou *–x*
27	Nível VI	Alterar as expressões lógicas das estruturas condicionais (“*if*” e “*switch-case*”)

28	Nível VI	Alterar as expressões lógicas das estruturas de repetição (“for”, “while” e “do-while”)
29	Nível VI	Converter de ascendente para descendente um *loop* (mudar para decremento a cada iteração)
30	Nível VI	Converter de descendente para ascendente um *loop* (mudar para incremento a cada iteração)
31	Nível VI	Converter instruções repetitivas em um *loop*
32	Nível VI	Converter *loop* para recursão

Fonte: Autoria Própria

Para cada cenário de teste de A até F, a implementação de seus códigos-fontes plagiados foi realizada integralmente com a participação de alunos da Universidade Tecnológica Federal do Paraná campus Ponta Grossa e de programadores de uma empresa de tecnologia da informação. Ao todo foram realizados 3 (três) experimentos contando com a participação de 34 (trinta e quatro) voluntários, sendo 25 (vinte e cinco) alunos de graduação em Ciência da Computação, 3 (três) mestrandos em Ciência da Computação e 6 (seis) programadores.

A realização dos experimentos dependeu da utilização de uma base de códigos-fontes elaborada por Mou *et al*. (2016) contendo 52.000 (cinquenta e dois mil) exemplares escritos na linguagem *C* e *C++*. Solicitou-se para os voluntários escolherem qualquer um dos exemplares que fossem na linguagem *C*, e fizessem uma cópia plagiada de suas respectivas escolhas. As técnicas utilizadas para plagiar poderiam ser escolhidas conforme o quadro 8, respeitando os níveis de plágio.

A condução da criação dos cenários de testes foi orientada pela premissa da liberdade de escolha dos voluntários. O objetivo é produzir situações reais de plágio em programação, uma vez que, os exemplos criados são produzidos por estudantes e programadores.

Para assegurar a qualidade dos testes produzidos a partir das implementações dos voluntários, realizou-se uma avaliação manual para certificar que todos os códigos-fontes plagiados estão livres de erro de compilação e de indicação incorreta das técnicas de plágio que foram utilizadas. Por exemplo, esse processo avaliativo verifica se foram utilizadas corretamente as técnicas para cada nível e se a implementação não apresenta erros de programação, além de verificar a linguagem de codificação que deve ser em *C*.

A relação das implementações enviadas pelos voluntários que foram reprovadas e aprovadas podem ser encontradas nas Tabelas 1, 2 e 3. Os três

experimentos contribuíram, respectivamente, com 98, 29 e 21 exemplos de plágio em programação para formar os testes dos cenários de A até F.

Tabela 1 - Avaliação dos testes produzidos no primeiro experimento

Cenário	Enviados	Reprovados	Aprovados
A	25	2	23
B	23	1	22
C	22	7	15
D	21	4	17
E	18	6	12
F	19	10	9
Total	128	30	98

Fonte: Autoria Própria

O cenário G foi criado a partir da seleção aleatória de 60 códigos-fontes pertencentes a base de códigos. Após fazer a seleção, houve uma avaliação manual para certificar-se que todos os exemplos sorteados sejam distintos e escritos em linguagem *C*. Espera-se que os detectores avaliados apresentem como resultado a inexistência de plágio ou um nível baixo de similaridade ao analisarem esses exemplos selecionados.

Tabela 2 - Avaliação dos testes produzidos no segundo experimento

Cenário	Enviados	Reprovados	Aprovados
A	3	0	3
B	3	1	2
C	6	2	4
D	6	1	5
E	9	2	7
F	12	4	8
Total	39	10	29

Fonte: Autoria Própria

Cada um dos testes pertencentes a um determinado cenário consiste em um código-fonte original e a sua respectiva cópia plagiada. Portanto, um teste corresponde a um par de códigos-fontes a serem avaliados. Ao considerar esses pares, os cenários de testes deste trabalho contemplam um total de 336 (trezentos e trinta e seis) códigos-fontes utilizados em 168 (cento e sessenta e oito) testes de plágio.

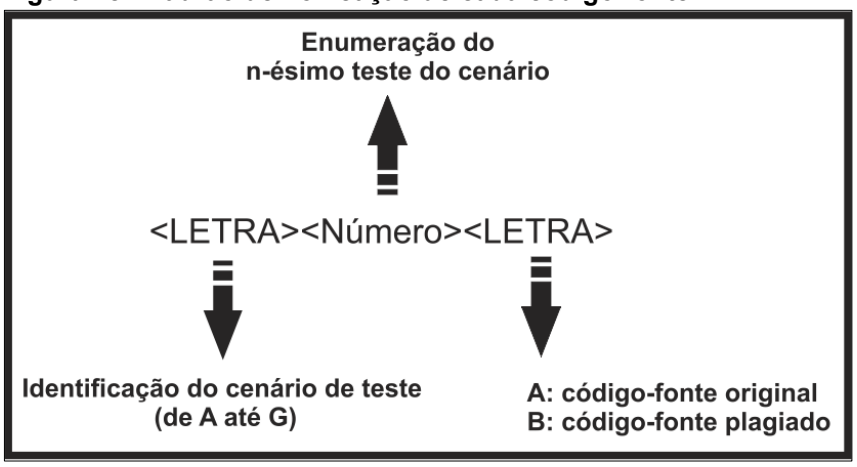
Tabela 3 - Avaliação dos testes produzidos no terceiro experimento

Cenário	Enviados	Reprovados	Aprovados
A	0	0	0
B	0	0	0
C	6	0	6
D	5	0	5
E	5	0	5
F	5	0	5
Total	21	0	21

Fonte: Autoria Própria

Para organizar os cenários de testes a serem executados pela BIOPLAG, um padrão de enumeração para os testes foi criado. A Figura 28 ilustra como esse padrão funciona para identificar todos os exemplos de plágio a serem testados, para cada um dos 336 (trezentos e trinta e seis) códigos-fontes propõe-se uma nomeação composta de três partes.

Figura 28 - Padrão de nomeação de cada código-fonte



Fonte: Autoria própria

Em suma, a nomeação completa de cada exemplo de plágio criado no experimento foi feita da seguinte forma: LETRA NÚMERO LETRA. As três partes são uma letra de A até F, um número, e a letra A ou B. A primeira letra identifica o cenário de teste, o número representa a identificação do teste no cenário, e a última letra indica se o código-fonte é original ou plagiado.

Acerca do padrão de nomeação, por exemplo, ao considerar o primeiro teste do primeiro cenário, o detector de plágio avalia o par de códigos-fontes: “A1A” e “A1B”.

Ambos representam um único teste do cenário A, a relação completa do nome dos 336 (trezentos e trinta e seis) códigos-fontes que compõem os 168 (cento e sessenta e oito) testes dos cenários de A até G é apresentada no Quadro 10.

Quadro 10 - Lista completa dos códigos-fontes que compõem os cenários de testes

Cenário A	Cenário B	Cenário C	Cenário D	Cenário E	Cenário F	Cenário G
A1A	B1A	C1A	D1A	E1A	F1A	G1A
A1B	B1B	C1B	D1B	E1B	F1B	G1B
A2A	B2A	C2A	D2A	E2A	F2A	G2A
A2B	B2B	C2B	D2B	E2B	F2B	G2B
A3A	B3A	C3A	D3A	E3A	F3A	G3A
A3B	B3B	C3B	D3B	E3B	F3B	G3B
A4A	B4A	C4A	D4A	E4A	F4A	G4A
A4B	B4B	C4B	D4B	E4B	F4B	G4B
A5A	B5A	C5A	D5A	E5A	F5A	G5A
A5B	B5B	C5B	D5B	E5B	F5B	G5B
A6A	B6A	C6A	D6A	E6A	F6A	G6A
A6B	B6B	C6B	D6B	E6B	F6B	G6B
A7A	B7A	C7A	D7A	E7A	F7A	G7A
A7B	B7B	C7B	D7B	E7B	F7B	G7B
A8A	B8A	C8A	D8A	E8A	F8A	G8A
A8B	B8A	C8A	D8A	E8B	F8B	G8B
A9A	B9A	C9A	D9A	E9A	F9A	G9A
A9B	B9B	C9B	D9B	E9B	F9B	G9B
A10A	B10A	C10A	D10A	E10A	F10A	G10A
A10B	B10B	C10B	D10B	E10B	F10B	G10B
A11A	B11A	C11A	D11A	E11A	F11A	G11A
A11B	B11B	C11B	D11B	E11B	F11B	G11B
A12A	B12A	C12A	D12A	E12A	F12A	G12A
A12B	B12B	C12B	D12B	E12B	F12B	G12B
A13A	B13A	C13A	D13A	E13A	F13A	G13A
A13B	B13B	C13B	D13B	E13B	F13B	G13B
A14A	B14A	C14A	D14A	E14A	F14A	G14A
A14B	B14B	C14B	D14B	E14B	F14B	G14B
A15A	B15A	C15A	D15A	E15A	F15A	G15A
A15B	B15B	C15B	D15B	E15B	F15B	G15B
A16A	B16A	C16A	D16A	E16A	F16A	G16A
A16B	B16B	C16B	D16B	E16B	F16B	G16B
A17A	B17A	C17A	D17A	E17A	F17A	G17A
A17B	B17B	C17B	D17B	E17B	F17B	G17B
A18A	B18A	C18A	D18A	E18A	F18A	G18A
A18B	B18B	C18B	D18B	E18B	F18B	G18B

A19A	B19A	C19A	D19A	E19A	F19A	G19A
A19B	B19B	C19B	D19B	E19B	F19B	G19B
A20A	B20A	C20A	D20A	E20A	F20A	G20A
A20B	B20B	C20B	D20B	E20B	F20B	G20B
A21A	B21A	C21A	D21A	E21A	F21A	
A21B	B21B	C21B	D21B	E21B	F21B	
A22A	B22A	C22A	D22A	E22A	F22A	
A22B	B22B	C22B	D22B	E22B	F22B	
A23A	B23A	C23A	D23A	E23A		
A23B	B23B	C23B	D23B	E23B		
A24A	B24A	C24A	D24A	E24A		
A24B	B24B	C24B	D24B	E24B		
A25A		C25A	D25A			
A25B		C25B	D25B			
A26A			D26A			
A26B			D26B			
			D27A			
			D27B			

Fonte: Autoria própria

Os detectores de plágio avaliados neste trabalho são testados por meio de exemplos de plágio em programação. O Quadro 11 identifica quais técnicas de modificação de código-fonte estão presentes em cada exemplo testado, além da enumeração dos testes. Ressalta-se que a identificação de cada técnica aplicada é feita a partir de um número identificador correspondente apresentado no Quadro 9.

Quadro 11 - Lista completa das técnicas de plágio utilizadas em cada teste nos cenários A-G

Cenário	Número do Teste	Técnicas de plágio utilizadas
A	1	1 e 2
A	2	1 e 2
A	3	1 e 2
A	4	1 e 2
A	5	1 e 2
A	6	2
A	7	1 e 2
A	8	1 e 2
A	9	1 e 2
A	10	2
A	11	1 e 2
A	12	1
A	13	1 e 2
A	14	1 e 2

A	15	2
A	16	1 e 2
A	17	2
A	18	1 e 2
A	19	1 e 2
A	20	1 e 2
A	21	1 e 2
A	22	2
A	23	1 e 2
A	24	1 e 2
A	25	1 e 2
A	26	1
B	1	1, 2 e 3
B	2	1, 2 e 3
B	3	1 e 3
B	4	2 e 3
B	5	1, 2 e 3
B	6	3
B	7	1, 2, 3 e 5
B	8	1 e 3
B	9	2, 3 e 6
B	10	3
B	11	1 e 3
B	12	1 e 3
B	13	3 e 6
B	14	2 e 3
B	15	1, 2 e 3
B	16	1, 2 e 3
B	17	3 e 6
B	18	1, 2 e 3
B	19	1, 2 e 3
B	20	2, 3 e 5
B	21	1, 2 e 3
B	22	3
B	23	2 e 3
B	24	1, 2 e 3
C	1	1, 2, 7, 8, 9 e 14
C	2	1, 2, 3, 7 e 8
C	3	2, 3, 8 e 10
C	4	2, 3, 9, 10 e 11
C	5	1, 2, 3, 7 e 8
C	6	7, 8 e 14
C	7	1, 3, 8 e 10

C	8	1, 2, 7, 8, 10 e 11.
C	9	7 e 8.
C	10	7 e 14.
C	11	1, 2, 3, 7, 8 e 9.
C	12	2, 3, 7 e 8.
C	13	1, 2, 3, 7, 8, 9 e 14.
C	14	2, 3, 7 e 8.
C	15	1, 2, 3, 7 e 8.
C	16	1, 2, 3, 7 e 10.
C	17	5 e 8.
C	18	2, 3, 7 e 8.
C	19	2, 3 e 9.
C	20	1, 2, 3 e 8.
C	21	1, 2, 3, 7, 8 e 14.
C	22	2, 3 e 14.
C	23	2, 7, 8 e 10.
C	24	1, 2, 3, 7, 8 e 12.
C	25	1, 2, 3, 7 e 8.
D	1	1, 2, 7, 8, 9, 14 e 20.
D	2	1 e 20.
D	3	21.
D	4	1, 2 e 20.
D	5	2, 3, 7 e 20.
D	6	12, 13 e 19.
D	7	1, 3, 8, 13, 15 e 19.
D	8	15 e 19.
D	9	2 e 19.
D	10	7, 14 e 20.
D	11	1, 2, 3, 7, 8, 9, 11 e 19.
D	12	3, 7, 15 e 19.
D	13	1, 2, 3, 4, 7, 8, 9, 13, 14, 19 e 20.
D	14	2, 7, 8 e 19.
D	15	2, 3, 7, 13 e 19.
D	16	1, 2, 3, 7, 8, 11, 19 e 20.
D	17	20.
D	18	1, 2, 7, 8 e 20
D	19	1, 2, 3, 7, 8, 14 e 20.
D	20	1, 2, 3, 7, 8 e 20
D	21	2, 3, 9, 11 e 20.
D	22	2, 3, 9, 11 e 20.
D	23	2, 10, 13 e 20.
D	24	2, 11 e 19.
D	25	2, 3, 7, 11, 14 e 19.

D	26	2, 13 e 19.
D	27	1, 2, 3, 7, 13 e 19.
E	1	1, 2, 9, 14, 20 e 25.
E	2	3, 7, 8, 20 e 23.
E	3	2 e 23.
E	4	1, 2, 8, 23 e 25.
E	5	1, 3, 13, 15, 19 e 23.
E	6	23.
E	7	7, 11, 14, 20 e 23.
E	8	1, 2, 3, 7, 8, 9, 11, 19 e 25.
E	9	2, 3, 15, 23 e 25.
E	10	8, 15, 23 e 25.
E	11	1, 2, 3, 7, 8, 11, 19, 20, 23 e 25.
E	12	23.
E	13	1, 2, 3, 7, 8, 20 e 23.
E	14	1, 2, 3, 5, 7 e 23.
E	15	1, 2, 3, 7, 8, 9, 13, 20 e 25.
E	16	1, 2, 3, 7, 8, 20 e 25.
E	17	1, 2, 3, 7, 8, 20, 25 e 26.
E	18	2, 3, 9, 20, 23 e 24.
E	19	2, 3, 8, 9, 20, 24 e 25.
E	20	2, 6 e 23.
E	21	1, 2, 3, 7, 8, 19 e 23.
E	22	2, 7, 10 e 23.
E	23	2, 23 e 25.
E	24	1, 2, 3, 8, 11, 20 e 25.
F	1	1, 3, 8, 11, 19, 25, 28 e 29.
F	2	1, 2, 9, 11, 14, 20 e 27.
F	3	1, 5, 7, 8 e 27.
F	4	2, 28 e 29.
F	5	28 e 29.
F	6	1, 3, 8, 11, 19, 23, 28 e 29.
F	7	2, 23, 28 e 29.
F	8	2, 8 e 28.
F	9	28 e 30.
F	10	1, 2, 15, 20, 28 e 29.
F	11	1, 2, 3, 7, 8, 9, 20, 25 e 28.
F	12	1, 2, 3, 7, 8, 20, 25 e 28.
F	13	1, 2, 3, 7, 8, 20, 25, 26 e 28.
F	14	1, 2, 3, 7, 8, 14, 20, 25, 27 e 28.
F	15	2, 3, 11, 19, 23, 27 e 31.
F	16	2, 3, 7, 9, 19, 23, 25, 27 e 28.
F	17	2, 3, 9, 20, 23, 25 e 27.

F	18	2, 3, 6, 23, 24 e 27.
F	19	2, 3, 7, 10, 14, 15, 28 e 29.
F	20	2, 7, 12, 13, 23, 25, 26, 27 e 28.
F	21	1, 2, 23, 25 e 29.
F	22	1, 2, 3, 8, 20, 28 e 29.
G	1	N/A
G	2	N/A
G	3	N/A
G	4	N/A
G	5	N/A
G	6	N/A
G	7	N/A
G	8	N/A
G	9	N/A
G	10	N/A
G	11	N/A
G	12	N/A
G	13	N/A
G	14	N/A
G	15	N/A
G	16	N/A
G	17	N/A
G	18	N/A
G	19	N/A
G	20	N/A

Fonte: Autoria Própria

Cada teste produzido nos cenários de A até F consegue representar diferentes técnicas de plágio em um único código-fonte plagiado. Essa representatividade é possível em função da categorização de Faidhi e Robinson (1987) que permite a acumulação de técnicas a partir de níveis crescentes de dificuldade de detecção.

Para os testes do cenário G não ocorre a aplicação de técnicas de plágio por avaliarem apenas códigos-fontes autênticos. As seções seguintes apresentam a aplicação da BIOPLAG a partir desses cenários de testes e a comparação de seus resultados em relação a outras ferramentas de referência: MOSS e JPLAG.

5.2 IMPLEMENTAÇÃO E AVALIAÇÃO DA BIOPLAG

Neste trabalho foi realizada a implementação da BIOPLAG, uma ferramenta de detecção automática de plágio em programação, que representa a abordagem fundamentada em bioinformática explicada no capítulo anterior. A solução realiza comparações de códigos-fontes escritos em linguagem *C* e fornece como resultado uma porcentagem indicando o nível de similaridade entre os comparados.

Para avaliar essa implementação da abordagem criada, os 7 cenários de testes elaborados foram utilizados na execução da ferramenta. Para definir um limiar de plágio no processo avaliativo, foram considerados os estudos de Xiong *et al*. (2009), Cosma e Joy (2012), Pawelczak (2013), Ajmal *et al*. (2014), Sulistiani e Karnalim (2019) e Mason, Gavrilovska e Joyner (2019).

Os detalhes da implementação da BIOPLAG são apresentados na subseção 5.2.1, indicando qual gerador de tokens foi utilizado e como foi integrada a ferramenta de bioinformática BLAST no funcionamento da solução desenvolvida. Acerca da avaliação de seu funcionamento, a subseção 5.2.2 discute o desempenho na correta detecção de diferentes níveis de plágio.

5.2.1 Implementação

Conforme o funcionamento da abordagem criada neste trabalho, para implementar a BIOPLAG definiu-se o gerador de *tokens* a ser utilizado, o mapeamento em proteína sintética e os parâmetros de alinhamento de proteína. A tecnologia utilizada para o desenvolvimento da solução foi a linguagem *Perl* (FOY, 2018) e *Shell Script* (BOTH, 2018) em ambiente de desenvolvimento *Unix* versão *BioLinux*.

O gerador de *tokens* adotado foi o *C Tokenize* versão 0.18, um módulo Perl disponibilizado a partir da *Comprehensive Perl Archive Network* (CPAN). Esse módulo é responsável por transformar um código-fonte em uma sequência de *tokens* e suporta apenas codificações em linguagem de programação C (CTOKENIZE, 2018).

O funcionamento do *C Tokenize* é alicerçado em 9 (nove) diferentes *tokens*, conforme o Quadro 12. Cada comando de codificação em um código-fonte é agrupado em uma ou mais categorias semânticas, que são representadas por *tokens*. Essas categorizações são mapeadas em aminoácidos no processo de mapeamento de código-fonte em proteína sintética.

Quadro 12 - Listagem de *tokens* e seus respectivos aminoácidos mapeados

Token	**Aminoácido**	**Abreviação do Aminoácido**	**Comandos Contemplados**
cpp	Arginina	R	Instruções de pré-processamento para inclusão e definição, por exemplo: #define EXEMPLO 1 #include “exemplo.h”
char_const	Asparagina	N	Um único caractere entre aspas simples, por exemplo: ‘\0’, ‘x’, ‘y’, ‘z’, dentre outros.
operator	Aspartato	D	Operadores aritméticos, relacionais, lógicos, *bitwise*, de atribuição, de endereço de variáveis, de ponteiros, de expressão condicional e de tamanho de variáveis. Por exemplo: +, -, *, /, %, ++, --, =, ==, , , =, =, &&, ||, !, &, |, , , +=, -=, *=, /=, dentre outros.
grammar	Cisteína	C	Componentes pertencentes a gramática de elaboração da sintaxe da linguagem, por exemplo: {, }, -, dentre outros.
number	Glutamina	Q	Representação de qualquer número, por exemplo: 1, 2, 3, 4, 5, 6, 7, 77, 777, 7.77, dentre outros.
reserved	Glutamato	E	Palavras reservadas em C, por exemplo: break, continue, int, float, void, struct, while, dentre outras.
word	Glicina	G	Identificadores, que podem ser nome de função ou variável, por exemplo: int exemplo1; void exemplo2 (void);
string	Histidina	H	Cadeias de caracteres atribuídas em vetores do tipo *char*, por exemplo: char nome_um[] = “Exemplo1”; char nome_dois[] = “Exemplo2”;
comment	N/A	N/A	Comentários em uma linha ou mais, por exemplo: // primeiro exemplo /* Segundo exemplo */

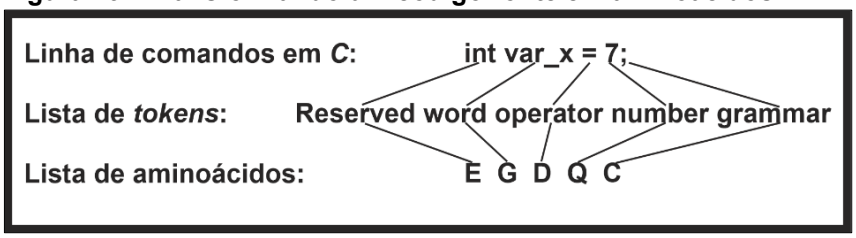
Fonte: Autoria Própria

Cada *token* é convertido em um aminoácido diferente, com exceção do “*comment*” que é eliminado por não agregar informação no processo de detecção de plágio. Um exemplo do funcionamento dessa conversão no mapeamento é ilustrado pela Figura 29, mostrando como uma linha de comandos em codificação *C* é transformada em uma lista de *tokens* e depois mapeada em aminoácidos.

A lista de aminoácidos representa uma proteína sintética criada a partir de um código-fonte. Essa lista ao ser armazenada em um arquivo no formato *fasta* permite a

utilização de ferramentas de bioinformática destinadas a realização de alinhamento de sequências biológicas.

Figura 29 - Transformando um código-fonte em aminoácidos



Fonte: Autoria própria

Após a obtenção de cada proteína sintética, realizou-se um alinhamento por meio do módulo *blastp* versão 2.2.28+ da ferramenta *BLAST* (NCBI, 2020). Os parâmetros utilizados são apresentados no Quadro 13. Ressalta-se que a escolha do valor de cada um foi pautada na busca por menor custo de processamento, com exceção do parâmetro "*matrix*" que apresenta diferentes valores conforme o tamanho da sequência de aminoácidos para aprimorar a qualidade do alinhamento.

Quadro 13 - Parâmetros adotados para o BLAST

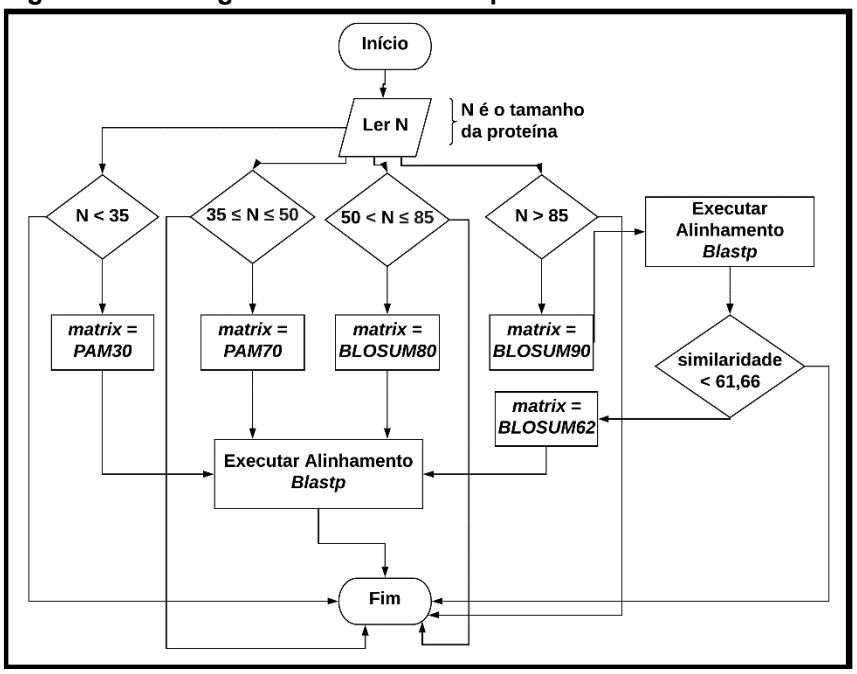
Parâmetros – Blastp	Valor
word_size	3
matrix	A depender do tamanho da sequência. Utilizadas: -*BLOSUM90*, *BLOSUM80*, *BLOSUM62*, *PAM70* e *PAM30*.
threshold	1
comp_based_stats	0
seg	*no*
soft_masking	*false*
db_soft_mask	*none*
db_hard_mask	*none*
xdrop_gap_final	25
window_size	40
evalue	1e-47
ungapped	*N/A*

Fonte: Autoria própria

O fluxograma ilustrado pela Figura 30 representa a variação dos valores adotados para o parâmetro *matrix*. Conforme o tamanho da proteína utilizada como

query no módulo *blastp*, um diferente sistema de pontuação é adotado para a matriz de substituição. O principal objetivo dessas mudanças é contribuir para a qualidade dos alinhamentos.

Figura 30 - Fluxograma da escolha do parâmetro *matrix*



Fonte: Autoria própria

Além da variação de parâmetros associada ao tamanho da proteína utilizada como *query*, existe um outro mecanismo de escolha de valor para a matriz de substituição. Esse mecanismo de variação adicional é feito devido ao cenário de aplicação da BIOPLAG neste trabalho. O cenário de aplicação é composto de conjuntos de testes com apenas proteínas sintéticas de tamanho superior a 85 aminoácidos.

Para aprimorar a qualidade dos resultados dos alinhamentos em proteínas com tamanho superior a 85, utilizou-se a matriz de substituição "*BLOSUM62*" para situações de não ocorrência de plágio. Ao executar a BIOPLAG, utiliza-se inicialmente a matriz "*BLOSUM90*" e caso seja constatada uma taxa de similaridade inferior a 61,66% esse mecanismo adicional é ativado.

A BIOPLAG foi implementada considerando esses mecanismos de mudanças para que ocorra em seu funcionamento apenas alinhamentos relevantes de qualidade. Para identificar essa relevância foi estabelecido que apenas os alinhamentos com taxas de identidade superiores a 80% fossem considerados.

5.2.2 Avaliação

Cada teste é composto de um par de códigos-fontes, sendo um original e o outro a sua versão plagiada. Espera-se como resultado a identificação de uma porcentagem de similaridade indicando a existência de plágio em cada um dos testes dos cenários de A até F, ao contrário do G, que deve indicar a inexistência de plágio.

Para definir uma porcentagem que indique o plágio em códigos-fontes, avaliou-se diferentes estudos. Ao todo foram 6 (seis) propostas diferentes analisadas, sendo as seguintes porcentagens propostas: 90% para Xiong *et al*. (2009), 70% para Cosma e Joy (2012), 80% para Pawelczak (2013), 50% para Ajmal *et al*. (2014), 50% para Sulistiani e Karnalim (2019) e 30% de acordo com os estudos de Mason, Gavrilovska e Joyner (2019). A média dos valores propostos é de 61.66%, sendo esse o valor adotado como limiar para avaliar os resultados da BIOPLAG ilustrados na Tabela 4.

Tabela 4 - Resultados da BIOPLAG nos cenários de testes

(continua)

Cenário	Número do Teste	Similaridade Detectada
A	1	100%
A	2	100%
A	3	84,61%
A	4	100%
A	5	95,06%
A	6	100%
A	7	100%
A	8	100%
A	9	100%
A	10	100%
A	11	100%
A	12	100%
A	13	98,35%
A	14	100%
A	15	100%
A	16	100%
A	17	100%
A	18	100%
A	19	100%
A	20	100%
A	21	100%
A	22	100%

Tabela 4 - Resultados da BIOPLAG nos cenários de testes

(continuação)

Cenário	Número do Teste	Técnicas de plágio utilizadas
A	23	100%
A	24	100%
A	25	100%
A	26	100%
B	1	100%
B	2	100%
B	3	100%
B	4	100%
B	5	100%
B	6	100%
B	7	100%
B	8	100%
B	9	100%
B	10	100%
B	11	100%
B	12	72,33%
B	13	100%
B	14	100%
B	15	100%
B	16	100%
B	17	100%
B	18	100%
B	19	100%
B	20	100%
B	21	100%
B	22	100%
B	23	100%
B	24	100%
C	1	67,01%
C	2	87,64%
C	3	47,72%
C	4	10,00%
C	5	89,70%
C	6	93,30%
C	7	74,33%
C	8	47,27%
C	9	87,20%
C	10	99,10%
C	11	89,13%
C	12	53,30%

Tabela 4 - Resultados da BIOPLAG nos cenários de testes

(continuação)

Cenário	Número do Teste	Técnicas de plágio utilizadas
C	13	97,89%
C	14	63,97%
C	15	93,75%
C	16	88,59%
C	17	34,14%
C	18	94,73%
C	19	97,70%
C	20	71,02%
C	21	19,87%
C	22	96,24%
C	23	94,55%
C	24	79,15%
C	25	64,13%
D	1	72,86%
D	2	74,80%
D	3	47,05%
D	4	72,25%
D	5	94,27%
D	6	99,12%
D	7	71,06%
D	8	93,96%
D	9	96,08%
D	10	92,61%
D	11	26,51%
D	12	62,34%
D	13	65,00%
D	14	68,94%
D	15	64,59%
D	16	74,39%
D	17	95,73%
D	18	83,00%
D	19	75,00%
D	20	89,24%
D	21	62,56%
D	22	64,63%
D	23	66,00%
D	24	72,26%
D	25	84,41%
D	26	81,73%

Tabela 4 - Resultados da BIOPLAG nos cenários de testes

(continuação)

Cenário	Número do Teste	Técnicas de plágio utilizadas
D	27	72,04%
E	1	36,84%
E	2	46,25%
E	3	89,47%
E	4	62,68%
E	5	69,45%
E	6	67,05%
E	7	82,25%
E	8	81,71%
E	9	76,58%
E	10	85,47%
E	11	62,98%
E	12	72,46%
E	13	78,15%
E	14	73,00%
E	15	83,39%
E	16	78,52%
E	17	73,80%
E	18	64,82%
E	19	62,50%
E	20	58,00%
E	21	61,70%
E	22	84,85%
E	23	64,17%
E	24	69,30%
F	1	93,98%
F	2	83,81%
F	3	20,97%
F	4	28,36%
F	5	96,13%
F	6	69,45%
F	7	95,16%
F	8	82,52%
F	9	79,44%
F	10	64,50%
F	11	63,63%
F	12	54,30%
F	13	62,50%
F	14	26,95%

Tabela 4 - Resultados da BIOPLAG nos cenários de testes

(conclusão)

Cenário	Número do Teste	Técnicas de plágio utilizadas
F	15	3,31%
F	16	63,52%
F	17	67,24%
F	18	64,08%
F	19	81,41%
F	20	80,48%
F	21	70,39%
F	22	79,45%
G	1	20,00%
G	2	10,41%
G	3	24,85%
G	4	27,54%
G	5	16,47%
G	6	5,60%
G	7	17,60%
G	8	17,08%
G	9	9,01%
G	10	8,96%
G	11	33,12%
G	12	7,32%
G	13	0%
G	14	9,09%
G	15	16,66%
G	16	10,00%
G	17	0%
G	18	0%
G	19	2,24%
G	20	19,19%

Fonte: Autoria Própria

Considerando o total de 168 testes, a BIOPLAG detectou plágio em 152 (90,48%) e não conseguiu detectar em 16 (9,52%). Analisando por cenários de testes, a taxa de acerto foi de 100% para o A, 100% para o B, 76% para o C, 92,59% para o D, 87,50% para o E, 77,27% para o F e 100% para o G.

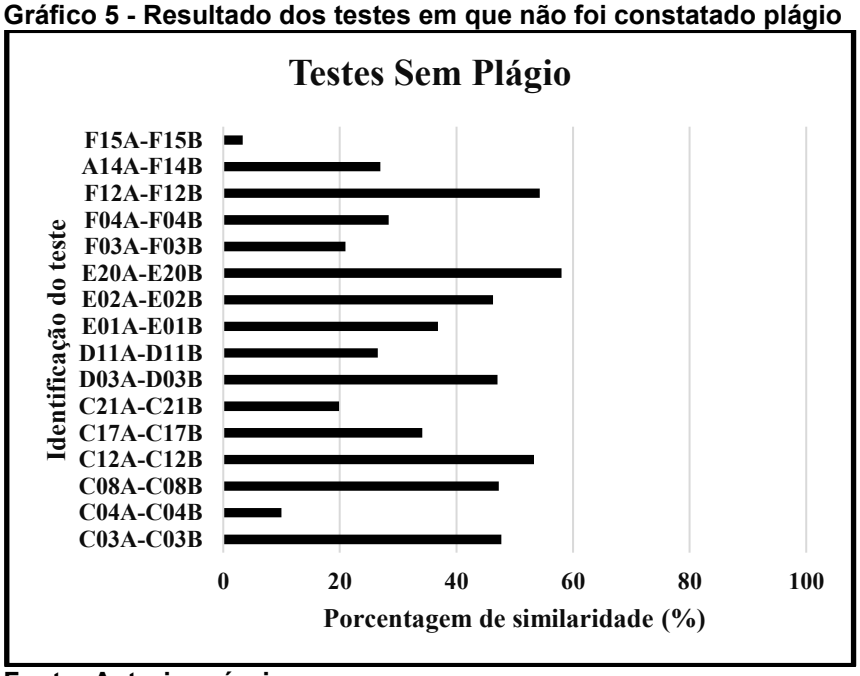
A média de acertos dos cenários de testes foi de 90,48%, sendo que em três cenários a taxa foi de 100%. Acerca daqueles que não obtiveram sucesso em todos os exemplos de plágio, o cenário C falhou em 6 de 25, o D em 2 de 27, o E em 3 de 24 e o F em 5 de 22. Ressalta-se que o cenário F que busca ilustrar falsos positivos e negativos obteve 100% de sucesso.

Constata-se que as taxas de acertos em cada cenário estão diretamente associadas ao grau de dificuldade do nível de plágio avaliado. Os testes que avaliaram os últimos níveis por apresentarem maior dificuldade acabaram obtendo menos sucesso em relação ao demais. Pode-se notar essa associação ao avaliar uma crescente queda nas taxas de acertos dos cenários D, E e F.

Apesar do grau de dificuldade se relacionar com a taxa de acerto, esse único fator não é determinante. Ao observar os resultados do cenário C, verifica-se que o seu desempenho foi inferior em relação aos demais que avaliam níveis superiores de plágio. Essa foi a única constatação de anomalia em relação aos resultados obtidos.

Um outro fator que afetou diretamente os resultados é o limiar adotado para indicar a presença de plágio. O Gráfico 5 apresenta a similaridade encontrada nos 16 testes em que não foi detectado plágio considerando a taxa média de 61.66%.

Gráfico 5 - Resultado dos testes em que não foi constatado plágio



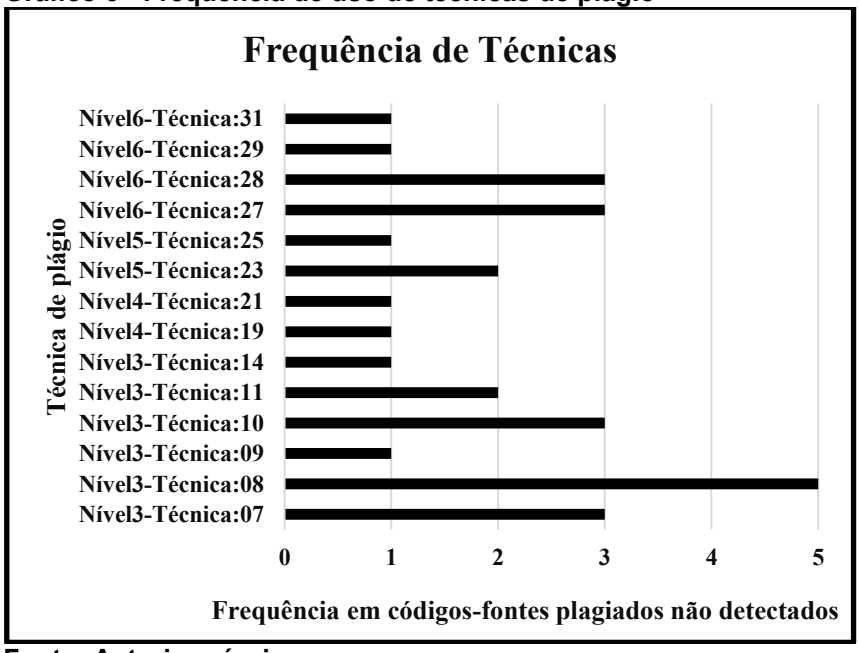
Fonte: Autoria própria

Adotando porcentagens menores como limiar, a taxa de acerto da BIOPLAG seria superior no processo de avaliação deste trabalho. Observa-se que em todos os testes avaliados foram constatados determinados níveis de similaridades, não houve como resultado uma porcentagem nula.

Para detalhar os resultados obtidos pela BIOPLAG, o Gráfico 6 apresenta a frequência de uso das técnicas de plágio nos 16 testes em que não foi detectado

plágio. Esse gráfico contabiliza as ocorrências das técnicas, sem considerar a acumulação de níveis e considerando apenas aquelas do próprio nível.

Gráfico 6 - Frequência de uso de técnicas de plágio



Fonte: Autoria própria

As técnicas do nível 3 e 4 presentes nos testes plagiados não detectados foram: alterar o posicionamento de uma variável, declarar variáveis extras, declarar constantes extras, alterar a declaração de uma função, alterar o posicionamento de funções, alterar o tipo das variáveis, criar novos procedimentos e substituir a chamada de uma função com o seu respectivo conteúdo.

Dentre aquelas do nível 5 e 6, as não detectadas foram: alterar instrução "*for*" para "*while*", alterar instruções de incremento, alterar as expressões lógicas das estruturas condicionais, alterar as expressões lógicas das estruturas de repetição, transformar *loop* ascendente para descendente e converter instruções repetitivas em um *loop*.

Essas técnicas presentes nos 16 testes não indicam que a BIOPLAG não seja capaz de detectá-las. Elas foram detectadas com sucesso em outros testes, no entanto, a combinação de determinadas técnicas podem afetar os resultados assim como o limiar adotado.

5.3 COMPARAÇÃO DE RESULTADOS DA BIOPLAG, MOSS E JPLAG

Para realizar uma avaliação comparativa entre o desempenho da BIOPLAG e outras ferramentas de referência no estado da arte, calculou-se diferentes parâmetros avaliativos tais como apresentados nos trabalhos de Xiong *et al* (2009), Cosma e Joy (2012), Narayanan e Simi (2012), Son *et al*. (2013), Durić e Gašević (2013), Acampora e Cosma (2015), Sulistiani e Karnalim (2019) e Nichols *et al*. (2019).

Os parâmetros avaliativos adotados para este trabalho foram os seguintes: precisão (*Precision*), revocação (*Recall*) e medida F (*F-measure*). A fórmula de cada um deles está representada pela Fórmula 3, Fórmula 4 e Fórmula 5, respectivamente. Em suas elaborações, a variável *VP* corresponde aos verdadeiros positivos, *FP* aos falsos positivos e *FN* aos falsos negativos.

Definiu-se que o rótulo “positivo” está vinculado a presença de plágio em um par de códigos-fontes comparados, enquanto o “negativo” indica a ausência. O *VP* mostra a quantidade total de pares em que o detector analisado identificou um plágio existente. Ao contrário do *FP* que indica a quantidade total de pares em que houve uma detecção de plágio não existente, e o *FN* que contabiliza os casos de ausência de plágio que não foram detectados.

$$Precisão = P = \left(\frac{VP}{VP + FP}\right), \in [0, 1] \tag{3}$$

$$Revocação = R = \left(\frac{VP}{VP + FN}\right), \in [0, 1] \tag{4}$$

$$Medida\ F = F = 2 * \left(\frac{P * R}{P + R}\right), \in [0, 1] \tag{5}$$

Observa-se que 148 (centro e quarenta e oito) de 168 (cento e sessenta e oito) pares de códigos-fontes de testes possuem apenas exemplos de plágio, portanto, levando apenas em consideração o rótulo “positivo”. Apenas 20 (vinte) de 168 (cento e sessenta e oito) pares são designados a testarem casos de ausência de plágio indicados pelo rótulo “negativo”.

Considerando esse contexto dos cenários de testes, outros parâmetros avaliativos não são utilizados. Aqueles que possuem associação a variável de verdadeiros negativos (*VN*) não são indicados neste trabalho, apesar de o cenário de teste G avaliar apenas casos de ausência de plágio.

Dentre os parâmetros considerados, o objetivo da precisão é indicar a capacidade da solução avaliada em identificar casos de plágio que realmente existam.

Para a revocação é mostrar a eficiência na detecção de todos tipos de plágios existentes, sem considerar falsos positivos. Por outro lado, a *medida F* contabiliza o desempenho geral da solução considerando a taxa de precisão e revocação.

Os resultados encontrados de cada parâmetro avaliativo para as ferramentas: BIOPLAG, MOSS e JPLAG, são apresentados na Tabela 5. Foram considerados 336 (trezentos e trinta e seis) códigos-fontes organizados em 168 (cento e sessenta e oito) pares representando 7 (sete) cenários de testes. Cada um dos pares nos cenários representa um teste diferente, conforme apresentados na primeira seção deste capítulo.

As opções padrões de configuração de cada ferramenta foram utilizados para os testes. A versão do JPLAG é a “2.11.8”, e seus resultados foram obtidos por meio de correspondências classificadas por similaridade média. Para a versão do MOSS foram consideradas as atualizações até 14 de dezembro de 2018.

Tabela 5 - Resultados dos parâmetros avaliativos para a BIOPLAG, MOSS e JPLAG

Cenário	Detector	*VP*	*FP*	*FN*	*Precisão*	*Revocação*	*Medida F*
A	BioPlag	26	0	0	1	1	1
B	BioPlag	24	0	0	1	1	1
C	BioPlag	19	0	6	1	0,76	0,863636
D	BioPlag	25	0	2	1	0,925926	0,961538
E	BioPlag	21	0	3	1	0,875	0,933333
F	BioPlag	17	0	5	1	0,772727	0,871795
G	BioPlag	20	0	0	1	1	1
A	JPLAG	25	0	1	1	0,961538	0,980392
B	JPLAG	24	0	0	1	1	1
C	JPLAG	18	0	7	1	0,72	0,837209
D	JPLAG	19	0	8	1	0,703704	0,826087
E	JPLAG	10	0	14	1	0,416667	0,588235
F	JPLAG	12	0	10	1	0,545455	0,705882
G	JPLAG	20	0	0	1	1	1
A	MOSS	25	0	1	1	0,961538	0,980392
B	MOSS	23	0	1	1	0,958333	0,978723
C	MOSS	19	0	6	1	0,76	0,863636
D	MOSS	18	0	9	1	0,666667	0,80
E	MOSS	5	0	19	1	0,208333	0,344828
F	MOSS	8	0	14	1	0,363636	0,533333
G	MOSS	20	0	0	1	1	1

Fonte: Autoria própria

Analisando o parâmetro precisão, identifica-se que todas as ferramentas conseguiram pontuação máxima. Esse desempenho indica que todos os plágios identificados eram realmente casos existentes. Para o último cenário, a precisão foi comprovada ao avaliar a ausência de plágio em todos os seus testes.

Para a revocação, a BIOPLAG apresentou como resultado médio o valor de 0,90481 superando a média geral incluindo as três ferramentas que apresentou a pontuação de 0,79045. Por outro lado, o resultado médio do MOSS e do JPLAG ficaram abaixo da média geral, apresentando os valores: 0,70264 e 0,76391, respectivamente.

Dentre as menores pontuações obtidas para a revocação em cada ferramenta, a BIOPLAG apresentou o maior valor: 0,76. O MOSS e o JPLAG apresentaram os valores: 0,20833 e 0,41667, respectivamente. Observa-se que a BIOPLAG consegue um aproveitamento 34,33% melhor do que a segunda maior pontuação.

A ferramenta que alcançou mais vezes a pontuação máxima de revocação nos cenários de testes foi a BIOPLAG, sendo em 3 de 7. O MOSS alcançou em 1 de 7 e o JPLAG em 2 de 7. Em valor médio de revocação nos 7 cenários, a BIOPLAG apresentou o maior aproveitamento com a taxa de 90,48%, sendo 14,09% melhor do que a segunda melhor média alcançada pelo JPLAG.

O desempenho da *medida F* foi melhor com a ferramenta BioPlag que apresentou as seguintes pontuações: média de 0,94719, máxima de 1 e mínima de 0,86364. Em comparação, JPLAG apresentou média de 0,84826, máxima de 1 e mínima de 0,58824. Com o MOSS, média de 0,78584, máxima de 1 e mínima de 0,34483.

A análise geral do desempenho das ferramentas acerca da detecção de plágio em códigos-fontes pode ser encontrada no Gráfico 7. Nesse levantamento estatístico é feita a consideração dos resultados apresentados nos 7 (sete) cenários de testes para os 3 (três) parâmetros avaliativos. A formulação da análise considera as seguintes medidas quantitativas: primeiro quartil, segundo quartil, terceiro quartil, média, limite superior, limite inferior e discrepância.

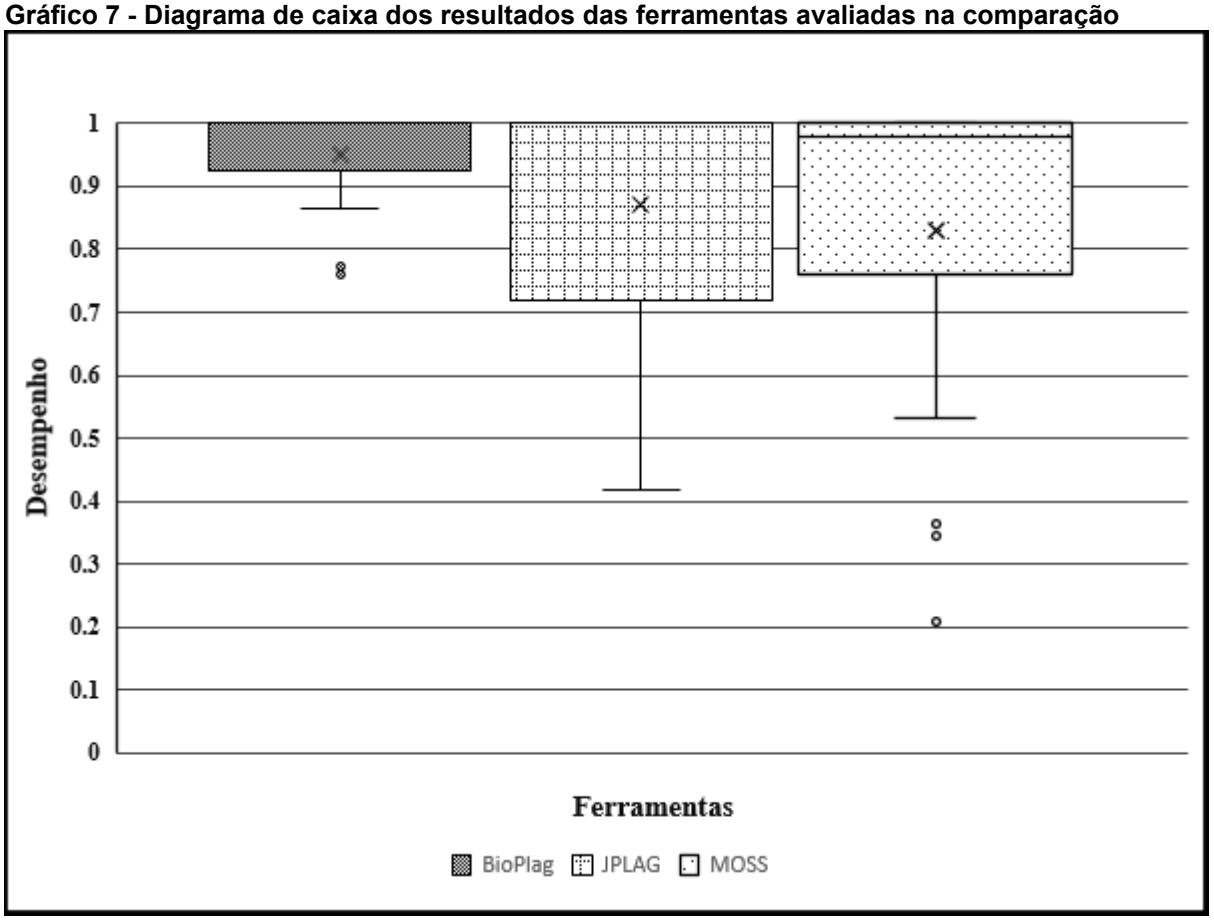
A BIOPLAG apresentou em 25% dos seus resultados pontuações menores ou iguais a aproximadamente 0,925926. Em 50% dos seus resultados obteve pontuações máximas e em 25% alcançou valores entre o intervalo inclusivo 0,925926 e 1. A média foi de 0,950665 e a mediana 1, sendo o limite superior 1 e o limite inferior

0,863636. Ressalta-se que o limite inferior foi afetado pela presença de dois resultados discrepantes: 0,76 e 0,772727.

O JPLAG em 25% dos seus resultados apresentou pontuações inferiores ou iguais a 0,72, em 50% obteve pontuações máximas e em 25% alcançou valores entre o intervalo inclusivo 0,72 e 1. A média foi de 0,870722 e a mediana 1, sendo o seu limite superior e inferior respectivamente: 1 e 0,41667. Não apresentou valores discrepantes.

Os resultados do MOSS, em 25% dos testes obteve pontuações inferiores ou iguais a 0,76, em 50% pontuações maiores ou iguais a 0,978723 e em 25% valores entre o intervalo inclusivo 0,76 e 0,978723. A média foi de 0,829496 e a mediana 0,978723, sendo o seu limite superior e inferior respectivamente: 1 e 0,208333. Apresentou três valores discrepantes: 0,20833, 0,36364 e 0,34483.

Gráfico 7 - Diagrama de caixa dos resultados das ferramentas avaliadas na comparação



Fonte: Autoria própria

Comparando o desempenho geral das ferramentas, constatou-se que a BIOPLAG alcançou as maiores pontuações e com uma menor variabilidade. Em 75% dos seus resultados as pontuações variaram entre o intervalo inclusivo 0,925926 e 1.

Diferente do JPLAG e do MOSS que apresentaram os seguintes intervalos inclusivos: 0,72 e 1, e 0,76 e 1, respectivamente.

A variação entre os resultados é menor com a BIOPLAG analisando a sua diferença de 0,049335 entre a média e a mediana. Comparando com o JPLAG e o MOSS, esse valor é 0,129278 e 0,170504, respectivamente. A diferença entre esses indicadores confirma uma maior simetria entre os resultados. Nota-se o melhor desempenho da BIOPLAG por apresentar maior simetria entre as suas pontuações, sendo que seus valores são os maiores em relação as outras ferramentas.

Analisando as menores pontuações mais frequentes de cada ferramenta, a BIOPLAG apresentou maiores pontuações sendo constatadas pelo intervalo inclusivo 0,863636 e 0,925926. Essa mesma faixa de valores para o JPLAG foi 0,72 e 0,870722, e para o MOSS foi 0,76 e 0,829496.

Observa-se que a menor pontuação da BIOPLAG foi considerada como sendo um ponto discrepante ou anômalo pelo Gráfico 7 devido a sua baixa frequência, que nesse caso foi de uma única vez e distante do limite inferior: 0,863636. Por outro lado, o JPLAG não apresentou nenhum ponto discrepante, no entanto, o seu limite inferior foi o menor de todos: 0,416667. O MOSS apresentou três pontos discrepantes e o seu limite inferior foi de 0,533333.

5.4 CONSIDERAÇÕES FINAIS DO CAPÍTULO

Este capítulo apresentou os resultados obtidos com a aplicação da BIOPLAG em diferentes cenários de testes. A aplicação da BIOPLAG foi feita a partir da implementação da abordagem criada e a sua avaliação. O processo avaliativo contou com 6 (seis) cenários diferentes de testes para detectar a presença de diferentes níveis de plágio em programação e 1 (um) cenário específico para o reconhecimento de falsos positivos e falsos negativo.

Os testes realizados utilizaram 336 (trezentos e trinta e seis) códigos-fontes criados a partir de 3 (três) experimentos. Os experimentos tiveram a participação de alunos e programadores para criar exemplos reais de plágio em programação. Ao todo foram 168 exemplos testados com o objetivo de contemplar diferentes técnicas de modificações aplicadas em códigos-fontes plagiados.

A avaliação apresentada comparou a BIOPLAG com ferramentas de referências no estado da arte: MOSS e JPLAG. Os testes foram realizados

considerando os seguintes parâmetros avaliativos: precisão, revocação e medida F. Os resultados da BIOPLAG demonstraram um desempenho igual em 3 (três) cenários de testes e superior em 4 (quatro) para todos os parâmetros comparados.

6 CONCLUSÃO

Este trabalho criou uma abordagem, nomeada BIOPLAG, capaz de realizar a detecção de níveis de plágio em códigos-fontes. Dentre as suas características, destaca-se a possibilidade de suporte a qualquer linguagem de programação. O seu resultado é uma porcentagem que indica a similaridade encontrada entre os códigos-fontes comparados. O funcionamento da abordagem foi elaborado considerando o objetivo de aprimorar a qualidade de detecção de plágio em programação. Para contemplar tal objetivo, identificou-se a necessidade de detectar diferentes técnicas de modificações em códigos-fontes, sem afetar a complexidade de tempo de execução de referência que são atingidas pelas ferramentas: JPLAG e MOSS.

Por meio da realização de um mapeamento sistemático foi possível entender o estado da arte desta área, contribuindo com orientações para o desenvolvimento da BIOPLAG. As necessidades identificadas tais como: abranger mais de uma linguagem de programação, detectar níveis de plágio e validar com diferentes cenários de testes foram contempladas.

A abordagem criada é fundamentada em um conjunto de técnicas da bioinformática e da computação. O seu funcionamento depende essencialmente de quatro etapas: geração de *tokens*, mapeamento de *tokens* em sequências biológicas, alinhamento de sequências biológicas e identificação da porcentagem de similaridade entre os códigos-fontes comparados.

Para implementar a BIOPLAG foi utilizado o gerador de *tokens*: *C Tokenize*. A partir de sua utilização foi desenvolvido o mapeamento de *tokens* em sequências biológicas do tipo de proteína. A tabela de mapeamento criada consiste em 8 (oito) *tokens* mapeados em 8 (oito) aminoácido. Durante esse processo é feita eliminação dos comentários presentes nos códigos-fontes.

O alinhamento de sequências biológicas foi desenvolvido por meio da utilização do módulo de proteínas da ferramenta de Bioinformática BLAST. Para a realização dos alinhamentos foram modificados os seguintes parâmetros da ferramenta: *word_size*, *matrix*, *threshold*, *comp_based_stats*, *seg*, *soft_masking*, *db_hard_mask*, *xdrop_gap_final*, *window_size*, *evalue* e *ungapped*. Uma das modificações se refere ao valor da matriz de substituição, implementou-se um mecanismo de escolha de diferentes valores para auxiliar a execução dos alinhamentos.

A identificação da taxa de similaridade foi realizada a partir análise do relatório gerado pela ferramenta de alinhamento. Para calcular a porcentagem indicativa da similaridade levou-se em consideração o tamanho da sequência analisada e as taxas de identidade das suas regiões alinhadas sem sobreposição. Também, definiu-se que somente regiões alinhadas com uma taxa de identidade igual ou superior a 80% poderiam ser consideradas.

Com a implementação da BIOPLAG foi realizado um processo avaliativo por meio de diferentes cenários de testes. Esse processo avaliativo utilizou 296 (duzentos e noventa e seis) códigos-fontes produzidos por meio de três experimentos reais com alunos de graduação, mestrado e programadores de uma empresa de tecnologia da informação. Além disso, para a composição do último cenário de teste foram utilizados mais 40 (quarenta) códigos-fontes escolhidos aleatoriamente a partir de uma base de códigos-fontes que serviu de apoio para os experimentos realizados.

A avaliação utilizou ao todo 336 (trezentos e trinta e seis) códigos-fontes, implementados em linguagem *C*, em sete cenários de testes diferentes. Para cada teste foram avaliados os seguintes parâmetros: precisão, revocação e medida *F*. Os resultados encontrados com a implementação da abordagem foram comparados com os detectores de plágio em programação: MOSS e JPLAG.

Os resultados obtidos pela BIOPLAG indicam que a abordagem criada foi capaz de detectar os 6 (seis) níveis de plágio em códigos-fontes categorizados por Faidhi e Robinson (1987). Considerando todos os cenários de testes, um total de 152 (90,48%) dos testes foram corretamente detectados com a presença ou ausência de plágio em programação.

Comparando os resultados com as outras ferramentas, a BIOPLAG obteve desempenho superior em 4 (quatro) cenários de testes e igual em 3 (três) deles. O seu desempenho médio acerca dos parâmetros avaliativos nos cenários de testes foi de 0,950665. Constatando ser superior ao do MOSS com 0,829496 e do JPLAG com 0,870722.

Dentre as dificuldades encontradas no decorrer do desenvolvimento deste trabalho, destaca-se aquelas relacionadas a elaboração e execução dos três experimentos reais que auxiliaram na avaliação da BIOPLAG. Acerca da elaboração, a busca por bases compartilhadas de códigos-fontes demandou um maior esforço do que o esperado, uma vez que, esse compartilhamento não é comum na área de plágio em programação.

Após definir como o experimento seria realizado, encontrou-se dificuldades acerca da sua execução. Não foram aprovados todos os códigos-fontes de exemplo produzidos pelos voluntários por não contemplarem os requisitos solicitados. As reprovações não esperadas ocasionaram a necessidade de realização de três experimentos, para que fosse contemplada a quantidade prevista de exemplos de plágio no processo avaliativo.

Por fim, encontrou-se dificuldade para elaborar o conceito de usar qualquer tipo de sequência biológica na abordagem. Para a definição desse conceito este trabalho analisou e testou situações de uso preferencial associado a quantidade de *tokens* produzidos.

6.1 TRABALHOS FUTUROS

Acerca da continuidade do desenvolvimento desta pesquisa, identifica-se a possibilidade de desenvolvimento ou aprimoramento de diferentes aspectos relacionados a BIOPLAG. Como sugestão de trabalhos futuros destacam-se:

- Realizar novos testes direcionados a avaliação de exemplos de códigos-fontes não plagiados. O objetivo é incorporar outros parâmetros avaliativos que lidam com falsos negativos e verdadeiros negativos. Por exemplo, a utilização da taxa de especificidade.
- Análise qualitativa e quantitativa direcionada aos dados indiretamente produzidos pelos três experimentos realizados. A frequência de uso das técnicas de plágio, assim como os padrões de suas utilizações são algumas das sugestões de tópicos a serem investigados;
- Testar a implementação da BIOPLAG com outras linguagens de programação suportadas, além do *C*.
- Adicionar novas funcionalidades a BIOPLAG, como a possibilidade de selecionar trechos de códigos-fontes a serem desconsiderados durante a busca por plágio.

REFERÊNCIAS

ACAMPORA, G.; COSMA, G. A fuzzy-based approach to programming language independent source-code plagiarism detection. In: INTERNATIONAL CONFERENCE ON FUZZY SYSTEMS (FUZZ-IEEE), 2015, Istanbul. **Proceedings…** Istanbul: IEEE, 2015. p. 1-8.

AGARWAL, M. R.; *et al.* Genetic sequence alignment: a comparative study of methods. In: INTERNATIONAL CONFERENCE ON COMPUTING METHODOLOGIES AND COMMUNICATION (ICCMC), 2., 2018, Erode. **Proceedings…** Erode: IEEE, 2018. p. 374-379.

AGRAWAL, M; SHARMA, D. K. A novel method to find out the similarity between source codes. In: UTTAR PRADESH SECTION INTERNATIONAL CONFERENCE ON ELECTRICAL, COMPUTER AND ELECTRONICS ENGINEERING (UPCON), 2016, Varanasi. **Proceedings…** Varanasi: IEEE, 2017. p. 339-343.

AHADI, A; MATHIESON, L. A comparison of three popular source code similarity tools for detecting student plagiarism. In: THE TWENTY-FIRST AUSTRALASIAN COMPUTING EDUCATION CONFERENCE (ACE ‘19), 21., 2019, Sydney. **Proceedings…** New York: ACM, 2019. p. 112-117.

AJMAL, O.; *et al.* EPlag: a two layer source code plagiarism detection system. In: INTERNATIONAL CONFERENCE ON DIGITAL INFORMATION MANAGEMENT (ICDIM), 8., 2013, Islamabad. **Proceedings…** Islamabad: IEEE, 2014. p. 256-261.

ALBERTS, B.; *et al.* **Biologia molecular da célula**. 6. ed. Porto Alegre: Artmed, 2017.

ALTSCHUL, S. F.; *et al.* Basic local alignment search tool. **Journal of Molecular Biology**, [S.l.], v. 215, n. 3, p. 403-410, Oct. 1990.

. Gapped blast and psi-blast: a new generation of protein database search programs. **Nucleic Acids Research**, [S.l.], v. 25, n. 17, p. 3389-3402, Sept. 1997.

ARAUJO, G. G; KYRILOV, A. Plagiarism prevention through project based learning with gitlab. **The Journal of Computing Sciences in Colleges**, San Marcos (CA), p. 53-57, Mar. 2020.

ARAÚJO, N. D.; *et al.* A era da bioinformática: seu potencial e suas implicações para as ciências da saúde. **Estudos de biologia**, Curitiba (PR), v. 30, n. 70/72, p. 143-148, jan. 2008.

ARWIN, C; TAHAGHOGHI, S. M. M. Plagiarism detection across programming languages. In: AUSTRALIAN COMPUTER SOCIETY, 29, 2006, Darlinghurst. **Proceedings of the 29th Australasian Computer Science Conference**. Darlinghurst: ACM, 2006, v. 48, p. 277-286.

ARYA, G. P.; *et al.* An improved method for dna sequence compression. In: INTERNATIONAL CONFERENCE ON TELECOMMUNICATION AND NETWORKS (TEL-NET), 2., 2017, Noida. **Proceedings…** Noida: IEEE, 2018. p. 1-4.

BABY, J.; *et al.* Distance indices for the detection of similarity in c programs. In: International Conference on Computation of Power, Energy, Information and Communication (ICCPEIC), 2014, Chennai. **Proceedings…** Chennai: IEEE, 2014. p. 462-467.

BIN-HABTOOR, A. S; ZAHER, M. A. A survey on plagiarism detection systems. **International Journal of Computer Theory and Engineering**, Singapore (SG), v. 4, n. 2, p. 185, Apr. 2012.

BORDINI, R. H; HÜBNER, J. F; WOOLDRIDGE, M. **Programming Multi-Agent Systems in AgentSpeak Using Jason (Wiley Series in Agent Technology)**. Chichester: John Wiley & Sons, 2007.

BOTH, D. Always use shell scripts. In: BOTH, D. **The linux philosophy for sysadmins**. Berkeley: Springer, 2018. p. 195-215.

BRERETON, P.; *et al.* Lessons from applying the systematic literature review process within the software engineering domain. **Journal of Systems and Software**, [S.l.], v. 80, n. 4, p. 571-583, Apr. 2007.

BUCHINGER, D; CAVALCANTI, G. A. S; HOUNSELL, M. S. Mecanismos de busca acadêmica: uma análise quantitativa. **Revista Brasileira de Computação Aplicada**, Passo Fundo (RS), v. 6, n. 1, p. 108-120, abr. 2014.

BURROWS, S; TAHAGHOGHI, S. M. M; ZOBEL, J. Efficient plagiarism detection for large code repositories. **Software: Practice and Experience**, [S.l.], v. 37, n. 2, p. 151-175, Sept. 2007.

CAMBRIDGE DICTIONARY. Disponível em: https://dictionary.cambridge.org/pt/dicionario/portugues-ingles/plagio. Acesso em: 9 maio 2020.

CHAN, P. P. F; HUI, L. C. K; YIU, S. M. Heap graph based software theft detection. **IEEE Transactions on Information Forensics and Security**, [S.l.], v. 8, n. 1, p. 101-110, Jan. 2013.

CHOI, J. C.; *et al.* A survey of feature extraction techniques to detect the theft of windows applications. In: INTERNATIONAL CONFERENCE ON INNOVATIVE MOBILE AND INTERNET SERVICES IN UBIQUITOUS COMPUTING (IMIS), 7., 2013, Taichung. **Proceedings…** Taichung: IEEE, 2013. p. 723-728.

CHUDA, D.; *et al.* The issue of (software) plagiarism: A student view. **IEEE Transactions on Education**, [S.l.], v. 55, n. 1, p. 22-28, Feb. 2012.

COSMA, G; JOY, M. An approach to source-code plagiarism detection and investigation using latent semantic analysis. **IEEE Transactions on Computers**, [S.l.], v. 61, n. 3, p. 379-394, Mar. 2012.

COULL, S.; *et al.* Intrusion detection: a bioinformatics approach. In: ANNUAL COMPUTER SECURITY APPLICATIONS CONFERENCE, 19., 2003, Las Vegas. **Proceedings...** Las Vegas: IEEE, 2004. p. 24-33.

CRUZ, A. R.; *et al.* On the importance of lexicon, structure and style for identifying source code plagiarism. In: FORUM FOR INFORMATION RETRIEVAL EVALUATION, 2., 2014, Bangalore. **Proceedings...** New York: ACM, 2014. p. 31-38.

CTOKENIZE. Disponível em: https://metacpan.org/pod/distribution/C-Tokenize/lib/C/Tokenize.pod#VERSION. Acesso em: 6 jul. 2020.

DAVID, W. M. **Bioinformatics:** sequence and genome analysis. New York: CSHL Press, p. 75-85, 2001.

DILI, U. E. U. C. P.; *et al.* Development of a software on distance education applications for compilation and plagiarism detection of c programming language assignments. **Journal of Science and Engineering Science**, [S.l.], v. 26, n. 1, p. 98-106, 2011.

DONALDSON, J. L; LANCASTER, A. M; SPOSATO, P. H. A plagiarism detection system. In: SIGCSE Technical symposium on computer science education, 20., 1981, New York. **Proceedings...** New York: ACM, 1981. v. 13, n. 1, p. 21–25.

ĐURIĆ, Z; GAŠEVIĆ, D. A source code similarity system for plagiarism detection. **The Computer Journal**, [S.l.], v. 56, n. 1, p. 70-86, Jan. 2013.

DUTTA, R. Efficient approach to detect logical equivalence in the paradigm of software plagiarism. In: INTERNATIONAL CONFERENCE ON COMPUTER, COMMUNICATION, CONTROL AND INFORMATION TECHNOLOGY (C3IT), 3., 2015, Hooghly. **Proceedings...** Hooghly: IEEE, 2015. p. 1-5.

FAIDHI, J. A. W; ROBINSON, S. K. An empirical approach for detecting program similarity and plagiarism within a university programming environment. **Computers & Education Journal Elsevier**, Elmsford (NY), v. 11, n. 1, p. 11-19, 1987.

FASSLER, J; COOPER, P. Blast glossary. In: BLAST HELP, 2011, Bethesda. **Proceedings...** Bethesda: NCBI, 2011. p. 1-9.

FLORES, E.; *et al*. DeSoCore: detecting source code re-use across programming languages. In: ASSOCIATION FOR COMPUTATIONAL LINGUISTICS, 2012, Stroudsburg. **Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Demonstration Session**. Stroudsburg: ACM, 2012. p. 1-4.

FOY, B. D. **Learning Perl 6**: keeping the easy, hard, and impossible within reach. 1. ed. Sebastopol: O'Reilly Media Inc., 2018.

FRANÇA, A. B. **O código por trás do código-fonte:** mudança de representação para a análise de similaridade. 2019. 99 f. Tese (Doutorado em Engenharia de Teleinformática) – Universidade Federal do Ceará, Fortaleza, 2019. Disponível em: http://repositorio.ufc.br/bitstream/riufc/44452/3/2019_tese_abfranca.pdf. Acesso em: 6 jul. 2017.

ALLYSON, F. B.; *et al*. Sherlock n-overlap: invasive normalization and overlap coefficient for the similarity analysis between source code. **IEEE Transactions on Computers**, [S.l.], v. 68, n. 5, p. 740-751, May. 2018.

GOAD, W. B; KANEHISA, M. I. Pattern recognition in nucleic acid sequences. I. A general method for finding local homologies and symmetries. **Nucleic Acids Research**, London (UK), v. 10, n. 1, p. 247-263, Jan. 1982.

GOMES, K. P. **Aplicação de bioinformática para reconhecimento de plágio em códigos de programação. Dissertação**. 2017. 66 f. Trabalho de Conclusão de Curso (Graduação em Ciência da Computação) – Universidade Tecnológica Federal do Paraná, Ponta Grossa, 2017. Disponível em: http://repositorio.roca.utfpr.edu.br/jspui/bitstream/1/8284/1/PG_COCIC_2017_2_02.pdf. Acesso em: 9 maio 2020.

GOMES, K. P; MATOS, S. N. Contributions of bioinformatics for computing education in the detection of programming assignment plagiarism. In: Congresso Brasileiro de Informática na Educação, 30., 2019, Brasília. **Anais do Simpósio Brasileiro de Informática na Educação**. Brasília: SBC, 2019. v. 30, n. 1, p. 1351-1360.

GOOGLE SCHOLAR. Disponível em: https://scholar.google.com.br/. Acesso em: 9 maio 2020.

HUNTER, L. E. **The processes of life:** an introduction to molecular biology. Cambridge: MIT Press, 2009.

INOUE, U; WADA, S. Detecting plagiarisms in elementary programming courses. In: INTERNATIONAL CONFERENCE ON FUZZY SYSTEMS AND KNOWLEDGE DISCOVERY (FSKD), 9., 2012, Sichuan. **Proceedings...** Sichuan, 2012. p. 2308-2312.

JAIN, S.; *et al*. CPLAG: efficient plagiarism detection using bitwise operations. In: INTERNATIONAL CONFERENCE ON CONTEMPORARY COMPUTING (IC3), 10., 2017, Noida. **Proceedings...** Noida: IEEE, 2017. p. 1-5.

JHI, Y. C.; *et al*. Program characterization using runtime values and its application to software plagiarism detection. **IEEE Transactions on Software Engineering**, [S.l.], v. 41, n. 9, p. 925-943, Sept. 2015.

JOY, M; LUCK, M. Plagiarism in programming assignments. **IEEE Transactions on Education**, [S.l.], v. 42, n. 2, p. 129-133, 1999.

JUNIOR, Q; DIAS, W. **ESSEX:** identificação de um aminoácido de interesse em sequências biológicas de origens diferentes. 2019, 97 f. Dissertação (Mestrado em

Engenharia de Computação) – Universidade Federal do Rio Grande, Rio Grande, 2019. Disponível em: http://repositorio.furg.br/bitstream/handle/1/8160/21.pdf?sequence=1. Acesso em: 9 maio 2020.

KANE, M. D; SPRINGER, J. A. Integrating bioinformatics, distributed data management, and distributed computing for applied training in high performance computing. In: Information Technology Education Conference, 8., 2007, Destin. **Proceedings of the Special Interest Group for Information Technology Education (SIGITE)**. New York: ACM, 2007. p. 33-36.

KARGÉN, U; SHAHMEHRI, N. Towards robust instruction-level trace alignment of binary code. In: IEEE/ACM INTERNATIONAL CONFERENCE ON AUTOMATED SOFTWARE ENGINEERING (ASE), 32., 2017, Urbana. **Proceedings...** Urbana: IEEE, 2017. p. 342-352.

KARNALIM, O. Detecting source code plagiarism on introductory programming course assignments using a bytecode approach. In: INTERNATIONAL CONFERENCE ON INFORMATION & COMMUNICATION TECHNOLOGY AND SYSTEMS (ICTS), 2016, Surubaya. **Proceedings...** Surubaya: IEEE, 2017. p. 63-68.

. An abstract method linearization for detecting source code plagiarism in object-oriented environment. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING AND SERVICE SCIENCE (ICSESS), 8., 2017, Beijing. **Proceedings...** Beijing: IEEE, 2018. p. 58-61.

KIKUCHI, H.; *et al*. A source code plagiarism detecting method using alignment with abstract syntax tree elements. In: IEEE/ACIS INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, ARTIFICIAL INTELLIGENCE, NETWORKING AND PARALLEL/DISTRIBUTED COMPUTING (SNPD), 15., 2014, Las Vegas. **Proceedings...** Las Vegas: IEEE, 2014. p. 1-6.

KIM, Y.; *et al*. A static birthmark of windows binary executables based on strings. In: INTERNATIONAL CONFERENCE ON INNOVATIVE MOBILE AND INTERNET SERVICES IN UBIQUITOUS COMPUTING (IMIS), 7., 2013, Taichung. **Proceedings...** Taichung: IEEE, 2013. p. 734-738.

KITCHENHAM, B.; *et al*. Systematic literature reviews in software engineering–a systematic literature review. **Information and Software Technology Journal Elsevier**, [S.l.], v. 51, n. 1, p. 7-15, Jan. 2009.

KUO, J. Y; CHENG, H. K; WANG, P. F. Program plagiarism detection with dynamic structure. In: INTERNATIONAL SYMPOSIUM ON NEXT GENERATION ELECTRONICS (ISNE), 7., 2018, Taipei. **Proceedings...** Taipei: IEEE, 2018. p. 1-3.

LANCASTER, T; CULWIN, F. A comparison of source code plagiarism detection engines. **Computer Science Education**, [S.l.], v. 14, n. 2, p. 101-112, 2004.

LAZAR, F. M; BANIAS, O. Clone detection algorithm based on the abstract syntax tree approach. In: INTERNATIONAL SYMPOSIUM ON APPLIED COMPUTATIONAL INTELLIGENCE AND INFORMATICS (SACI), 9., 2014, Timisoara. **Proceedings...** Timisoara: IEEE, 2014. p. 73-78.

LEACH, R. J. Using metrics to evaluate student programs. **ACM SIGCSE Bulletin**, [S.l.], v. 27, n. 2, p. 41-43, June 1995.

LI, Y.; *et al*. A similarity detection platform for programming learning. In: INTERNATIONAL CONFERENCE ON COMPUTER SUPPORTED EDUCATION (CSEDU), 7., 2015, [S.l.]. **Proceedings...** [S.l.]: SCITEPRESS, 2015. p. 480-485.

LIU, K; ZHENG, T; WEI, L. A software birthmark based on system call and program data dependence. In: WEB INFORMATION SYSTEM AND APPLICATION CONFERENCE (WISA), 11., 2014, Tianjin. **Proceedings...** Tianjin: IEEE, 2014. p. 105-110.

LUO, L.; *et al*. Semantics-based obfuscation-resilient binary code similarity comparison with applications to software and algorithm plagiarism detection. **IEEE Transactions on Software Engineering**, [S.l.], v. 43, n. 12, p. 1157-1177, Dec. 2017.

MAKADY, S; WALKER, R. J. Test code reuse from oss: current and future challenges. In: AFRICA AND MIDDLE EAST CONFERENCE ON SOFTWARE ENGINEERING (AMECSE), 3., 2017, [S.l.]. **Proceedings...** [S.l.]: ACM, 2017. p. 31-36.

MASON, T; GAVRILOVSKA, A; JOYNER, D. A. Collaboration versus cheating: reducing code plagiarism in an online ms computer science program. In: SPECIAL INTEREST GROUP ON COMPUTER SCIENCE EDUCATION (SIGCSE), 50., 2019, Minneapolis. **Proceedings of the Technical Symposium on Computer Science Education**. New York: ACM, 2019. p. 1004-1010.

MAURER, H. A; KAPPE, F; ZAKA, B. Plagiarism-a survey. **Journal of Universal Computer Science**, [S.l.], v. 12, n. 8, p. 1050-1084, Aug. 2006.

MCGINNIS, S; MADDEN, T. L. Blast: at the core of a powerful and diverse set of sequence analysis tools. **Nucleic Acids Research**, [S.l.], v. 32, n. suppl_2, p. W20-W25, July 2004.

MENAI, M. E. B; AL-HASSOUN, N. S. Similarity detection in java programming assignments. In: INTERNATIONAL CONFERENCE ON COMPUTER SCIENCE AND EDUCATION (ICCSE), 5., 2010, Hefei. **Proceedings...** Hefei: IEEE, 2010. p. 356-361.

MEUSCHKE, N.; *et al*. An adaptive image-based plagiarism detection approach. In: ACM/IEEE ON JOINT CONFERENCE ON DIGITAL LIBRARIES (JCDL), 18., 2018, Fort Worth. **Proceedings...** New York: ACM, 2018. p. 131-140.

MING, J.; *et al*. Deviation-based obfuscation-resilient program equivalence checking with application to software plagiarism detection. **IEEE Transactions on Reliability**, [S.l.], v. 65, n. 4, p. 1647-1664, Dec. 2016.

MIRZA, O. M; JOY, M; COSMA, G. Style analysis for source code plagiarism detection — an analysis of a dataset of student coursework. In: INTERNATIONAL CONFERENCE ON ADVANCED LEARNING TECHNOLOGIES (ICALT), 17., 2017, Timisoara. **Proceedings...** Timisoara: IEEE, 2017. p. 296-297.

MIŠIĆ, M. J; PROTIĆ, J. Ž; TOMAŠEVIĆ, M. V. Improving source code plagiarism detection: lessons learned. In: Telecommunication Forum (TELFOR), 25., 2017, Belgrade. **Proceedings...** Belgrade: IEEE, 2017. p. 1-8.

MOSS. Disponível em: https://theory.stanford.edu/~aiken/moss/. Acesso em: 9 maio 2020.

MOU, L; LI, G; ZHANG, L; WANG, T; JIN, Z. Convolutional neural networks over three structures for programming language processing. In: THE THIRTIETH AAAI CONFERENCE ON ARTIFICIAL INTELLIGENCE (AAAI-16), 30., 2016, Phoenix. **Proceedings…** Palo Alto: AAAI, 2016. p. 1287-1293.

MOZGOVOY, M; KARAKOVSKIYZ, S; KLYUEV, V. Fast and reliable plagiarism detection system. In: ANNUAL FRONTIERS IN EDUCATION CONFERENCE - GLOBAL ENGINEERING: KNOWLEDGE WITHOUT BORDERS, OPPORTUNITIES

WITHOUT PASSPORTS, 37., 2007, Milwaukee. **Proceedings...** Milwaukee: IEEE, 2008. p. S4H-11-S4H-14.

MUSHTAQ, Z; RASOOL, G; SHEHZAD, B. Multilingual source code analysis: a systematic literature review. **IEEE Access**, [S.l.], v. 5, p. 11307-11336, June 2017.

NARAYANAN, S; SIMI, S. Source code plagiarism detection and performance analysis using fingerprint based distance measure method. In: INTERNATIONAL CONFERENCE ON COMPUTER SCIENCE & EDUCATION (ICCSE), 7., 2012, Melbourne. **Proceedings...** Melbourne: IEEE, 2012. p. 1065-1068.

NCBI. Disponível em: https://blast.ncbi.nlm.nih.gov/Blast.cgi. Acesso em: 6 jul. 2020.

NICHOLS, L.; *et al*. Syntax-based improvements to plagiarism detectors and their evaluations. In: INNOVATION AND TECHNOLOGY IN COMPUTER SCIENCE EDUCATION, 24., 2019, Aberdeen. **Proceedings...** New York: ACM, 2019. p. 555-561.

NILSSON, R. H.; *et al*. Five simple guidelines for establishing basic authenticity and reliability of newly generated fungal its sequences. **MycoKeys**, [S.l.], v. 4, p. 37-63, 2012.

NOH, S. Y. An xml plagiarism detection model for procedural programming languages. In: IOWA STATE UNIVERSITY DIGITAL REPOSITORY, 2003, Ames. **Technical Reports of Computer Science**. Ames: Iowa State University, 2003. p. 1-12.

NOVAK, M. Review of source-code plagiarism detection in academia. In: INTERNATIONAL CONVENTION ON INFORMATION AND COMMUNICATION TECHNOLOGY, ELECTRONICS AND MICROELECTRONICS (MIPRO), 39., 2016, Opatija. **Proceedings...** Opatija: IEEE, 2016. p. 796-801.

OLIVEIRA, A. M.; *et al*. **Um método de detecção de plágio em códigos-fonte para disciplinas iniciais de programação**. 2016. 64 f. Dissertação (Mestrado em Ciência da Computação) – Universidade Federal do Amazonas, Manaus, 2016. Disponível em: http://200.129.163.131:8080/bitstream/tede/5666/5/Disserta%c3%a7%c3%a3o%20-%20Adria%20M.%20Oliveira.pdf . Acesso em: 9 maio 2020.

OPRIŞA, C; IGNAT, N. A measure of similarity for binary programs with a hierarchical structure. In: INTERNATIONAL CONFERENCE ON INTELLIGENT COMPUTER COMMUNICATION AND PROCESSING (ICCP), 2015, Cluj-Napoca. **Proceedings...** Cluj-Napoca: IEEE, 2015. p. 117-123.

ORABI, E. S.; *et al*. Dna fingerprint using smith waterman algorithm by grid computing. In: INTERNATIONAL CONFERENCE ON INFORMATICS AND SYSTEMS, 9., 2014, Cairo. **Proceedings...** Cairo: IEEE, 2015. p. PDC-74-PDC-79.

OTTENSTEIN, K. J. An algorithmic approach to the detection and prevention of plagiarism. **ACM SIGCSE Bulletin**, [S.l.], v. 8, n. 4, p. 30-41, Dec. 1976.

PAGANI, R. N; KOVALESKI, J. L; RESENDE, L. M. Methodi ordinatio: a proposed methodology to select and rank relevant scientific papers encompassing the impact factor, number of citation, and year of publication. **Scientometrics**, [S.l.], v. 105, n. 3, p. 2109-2135, Sept. 2015.

PARKER, A; HAMBLEN, J. O. Computer algorithms for plagiarism detection. **IEEE Transactions on Education**, [S.l.], v. 32, n. 2, p. 94-99, May 1989.

PAWELCZAK, D. Online detection of source-code plagiarism in undergraduate programming courses. In: THE STEERING COMMITTEE OF THE WORLD CONGRESS IN COMPUTER SCIENCE, COMPUTER, 2013, [S.l.]. **Proceedings of the International Conference on Frontiers in Education: Computer Science and Computer Engineering (FECS)**. [S.l.: s.n.], 2013. p. 1-7.

PEARSON, W. R; LIPMAN, D. J. Improved tools for biological sequence comparison. **National Academy of Sciences**, [S.l.], v. 85, n. 8, p. 2444-2448, Apr. 1988.

PEDERSEN, J.; *et al*. Blast your way through malware analysis assisted by bioinformatics tools. In: THE 2012 WORLD CONGRESS IN COMPUTER SCIENCE, COMPUTER ENGINEERING AND APPLIED COMPUTING (WORLDCOMP), 6., 2012, Las Vegas. **Proceedings of the International Conference on Security and Management (SAM)**. Las Vegas: [s.n.], 2012. p. 1-7.

POHUBA, D; DULIK, T; JANKU, P. Automatic evaluation of correctness and originality of source codes. In: EUROPEAN WORKSHOP ON MICROELECTRONICS EDUCATION (EWME), 10., 2014, Tallinn. **Proceedings...** Tallinn: IEEE, 2014. p. 49-52.

POON, J. Y. H.; *et al*. Instructor-centric source code plagiarism detection and plagiarism corpus. In: CONFERENCE ON INNOVATION AND TECHNOLOGY IN COMPUTER SCIENCE EDUCATION, 17, 2012, Haifa. **Proceedings...** New York: ACM, 2012. p. 122-127.

POTTHAST, M.; *et al*. An evaluation framework for plagiarism detection. In: INTERNATIONAL CONFERENCE ON COMPUTATIONAL LINGUISTICS: POSTERS (COLING), 23., 2010, Stroudsburg. **Proceedings...** Stroudsburg: ACM, 2010. p. 997-1005.

PRADO, B; BISPO, K. A; ANDRADE, R. X9: An obfuscation resilient approach for source code plagiarism detection in virtual learning environments. In: ICEIS, 1., 2018, [S.l.]. **Proceedings...** [S.l.: s.n.], 2018. p. 517-524.

PRECHELT, L; MALPOHL, G; PHILIPPSEN, M. Finding plagiarisms among a set of programs with jplag. **Journal of Universal Computer Science**, [S.l.], v. 8, n. 11, p. 1016-1038, Nov. 2002.

PROSDOCIMI, F.; *et al*. Bioinformática: manual do usuário. In: BIOTECNOLOGIA CIÊNCIA & DESENVOLVIMENTO, 29., 2002, [S.l.]. **Proceedings...** [S.l.: s.n.], 2002. p. 12-25.

RAGKHITWETSAGUL, C. Measuring code similarity in large-scaled code corpora. In: INTERNATIONAL CONFERENCE ON SOFTWARE MAINTENANCE AND EVOLUTION (ICSME), 2016, Raleigh. **Proceedings...** Raleigh: IEEE, 2017. p. 626-630.

RATTAN, D; BHATIA, R; SINGH, M. Software clone detection: a systematic review. **Information and Software Technology**, [S.l.], v. 55, n. 7, p. 1165-1199, July 2013.

REVETT, K. A bioinformatics based approach to user authentication via keystroke dynamics. **International Journal of Control, Automation and Systems**, [S.l.], v. 7, n. 1, p. 7-15, Mar. 2009.

ROOPAM; SINGH, G. To enhance the code clone detection algorithm by using hybrid approach for detection of code clones. In: INTERNATIONAL CONFERENCE ON INTELLIGENT COMPUTING AND CONTROL SYSTEMS (ICICCS), 2017, Madurai. **Proceedings...** Madurai: IEEE, 2018. p. 192-198.

ROY, C. K; CORDY, J. R. A survey on software clone detection research. **Queen's**

School of Computing TR, [S.l.], v. 541, n. 115, p. 64-68, Sept. 2007.

SAITOU, N. Homology search and multiple alignment. In: SAITOU, N. **Introduction to Evolutionary Genomics**. 2. ed. Basel: Springer, 2018. p. 325-360.

SCHLEIMER, S; WILKERSON, D. S; AIKEN, A. Winnowing: local algorithms for document fingerprinting. In: INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, 2003, San Diego. **Proceedings of the SIGMOD**. New York: ACM, 2003. p. 76-85.

SCHNEIDER, J.; *et al*. Detecting plagiarism based on the creation process. **IEEE Transactions on Learning Technologies**, [S.l.], v. 11, n. 13, p. 348-361, June 2017.

SHANMUGHASUNDARAM, M; SUBRAMANI, S. A measurement of similarity to identify identical code clones. **International Arab Journal of Information Technology**, [S.l.], v. 12, n. 6A, p. 735-740, Aug. 2015.

SHARMA, S; SHARMA, C. S; TYAGI, V. Plagiarism detection tool “parikshak”. In: INTERNATIONAL CONFERENCE ON COMMUNICATION, INFORMATION & COMPUTING TECHNOLOGY (ICCICT), 2015, Mumbai. **Proceedings...** Mumbai: IEEE, 2015. p. 1-7.

SOH, C.; *et al*. Detecting clones in android applications through analyzing user interfaces. In: INTERNATIONAL CONFERENCE ON PROGRAM COMPREHENSION, 23., 2015, Florence. **Proceedings...** Florence: IEEE, 2015. p. 163-173.

SON, J. W.; *et al*. An application for plagiarized source code detection based on a parse tree kernel. **Engineering Applications of Artificial Intelligence**, [S.l.], v. 26, n. 8, p. 1911-1918, Sept. 2013.

SRAKA, D; KAUCIC, B. Source code plagiarism. In: INTERNATIONAL CONFERENCE ON INFORMATION TECHNOLOGY INTERFACES, 31., 2009, Dubrovnik. **Proceedings...** Dubrovnik: IEEE, 2009. p. 461-466.

STRILEȚCHI, C.; *et al*. A cross-platform solution for software plagiarism detection. In: INTERNATIONAL SYMPOSIUM ON ELECTRONICS AND TELECOMMUNICATIONS (ISETC), 12., 2016, Timisoara. **Proceedings...** Timisoara: IEEE, 2016. p. 141-144.

SUDHAMANI, M; RANGARAJAN, L. Code clone detection based on order and content of control statements. In: INTERNATIONAL CONFERENCE ON CONTEMPORARY COMPUTING AND INFORMATICS (IC3I), 2., 2016, Noida. **Proceedings...** Noida: IEEE, 2017. p. 59-64.

SULISTIANI, L; KARNALIM, O. Es-plag: efficient and sensitive source code plagiarism detection tool for academic environment. **Computer Applications in Engineering Education**, [S.l.], v. 27, n. 1, p. 166-182, Jan. 2019.

TANG, Y; XIAO, B; LU, X. Using a bioinformatics approach to generate accurate exploit-based signatures for polymorphic worms. **Computers & Security**, [S.l.], v. 28, n. 8, p. 827-842, Nov. 2009.

TATUSOVA, T. A; MADDEN, T. L. Blast 2 sequences, a new tool for comparing protein and nucleotide sequences. **FEMS microbiology letters**, [S.l.], v. 174, n. 2, p. 247-250, May 1999.

TIAN, Z.; *et al*. Dkisb: Dynamic key instruction sequence birthmark for software plagiarism detection. In: INTERNATIONAL CONFERENCE ON HIGH PERFORMANCE COMPUTING AND COMMUNICATIONS & 2013 IEEE INTERNATIONAL CONFERENCE ON EMBEDDED AND UBIQUITOUS COMPUTING (HPCC_EUC), 10., 2013, Zhangjiajie. **Proceedings...** Zhangjiajie: IEEE, 2013. p. 619-627.

. Software plagiarism detection with birthmarks based on dynamic key instruction sequences. **IEEE Transactions on Software Engineering**, [S.l.], v. 41, n. 12, p. 1217-1235, Dec. 2015.

TICONA, W. G. C. **Aplicação de algorítmos genéticos multi-objetivo para alinhamento de seqüências biológicas**. 2003. 112 f. Dissertação (Mestrado em Ciências) – Universidade de São Paulo, São Carlos, 2003.

ULLAH, F.; *et al*. Plagiarism detection in students' programming assignments based on semantics: multimedia e-learning based smart assessment methodology. **Multimedia Tools and Applications**, [S.l.], v. 79, n. 17-18, p. 1-18, Mar. 2018.

VERCO, K. L; WISE, M. J. Plagiarism à la mode: a comparison of automated systems for detecting suspected plagiarism. **The Computer Journal**, [S.l.], v. 39, n. 9, p. 741-750, Jan. 1996.

VERLI, H (Org). **Bioinformática:** da biologia à flexibilidade molecular. 1. ed. São Paulo: Sociedade Brasileira de Bioquímica e Biologia Molecular, 2014.

WANG, H; ZHONG, J; ZHANG, D. A duplicate code checking algorithm for the programming experiment. In: INTERNATIONAL CONFERENCE ON MATHEMATICS AND COMPUTERS IN SCIENCES AND IN INDUSTRY (MCSI), 2., 2015, Sliema. **Proceedings...** Sliema: IEEE, 2015. p. 39-42.

WATERMAN, M. S. **Introduction to computational biology:** maps, sequences and genomes. [S.l.]: CRC Press, 1995.

WATSON, J. D; CRICK, F. H. C *et al*. Molecular structure of nucleic acids. **Nature**, [S.l.], v. 171, n. 4356, p. 737-738, Apr. 1953.

WISE, M. J. Yap3: Improved detection of similarities in computer program and other texts. In: SIGCSE TECHNICAL SYMPOSIUM ON COMPUTER SCIENCE EDUCATION, 27., 1996, Philadelphia. **Proceedings...** New York: ACM, 1996. v. 28, n. 1, p. 130-134.

WITTEN, I. H.; *et al*. **Managing gigabytes:** compressing and indexing documents and images. 2. ed. San Diego: Morgan Kaufmann, 1999.

XIONG, H.; *et al*. Buaa_antiplagiarism: A system to detect plagiarism for c source code. In: INTERNATIONAL CONFERENCE ON COMPUTATIONAL INTELLIGENCE AND SOFTWARE ENGINEERING, 2009, Wuhan. **Proceedings...** Wuhan: IEEE, 2009. p. 1-5.

XU, G.; *et al*. An approach to soa-based bioinformatics grid. In: ASIA-PACIFIC CONFERENCE ON SERVICES COMPUTING (APSCC), 2006, Guangzhou. **Proceedings...** Guangzhou: IEEE, 2006. p. 323-328.

ZEIDMAN, R. Software source code correlation. In: 5TH IEEE/ACIS INTERNATIONAL CONFERENCE ON COMPUTER AND INFORMATION SCIENCE AND 1ST IEEE/ACIS INTERNATIONAL WORKSHOP ON COMPONENT-BASED SOFTWARE ENGINEERING, SOFTWARE ARCHITECTURE AND REUSE (ICIS-COMSAR), 2006, Honolulu. **Proceedings...** Honolulu: IEEE, 2006. p. 383-392.

ZHANG, F.; *et al*. Program logic based software plagiarism detection. In: INTERNATIONAL SYMPOSIUM ON SOFTWARE RELIABILITY ENGINEERING (ISSRE), 25., 2014, Naples. **Proceedings...** Naples: IEEE, 2014. p. 66-77.

ZHANG, L. P; LIU, D. S. Ast-based multi-language plagiarism detection method. In: CONFERENCE ON SOFTWARE ENGINEERING AND SERVICE SCIENCE (ICSESS), 4, 2013, Beijing. **Proceedings...** Beijing: IEEE, 2013. p. 738-742.

ZHANG, Z.; *et al*. A greedy algorithm for aligning dna sequences. **Journal of Computational biology**, [S.l.], v. 7, n. 1-2, p. 203-214, July 2004.