# Detecção de Suspeita de Plágio de Códigos C para Apoio ao Ensino em Programação

**José Carlos Campana Filho**[1], **Elias Oliveira**[1],
**Márcia Gonçalves de Oliveira**[1], **Mateus Nogueira**[1]

[1]Programa de Pós Graduação em Informática
Universidade Federal do Espírito Santo (UFES) – Vitória – ES – Brazil

{zeccfilho, clickmarcia, nogueira.mateus2109}@gmail.com, elias@acm.org

***Abstract.*** *In programming classes, checking for plagiarism practices makes it harder and time consuming for a teacher to work. The pPlagio system, proposed here, assists the teacher by automatically identifying possible plagiarisms in C-language codes. pPlagio makes use of techniques for detecting various types of plagiarism through the prior processing of source codes and their similarity analysis. The system is integrated into Moodle, which facilitates the administration of classes. The pPlagio was tested with a base of codes used in a competition event called SOurce COde re-use (SOCO) and the results were better than the one obtained by the best placed in the event. Our system was superior in 3 situations when tested and compared against the Moss.*

***Resumo.*** *Em turmas de programação, a verificação de práticas de plágio torna mais difícil e demorado o trabalho de um professor. O sistema pPlagio, proposto aqui, auxilia o professor identificando possíveis plágios de códigos em linguagem C de forma automática. O pPlagio faz uso de técnicas para detectar vários tipos de plágios através de processamento prévio dos códigos-fontes e análise de similaridade destes. O sistema é integrado ao Moodle, o que facilita a administração de turmas. O pPlagio foi testado em uma base de códigos utilizada em um evento de competição chamado SOurce COde re-use (SOCO) e os resultados foram melhores que o obtido pelo melhor colocado no evento. Nosso sistema foi superior em 3 situações quando testado e comparado com o Moss.*

## 1. Introdução

O plágio de código-fonte é comum em atividades de disciplinas de programação. Uma prática adotada entre alunos é a cópia total ou parcial de soluções entre colegas, com a realização de mudanças para dificultar a percepção do plágio.

Esse problema se agrava quando as turmas são numerosas, ou quando é aplicada uma grande quantidade de exercícios. Em uma ou em outra situação, é muito custoso o trabalho para um avaliador humano analisar todos os indícios de plágios existentes nas atividades submetidas. Para se ter uma ideia da complexidade em analisar-se plágios, para se encontrar um plágio em $n$ submissões, é preciso analisar $n(n-1)/2$ pares de submissões. Em uma turma de 50 alunos, por exemplo, seria preciso analisar $50(49)/2 = 1.225$ pares de submissões.

Para auxiliar professores nesta onerosa tarefa, muitas pesquisas já foram realizadas sobre esse assunto e sistemas foram desenvolvidos para prover, de forma automática,

a análise de códigos em disciplinas de programação, tendo como objetivos a redução da carga de trabalho e a possibilidade de fornecer resultados de forma ágil. Embora tenham sido desenvolvidas ferramentas para detecção de indícios de plágio, os códigos similares encontrados são apenas indícios de plágio, podendo indicar outros aspectos. Dependendo do contexto, a semelhança pode ser indicativa de parceria, de referência de uma solução encontrada em livros ou exemplos fornecidos pelo professor, etc. Por essa razão, cabe ao professor analisar caso a caso e identificar a efetiva ocorrência ou não de plágio.

A motivação deste trabalho é auxiliar o professor no controle de submissões de trabalhos copiados, reduzindo o esforço e tempo gasto na verificação de plágios, permitindo uma maior dedicação de tempo para atividades de ensino e atendimento aos alunos.

Nesse trabalho, o Moodle foi escolhido como plataforma de submissão e exibição de resultados por ser um ambiente extensível e completo em termos de recursos para gerenciamento de atividades educacionais, sendo um bom ambiente para integrar a ferramenta de detecção de plágio em disciplinas de programação.

O objetivo deste trabalho é propor uma ferramenta, o *pPlagio*, na qual foram implementadas estratégias para detecção de alguns tipos de plágio, integrada ao Moodle, capaz de detectar, de forma automática e ágil, possíveis plágios de códigos na linguagem C para que o professor decida se o plágio realmente ocorreu.

Este trabalho está organizado da seguinte forma, na Seção 2 são apresentados os tipos de plágios. Na Seção 3 são listados os trabalhos relacionados. Na Seção 4 é explicado o modelo proposto de detecção de tipos de plágio. Na Seção 5 são relatados os experimentos. Na Seção 6 são feitas as considerações finais e propostos trabalhos futuros.

## 2. Tipos de Plágio

São diversas as formas de se alterar o código-fonte original de um programa para se cometer um plágio. Alunos fazem uso de alterações diferenciadas para disfarçar o plágio com o objetivo de não serem identificados.

Trabalhos existentes na literatura relacionam alguns tipos de disfarce de plágio [Faidhi 1987, Whale 1990, Joy 1999, Mozgovoy 2006, Poon et al. 2012], alguns deles fazem categorização entre eles, seja por tipo de modificação no código-fonte ou pelo nível de dificuldade de detecção do plágio. Observando os tipos citados nesses trabalhos, nota-se que muitos deles são citados por mais de um autor.

Com o intuito de solucionar os problemas mais simples para depois os mais complexos, são listados, na Tabela 1, os tipos de plágio citados pelos autores, ordenados pelo grau de dificuldade de identificação. O objetivo deste trabalho é propor soluções capazes de eliminar o efeito de alguns desses tipos de disfarce de plágio.

## 3. Trabalhos Relacionados

Várias ferramentas foram desenvolvidas no ramo de análise de similaridade e detecção de plágio em códigos-fontes de programas. Entre elas podem ser citadas: *jPlag* [Prechelt et al. 2002], *MOSS* [Univ. Stanford 2014], *Sherlock* [Pike 2012], *CodeMatch* [Zeidman 2006] e o BOCA-LAB [França 2011].

O *jPlag* [Prechelt et al. 2002] é uma ferramenta desenvolvida em Java para análise de similaridade entre códigos-fonte nas linguagens Java, C#, C, C++ e Scheme, assim

**Tabela 1. Tipos de plágio ordenados pelo grau de dificuldade de identificação**

| Identificador | Grau de dificuldade | Tipo de plágio |
|---|---|---|
| SEM_ALT | 0 | Cópia sem alteração |
| COMENT | 1 | Reescrita, adição ou omissão de comentários |
| NOM_ID | 1 | Modificação de nomes de identificadores |
| FORMAT | 1 | Alteração de espaçamento, identação e quebra de linha |
| PARENT | 1 | Inserção ou exclusão de parênteses |
| POS_VAR | 2 | Modificação de posição de variável |
| POS_OPE | 2 | Mudança na ordem de operadores/operandos em expressões |
| POS_BLO | 2 | Reordenação de blocos de código |
| POS_INS | 2 | Alterar a ordem de instruções sem afetar o funcionamento |
| POS_MET | 2 | Reordenação de métodos |
| ESCOPO | 2 | Modificação de escopo |
| TIPO | 3 | Alteração de tipos de dados |
| BUG | 4 | Introdução de *bug* |
| PARAMET | 5 | Modificação de parâmetros de métodos |
| FUNCAO | 6 | Substituir chamadas a funções pelo respectivo conteúdo |
| MULT_F | 6 | Substituição de múltiplas chamadas de procedimentos por chamadas de uma função única, ou vice-versa |
| EXP_EQU | 7 | Substituição de expressões por equivalentes |
| FOR_WHI | 7 | Substituição de estruturas de repetição |
| IF_CASE | 7 | Alteração das estruturas das instruções de seleção |
| INLINE | 7 | Refatoração de código (inline) |
| INUTEIS | 8 | Adição de instruções redundantes ou variáveis inúteis |
| ESTRUT | 9 | Redesenho estrutural do código |
| COP_ORIG | 10 | Combinação de código copiado com código original |

como texto em linguagem natural. É disponibilizado por meio de um *WebService* e retorna, no formato *HTML*, a lista dos arquivos com maior percentual de similaridade. Por ser um serviço remoto, as submissões entram em uma fila de espera de execução, e o tempo de resposta dependente da quantidade de requisições. No pré-processamento do *jPlag*, os arquivos de código fonte são normalizados com a remoção de espaços em branco, comentários e nomes de identificadores. Na fase de processamento é utilizado um algoritmo baseado no algoritmo utilizado no *YAP3*. Ele gera um *hash* para cada *substring* dos arquivos e realiza comparação entre os *hashs* retornando a lista de *substrings* comuns entre os arquivos. A similaridade é então calculada como fração da quantidade de *substrings* comuns e a quantidade total de *substrings*.

O sistema *MOSS* [Univ. Stanford 2014] foi desenvolvido em 1994 na *Stanford University* e trabalha com comparação das impressões digitais dos documentos. Assim como o *jPlag*, também é acessado por meio de um *WebService*. O *MOSS* aceita submissões de códigos nas linguagens C, C++, Java, Pascal, Ada, LISP, entre outras.

O *MOSS* faz uma divisão do documento em *substrings* de tamanho $k$, sendo $k$ escolhido pelo usuário e então aplica uma função *hash* para cada *substring* gerada, formando as impressões digitais do documento. A similaridade é calculada com base no número de *hashs* semelhantes nos arquivos. Comparar *hashs* exige menos custo computa-

cional do que comparar *strings* caractere a caractere. O tamanho das *substrings* escolhido pelo usuário impacta no tempo de processamento e na qualidade dos resultados.

O programa *Sherlock* [Pike 2012] é uma ferramenta de código aberto, em linguagem C, proposta para auxiliar na detecção de plágio de textos. Na fase de pré-processamento, é gerada uma assinatura digital calculando valores *hash* para sequência de palavras. O tamanho da sequência de palavras é definido pelo usuário. O grau de similaridade dos documentos é dado pelo percentual de semelhança das assinaturas geradas.

O *Sherlock* pode ser usado para encontrar plágio em códigos-fonte. Para isso, é preciso fazer uso de técnicas de normalização nos códigos antes de executá-lo. O sistema *Sherlock N-Overlap* [Maciel et al. 2012] é uma modificação do programa *Sherlock* para detectar plágio em respostas de problemas de programação. Antes de executar o *Sherlock*, são aplicadas quatro técnicas de normalização.

*CodeMatch* [Zeidman 2006] é um sistema que, diferente das outras ferramentas citadas, além de comparar códigos-fonte normalizados, também faz comparações de identificadores que não são palavras reservadas da linguagem e de linhas de comentários. Ele possui cinco algoritmos de comparação: *Statement Matching*, *Comment/String Matching*, *Instruction Sequence Matching*, *Identifier Matching* e *Correlation Score*.

O algoritmo *Statement Matching* compara cada linha do código-fonte dos arquivos, sem considerar comentários e linhas que possuem apenas palavras reservadas da linguagem. O *Comment/String Matching* compara as linhas de comentário e as palavras dos arquivos. O *Instruction Sequence Matching* compara a primeira instrução de cada linha dos arquivos. O *Identifier Matching* compara o número de identificadores que não são palavras reservadas. O *Correlation Score* calcula a semelhança entre os pares de arquivos.

Em [França 2011], foi desenvolvido o BOCA-LAB, uma ferramenta para avaliação de programas em linguagem C, C++ e Java e detecção de plágios. O BOCA-LAB é integrado ao Moodle em sua arquitetura para gerenciar as atividades educacionais e utiliza o *Sherlock* [Pike 2012] para detecção de plágio.

As ferramentas *jPlag* e *MOSS* possuem maior destaque entre as ferramentas para detecção de plágio, sendo, por esse motivo, usadas como comparação dos resultados obtidos nos experimentos realizados com *pPlagio*.

## 4. Metodologia

O *pPlagio* visa identificar e elimininar os efeitos de uma grande parte de tipos de disfarce de plágio através de transformações dos códigos na fase de pré-processamento antes da fase de comparação para detecção. A Figura 1 ilustra o processo de detecção do *pPlagio*.

O professor consulta um arquivo de código C no sistema. O *pPlagio* verifica quais arquivos da base são mais similares e exibe uma lista com todos os pares suspeitos de plágio para que o professor verifique se relamente houve plágio.

Para reduzir o esforço computacional na detecção de possíveis plágios, é utilizado um método de classificação, utilizado em [Campana-Filho et al. 2016] baseado no proposto em [Baby et al. 2014], para se ter uma base já classificada de códigos antes da verificação de plágio. Ao fazer a consulta, é feita a identificação da classe que o arquivo consultado pertence, e é feita comparação par a par apenas com os códigos da classe e são

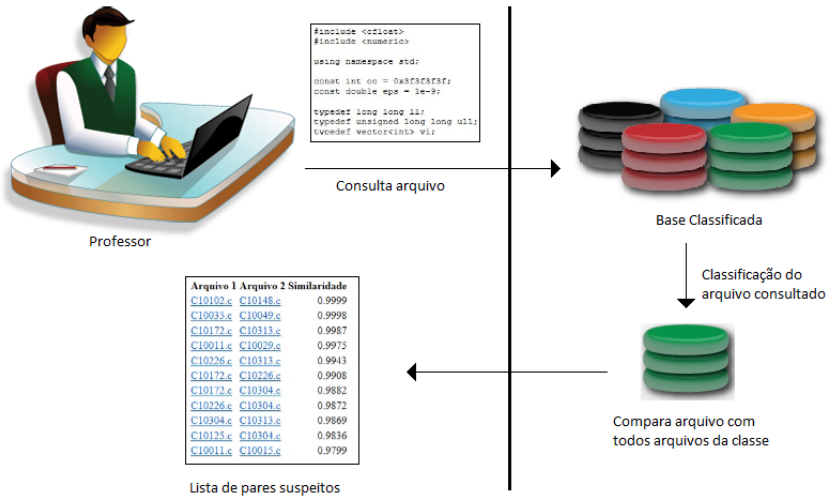


**Figura 1. Método para detecção de plágio**

listados os códigos mais similares com o código suspeito.

Se a base não estivesse classificada, a pesquisa seria feita comparando todos os códigos par a par, gerando um grande custo computacional para bases muito grandes.

### 4.1. Pré-processamento

Na fase de pré-processamento, o arquivo em linguagem C é normalizado, ou seja, passa por uma série de transformações, para então passar para a fase de detecção.

Uma lista de *tokens* é criada contendo as funções nativas e operadores usados na linguagem C. Na conversão, o código C é transformado em um vetor de ocorrência de termos. Um termo pode ser uma palavra-chave, um operador, um literal, uma constante, etc. Cada termo é convertido em um código inteiro que representa sua posição no vetor de frequência. O valor na posição referente ao termo é a sua frequência no código. A Tabela 2 mostra um exemplo de conversão de código C em vetor de frequência dos termos.

**Tabela 2. Exemplo de conversão de código em vetor de frequências de termos**

| Código | *Token* | Posição do *token* | Vetor de frequências |
|---|---|---|---|
| | @compila | 1 | |
| | @funciona | 2 | |
| | int | 3 | |
| `int main {` | main | 4 | |
| `int x = 10;` | { | 5 | |
| `int y = x;` | literal (x e y) | 6 | $v = \{1,1,3,1,1,3,2,2,3,1,1\}$ |
| `return 0;` | = | 7 | |
| } | constante (10 e 0) | 8 | |
| | ; | 9 | |
| | return | 10 | |
| | } | 11 | |

No exemplo da Tabela 2, os valores do vetor $v$ correspondem a frequência dos termos @compila, @funciona, ”int”, ”main”, ”{”, ”literal (x e y)”, ”=”, ”constante (10

e 0)", ";", "return" e "}" respectivamente. O termo @compila indica se o programa compila sem erros, gerando um executável, e @funciona, se sua execução funciona. A similaridade entre dois códigos é dada, então, com o cálculo da distância entre 2 vetores.

O pré-processamento dos arquivos tem influencia direta na qualidade da detecção. Por isso, foram desenvolvidas quatro versões de pré-processamento visando melhorar a detecção, adicionando funcionalidades para identificar os tipos de disfarce de plágio.

**Versão 1** – Nessa versão de pré-processamento, comentários são substituídos por @comentario, textos entre aspas por @textostr, e são consideradas como *tokens* válidos as palavras reservadas e operadores da linguagem C, como mostra a Tabela 3.

**Tabela 3. Lista de *tokens* convertidos da linguagem C**

| Lista de *tokens* para a linguagem C | | | | |
|---|---|---|---|---|
| @compila | @funciona | for | do | while |
| auto | break | continue | if | else |
| switch | case | default | struct | typedef |
| include | main | scanf | @imprimir | @comentario |
| gets | puts | getchar | return | @enderecamento |
| @igual | @menorigual | @maiorigual | @diferente | @atribuicao |
| @negacao | @maior | @menor | @e | @ou |
| @inc | @dec | @soma | @subtrai | @multiplica |
| @divide | @mod | const | char | double |
| float | int | long | short | unsignal |
| signed | void | enum | extern | goto |
| register | sizeof | static | union | volatile |
| @abreparenteses | @fechaparenteses | | | |

Com essa conversão consegue-se eliminar o efeito de alguns tipos de plágio como a modificação de texto de comentários, alteração de espaços em branco, identação e quebra de linha, visto que o texto do comentário é convertido em um único *token* e espaços em branco e outras formatações de texto são ignorados. Por outro lado, o fato de não considerar os textos de comentários tem um lado negativo, pois comentários semelhantes são indícios de plágio.

Os tipos de plágio de modificação de nome de identificadores e inclusão de variáveis inúteis também são eliminados, pois os identificadores que não são da linguagem são descartados. Desconsiderar identificadores tem um lado negativo, pois identificadores semelhantes são indícios de plágio.

Como o algoritmo que verifica a similaridade não considera a ordem dos termos, o efeito de outros tipos de plágio são eliminados, como a modificação de posição de variável, mudança na ordem de operadores/operandos em expressões, reordenação de métodos, instruções ou bloco de código. Porém, não considerar a ordem dos termos no código tem um lado negativo, pois a ordem representa a estrutura do código, e se forem semelhantes podem ser um indício de plágio. Além disso, nesta versão é verificado se o código fonte compila, e com isso é eliminado o tipo de plágio de introdução de *bug* no código.

**Versão 2** – Nessa versão são adicionados alguns *tokens* à lista de *tokens* válidos com o objetivo de melhorar a comparação dos códigos na fase de detecção. O nome das bibliotecas incluídas com o comando *include* também são considerados *tokens*.

As variáveis são convertidas no *token* @variavel e os nomes de funções para @funcao, pois a quantidade de variáveis e funções utilizadas no código é informação relevante para detectar plágio. O fato de considerar a quantidade de variáveis faz com que esta versão passe a não eliminar o efeito do tipo de plágio de inclusão de variáveis inúteis.

Os comentários são retirados ao invés de convertidos em @comentario. Com isso é eliminado o efeito da adição e exclusão de comentários.

**Versão 3** – Esta versão adiciona à *Versão 2* uma funcionalidade para tratar do tipo de plágio de inserção ou exclusão de parênteses desnecessários.

São consideradas duas situações. A primeira, na qual a quantidade de parênteses está incorreta, nesse caso ele é tratado como introdução de *bug* que o @compila sinaliza. A segunda, com a quantidade de parênteses correta, faz-se o uso de expresão REGEX para retirar parênteses desnecessários nas expressões.

Como exemplo da transformação feita pelo REGEX, a linha de código `a = (((b + c)/(d+5)));` é subtituída por `a = (b + c)/(d+5);`.

**Versão 4** – Esta versão adiciona à versão 3 uma funcionalidade para tratar do tipo de plágio de variáveis e constantes inúteis. É utilizado o próprio compilador da linguagem C, o `gcc`, para isso. Ao executá-lo com a opção `-Wall`, ele retorna uma lista com as variáveis que não estão sendo utilizadas no código. Com essa lista, é executado um script do *shell* que retira essas variáveis do código.

Por exemplo, se o código tiver um comando de declaração de constante `const` com uma constante não utilizada no programa, esta linha é retirada do código.

## 5. Experimentos

Para o experimento utilizou-se a base que fez parte do evento *Detection of SOurce COde Re-use* (SOCO) [Flores et al. 2014]. Esta base possui 19.895 códigos em C/C++, divididos em 6 cenários, o evento disponibilizou a lista de pares de códigos plagiados para as bases. Os plágios ocorrem somente em códigos do mesmo cenário. A Tabela 4 mostra a quantidade de arquivos de código e a quantidade de pares de plágio para cada cenário.

**Tabela 4. Base de dados do evento SOCO**

| Cenário | Quantidade de arquivos | Pares de plágio |
|---|---|---|
| A1 | 5408 | 99 |
| A2 | 5195 | 86 |
| B1 | 4939 | 86 |
| B2 | 3873 | 43 |
| C1 | 335 | 8 |
| C2 | 145 | 0 |

Foram feitos dois experimentos. O primeiro com a base utilizada no evento SOCO e comparação dos resultados do *pPlagio* com os obtidos no evento, inclusive com a ferramenta *jPlag*. O segundo, com arquivos criados *ad hoc*, para verificar se o *pPlagio* conseguiria identificar suspeita em arquivos alterados por determinados tipos de plágio, os resultados deste experimento foram comparados com os da ferramenta *MOSS*.

### 5.1. Detecção de plágio

Nesse experimento, foram feitos cálculos de similaridade para identificar suspeita de plágio para as quatro versões de pré-processamento do *pPlagio*. No *site* do evento SOCO

é possível visualizar os resultados que os participantes produziram com seus algoritmos. O resultado é dado nas seguintes medidas F1, Precisão e Revocação, priorizando nessa ordem. Precisão indica o percentual de amostras ditas da classe que realmente são. Revocação indica o percentual de amostra de uma classe que foram classificadas corretamente. F1 é uma ponderação entre Precisão e Revocação. Na Tabela 5 são comparados os resultados obtidos com as quatro versões do *pPlagio* e os dos participantes do evento.

**Tabela 5. Comparação de resultados obtidos com resultados do evento SOCO**

| Ranking | Método de detecção | F1 | Precisão | Revocação |
|---|---|---|---|---|
| 1 | *pPlagio* V4 | 0.452 | 0.488 | 0.449 |
| 2 | *pPlagio* V3 | 0.448 | 0.486 | 0.443 |
| 3 | *pPlagio* V2 | 0.448 | 0.486 | 0.443 |
| 4 | UAEM-run1 | 0.440 | 0.282 | 1.000 |
| 5 | UAEM-run2 | 0.387 | 0.240 | 1.000 |
| 6 | *baseline* 2 | 0.295 | 0.258 | 0.345 |
| 7 | *baseline* 1 | 0.190 | 0.350 | 0.130 |
| 8 | *pPlagio* V1 | 0.098 | 0.062 | 0.661 |
| 9 | UAM-C-run1 | 0.013 | 0.006 | 1.000 |
| 10 | UAM-C-run3 | 0.013 | 0.006 | 0.997 |
| 11 | UAM-C-run2 | 0.010 | 0.005 | 0.950 |

Como pode-se observar, a Versão 1 do *pPlagio* obteve resultados para a medida F1 abaixo das 2 *baselines* e acima das 3 tentativas de UAM-C. As Versões 2, 3 e 4 tiveram resultados melhores que o primeiro colocado do evento, isso se deve ao aumento da medida Precisão que elevou a medida F1 em relação a Versão 1. Melhores resultados foram obtidos para F1 quando utilizou-se o valor de 0.97 como mínimo de similaridade entre os pares. O valor da Revocação foi melhor que a das *baselines* 1 e 2. A *baseline* 1 é o resultado de execução do programa *jPlag* e o *baseline* 2 de uma técnica de *n-gram*.

### 5.2. Identificação dos tipos de plágio

Para verificar se o *pPlagio* consegue eliminar o efeito dos tipos de plágio as quais se propõe, foram utilizados os códigos 042, 045, C10097 e C10278 da base do evento SOCO e aplicados os tipos de plágio para gerar versões plagiadas. Essas versões foram submetidas ao *MOSS* e ao *pPlagio* e os resultados foram comparados, conforme a Tabela 6.

A primeira coluna mostra o par de arquivos a ser identificado como suspeito de plágio, a segunda mostra o tipos de plágio que foi aplicado para gerar o arquivo de cópia. As outras colunas indicam se a as versões do *pPlagio* e o *MOSS* conseguiram detectar o possível plágio. A detecção foi considerada para similaridade entre o par acima de 95%.

Conforme Tabela 6, nota-se que o *pPlagio* Versão 1 detectou 10 de 22 pares de arquivos simulados com tipos de plágio e detectou 1 dos 2 pares de arquivos reais contendo tipos de plágio combinados. A Versão 4 aumentou para 12 detecções das 22 e manteve a detecção de 1 dos 2 pares reais. O *MOSS* conseguiu detectar 11 pares dos 22, e não detectou os pares de arquivos reais com tipos combinados de plágio.

## 6. Considerações Finais

Este trabalho apresentou o *pPlagio*, sistema que tem como propósito auxiliar professores na detecção de ocorrências de plágio entre alunos de disciplina de programação de com-

**Tabela 6. Resultados obtidos com *pPlagio* e *MOSS* para arquivos simulados**

| Arquivos plagiados | Tipo de plágio aplicado | *pPlagio* v1 | v2 | v3 | v4 | *MOSS* |
|---|---|---|---|---|---|---|
| 42 e A42 | SEM_ALT | SIM | SIM | SIM | SIM | SIM |
| 42 e B42 | COMENT | - | SIM | SIM | SIM | SIM |
| 42 e C42 | NOM_ID | SIM | SIM | SIM | SIM | SIM |
| 42 e D42 | FORMAT | SIM | SIM | SIM | SIM | SIM |
| 42 e E42 | PARENT | - | - | SIM | SIM | SIM |
| 42 e F42 | POS_VAR | SIM | SIM | SIM | SIM | - |
| C10097 e C1H097 | POS_OPE | SIM | SIM | SIM | SIM | - |
| C10097 e C1I097 | POS_BLO | SIM | SIM | SIM | SIM | SIM |
| C10097 e C1J097 | POS_INS | SIM | SIM | SIM | SIM | - |
| C10097 e C1K097 | POS_MET | SIM | SIM | SIM | SIM | - |
| C10097 e C1L097 | ESCOPO | - | - | - | - | SIM |
| C10097 e C1M097 | TIPO | - | - | - | - | - |
| C10097 e C1N097 | BUG | SIM | SIM | SIM | SIM | SIM |
| C10097 e C1O097 | PARAMET | - | - | - | - | SIM |
| C10097 e C1P097 | FUNCAO | - | - | - | - | SIM |
| C10097 e C1Q097 | MULT_F | - | - | - | - | - |
| C10097 e C1R097 | EXP_EQU | - | - | - | - | - |
| C10097 e C1S097 | FOR_WHI | - | - | - | - | - |
| C10097 e C1T097 | IF_CASE | - | - | - | - | - |
| C10097 e C1U097 | INLINE | - | - | - | - | SIM |
| C10097 e C1V097 | INUTEIS | SIM | - | - | SIM | - |
| 45 e X45 | COP_ORIG | - | - | - | - | - |
| 042 e 045 | Tipos combinados | - | - | - | - | - |
| C10097 e C10278 | Tipos combinados | SIM | - | - | SIM | - |

putadores. O *pPlagio* faz o pré-processamento dos códigos em linguagem C submetidos pelos alunos, faz comparação entre eles para obter suas respectivas similaridades e exibe para o professor aqueles códigos mais similares, possíveis de serem casos de plágio.

Visando uma maior abrangência na descoberta de plágio, foram implementadas várias técnicas de pré-processamento para tratar alguns tipos de plágio geralmente aplicados por alunos. Com a utilização dessas técnicas, a medida F1 na detecção com a melhor versão do *pPlagio* foi de 0.452 na base do evento SOCO, superando o primeiro colocado com F1 de 0.440 e a ferramenta *jPlag* 0.295, utilizada como *baseline*.

Comparando o *pPlagio* com o MOSS, foi utilizada uma base construída *ad hoc* onde simulamos 22 tipos de plágio e mais 2 pares de plágio em arquivos reais contendo vários tipos de plágio combinados. A melhor versão do *pPlagio* identificou 12 dos 22 pares avaliados e 1 dos 2 pares reais, enquanto que o MOSS identificou 11 dos 22 pares avaliados e nenhum dos 2 pares reais.

Como trabalho futuro, é interessante o desenvolvimento de outras técnicas para a detecção de outras formas de plágio não tratadas neste trabalho. A implementação dessas técnicas contribuiriam para melhores resultados. Uma outra melhoria ao trabalho seria a adição ao *pPlagio* da funcionalidade de identificar plágio em códigos de outras linguagens de programação além da linguagem C, como por exemplo Java. Essa melhoria tornaria o *pPlagio* ainda mais popular. Essa funcionalidade pode ser extendida para viabilizar

comparações entre códigos em linguagens diferentes. Seria possível comparar um código escrito em C com um escrito em Java, por exemplo.

## Referências


Baby, J., T, K., P, V., and Gopal, V. (2014). Distance indices for the detection of similarity in c programs. In *Computation of Power, Energy, Information and Communication (ICCPEIC)*, Chennai. IEEE.

Campana-Filho, J. C., Oliveira, M. G., and Oliveira, E. (2016). Classificação de Códigos C Usando Medidas de Similaridade para Apoio ao Ensino em Programação. In *XXVII Simpósio Brasileiro de Informática na Educação (SBIE)*, Uberlândia, MG. SBC.

Flores E. and Rosso P. and Moreno L. and Villatoro-Tello E. (2014). On the detection of source code re-use. *FIRE'14 Proceedings of the Forum for Information Retrieval Evaluation*, pages 21–30.

Faidhi, J. A. W.; Robinson, S. K. (1987). An empirical approach for detecting program similarity and plagiarism within a university programming environment. *Computers & Education, Elsevier Science Ltd*, 11:11–19.

França, A. B.; Soares, J. M. (2011). Sistema de apoio a atividades de laboratório de programação via moodle com suporte ao balanceamento de carga. *XXII Simpósio Brasileiro de Informática na Educação*, pages 710–719.

Joy, M.; Luck, M. (1999). Plagiarism in programming assignments. *IEEE Transactions on Education, Institute of Electrical and Electronics Engineers*, 42:129–133.

Maciel, D. L., Soares, J. M., Bonetti, A., and Gomes, D. (2012). Análise de similaridade de códigos-fonte como estratégia para o acompanhamento de atividades de laboratório de programação. *Revista Novas Tecnologias na Educação (RENOTE)*, 10:1–10.

Mozgovoy, M. (2006). Desktop tools for offline plagiarism detection in computer programs. *Informatics in Education*, 5:97–112.

Pike, R. (2012). The sherlock plagiarism detector. In *Disponível em* `http://sydney.edu.au/engineering/it/~scilect/sherlock/`.

Poon, J. Y., Sugiyama, K., Tan, Y. F., and Kan, M.-Y. (2012). Instructor-centric source code plagiarism detection and plagiarism corpus. *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education*, pages 122–127.

Prechelt, L., Malpohl, G., and Phlippsen, M. (2002). Finding plagiarisms among a set of programs with jplag. In *J. UCS Journal of Universal Computer Science*.

Univ. Stanford (2014). Moss (measure of software similarity) plagiarism detection system. In *Disponível em* `http://theory.stanford.edu/~aiken/moss/`, Univ. Stanford, Califórnia.

Whale, G. (1990). Identification of program similarity in large populations. *The Computer Journal*, pages 140–146.

Zeidman, R. (2006). Software source code correlation. *Computer and Information Science, 2006 and 2006 1st IEEE/ACIS International Workshop on Component-Based Software Engineering, Software Architecture and Reuse*, pages 383–392.