ANÁLISE DE SIMILARIDADE DE CÓDIGOS-FONTE COMO ESTRATÉGIA PARA O ACOMPANHAMENTO DE ATIVIDADES DE LABORATÓRIO DE PROGRAMAÇÃO

Danilo Leal Maciel[1] - daniloleal@alu.ufc.br
José Marques Soares[1] - marques@ufc.br
Allyson Bonetti França[1] - allysonbonetti@gmail.com
Danielo Gonçalves Gomes[1] - danielo@ufc.br

[1]Programa de Pós-Graduação em Engenharia de Teleinformática (PPGETI/UFC)

Resumo. *Este trabalho apresenta a implementação e a avaliação de uma ferramenta de apoio ao acompanhamento de atividades laboratoriais de programação de computadores. Um suporte automatizado instrumentaliza o professor para a comparação e a análise de similaridade entre as soluções apresentadas pelos alunos, para os problemas de programação propostos. A fim de desconsiderar diferenças irrelevantes entre códigos-fonte, foram desenvolvidas técnicas para normalizar os códigos antes do processo de comparação realizado pelo algoritmo Sherlock, frequentemente empregado para detecção de plágio em documentos de caráter geral. As técnicas desenvolvidas apresentam resultados equivalentes aos das melhores ferramentas usadas para a detecção de plágio. A proposição do problema pelo professor, o envio das soluções dos alunos e a análise de similaridade são feitos através de um módulo implantado no Moodle, facilitando a gestão desse tipo de atividade em turmas numerosas. O sistema foi experimentado em uma turma de programação ao longo de um semestre e os resultados são apresentados nesse artigo.*

Palavras-chave: *análise de similaridade, laboratório de programação, avaliação.*

SIMILARITY ANALYSIS OF SOURCE CODE AS A STRATEGY FOR MONITORING OF PROGRAMMING LABORATORY ACTIVITIES

Abstract. *This paper presents the implementation and evaluation of a tool to support the monitoring of computer programming laboratory activities. This tool assists the professor in the analysis of similarity between the students' solutions of the proposed activities. In order to ignore irrelevant differences between source-codes, techniques have been developed to normalize the source-codes before the comparison performed by the algorithm Sherlock, often used for detecting plagiarism in documents of a general nature. The techniques developed have equivalent results to those of the best tools used to detect plagiarism. The proposition of the problem by the teacher, the upload of the solution by the student, and the similarity analysis is done through a module deployed in Moodle, facilitating the management of this type of activity in large classes. The system was tested on a programming class over a semester and the results are presented in this article.*

Keywords: *similarity analysis, laboratory programming, assessment.*

1. Introdução

Para dar suporte ao trabalho com turmas numerosas e minimizar as dificuldades associadas aos requisitos de acompanhamento de alunos, professores buscam usar a tecnologia para facilitar o gerenciamento de turmas grandes. Alguns ambientes virtuais, como, por exemplo, o Moodle (Dougiamas e Taylor, 2003), podem ser usados para disponibilização de notas de aula, proposta e submissão de trabalhos e registro de notas. Outras ferramentas são mais especificamente adequadas para auxiliar nas tarefas de compilação e execução dos programas desenvolvidos pelos alunos, como o VPL (2012) e o Onlinejudge (2012). No entanto, embora o uso de tais ferramentas possa mitigar os problemas de natureza organizacional e de conferência de resultados em práticas laboratoriais, não são suficientes para contornar todas as dificuldades inerentes ao acompanhamento individualizado.

Um problema não raramente encontrado em laboratórios de programação é a cópia total ou parcial de soluções entre colegas, frequentemente com a mudança de nomes de variáveis ou com a inserção de comentários, para dificultar a percepção da ação. Em cenários de turmas numerosas, a detecção deste tipo de conduta se torna mais problemática.

Embora a detecção de plágio seja um aspecto importante na condução de uma turma de programação, os estudos desenvolvidos neste trabalho revelam que os resultados da comparação entre códigos podem indicar outros aspectos que são de natureza irrepreensível. Dependendo da perspectiva e do contexto, a semelhança pode ser indicativa de parceria, de trabalho colaborativo, de referência ou apoio sobre solução encontrada em livros ou exemplos fornecidos pelo professor, entre outros motivos. Por essa razão, ao invés de “detecção de plágio”, adota-se neste trabalho a expressão “análise de similaridade”.

A detecção de plágio (e a análise de similaridade) em código-fonte vem sendo estudada em diversos trabalhos (Hage *et al.*, 2010) (Bin-Habtoor, 2012) (Djuric e Gasevic, 2012) e algumas ferramentas foram disponibilizadas para este fim (Prechelt, 2002) (MOSS, 2012). Neste trabalho, avalia-se especialmente a sensibilidade do algoritmo Sherlock (Pike, 2012) quando os códigos são submetidos a técnicas de pré-processamento que visam a normalização antes da comparação. O objetivo é reduzir a diferenciação dos códigos por aspectos irrelevantes, como comentários, valores literais ou nomes de variáveis, entre outros aspectos. Os resultados são comparados com aqueles obtidos com o uso do JPlag (Prechelt, 2002) e MOSS (2012). Os testes foram realizados em duas etapas, na primeira, a partir dois códigos base, foram geradas variações baseadas nas principais modificações que comumente são realizadas ao se plagiar um código, no segundo utilizou-se de 32 trabalhos de programação propostos e desenvolvidos para uma turma iniciante com 97 alunos.

A função de análise de similaridade corresponde a um dos módulos do BOCA-LAB (França e Soares, 2011), ferramenta que foi integrada ao Moodle e permite a submissão *online* das atividades de laboratório desenvolvidas nas disciplinas de programação. O BOCA-LAB compila e executa remotamente as soluções enviadas pelos alunos, efetuando, ainda, a verificação das saídas geradas pelos programas. As técnicas de pré-processamento para o uso do Sherlock e a análise comparativa dos resultados alcançados representam a principal contribuição deste trabalho, o que permitiu a implementação de melhorias no módulo de análise de similaridade do BOCA-LAB.

Este artigo está organizado da seguinte maneira: após a introdução, discutem-se,

na seção 2, as principais ferramentas para análise de similaridade; na seção 3, enumeram-se os detalhes das técnicas de normalização desenvolvidas neste trabalho. Posteriormente, na seção 4, exploram-se os resultados obtidos com essas técnicas analisando códigos em duas abordagens: manual e com códigos produzidos por alunos de graduação, seguindo, na seção 5, apresenta-se como a ferramenta alvo deste trabalho está inserida contexto do ambiente Moodle e a importância para o professor. Por fim, são apresentadas as conclusões e as perspectivas para o trabalho.

2. Ferramentas para Análise de Similaridade

Diversas ferramentas podem ser utilizadas para realizar a análise de similaridade entre códigos-fonte, tais como SIM (Grune e Vakgroep, 1989), YAP (Lancaster e Culwin, 2001), JPlag (Prechelt, 2002), SID (Chen *et al*., 2004), Plaggie (Ahtiainen, 2006) e MOSS (2012). Estudos comparativos sobre algumas destas ferramentas foram realizados por Hage *et al.* (2010) e Kleiman (2007). De modo geral, ferramentas especificamente construídas para esta finalidade permitem encontrar semelhanças no caso de alterações de nome de variáveis, nome de funções, comentários ou, ainda, alterando-se a ordem de partes do código. Segundo Burrows *et al.* (2007), JPlag e MOSS são algumas das mais importantes e mais citadas para detecção de plágio, que são sucintamente descritos nos próximos parágrafos, seguidos pelo Sherlock, sobre o qual se apóiam as contribuições deste trabalho.

O JPlag (Prechelt, 2002) é uma ferramenta desenvolvida em Java para análise de similaridade entre códigos-fonte. É disponibilizada exclusivamente por meio de um *WebService* e possui código fechado. Ela requer o cadastramento e a requisição de autorização para o acesso ao serviço. O envio de arquivos para comparação é realizado através de um *applet* que retorna, em formato HTML, o resultado do processamento. A semelhança é apresentada em percentual apenas para aqueles com maior similaridade. Não é disponibilizada, portanto, a visualização de pares com porcentagens pequenas.

O MOSS (*Measure of Software Similarity*) (MOSS, 2012) também é acessado por meio de um *WebService*, disponibilizado na Universidade da Califórnia. Para usá-lo, é necessário realizar um cadastramento por *email*. Os arquivos são enviados para o servidor por um *script* fornecido pelo seu desenvolvedor. Ao final do envio, é gerada uma URL que fornece o resultado da comparação. Segundo informações obtidas na página Web da aplicação, o desenvolvimento do MOSS iniciou-se em 1994, apresentando uma melhora significativa em relação a outros algoritmos (que não são mencionados) para detecção de plágio. O MOSS é um sistema de código fechado baseado no algoritmo Winnowing (Schleimer, 2003).

O Sherlock (Pike, 2012) é uma alternativa às ferramentas apresentadas precedentemente por ser de código aberto, permitindo modificações e melhorias. Além disso, foi possível implantá-la no BOCA-LAB, sem exigir conectividade com *WebServices* de terceiros, o que por si só já representa perda performática expressiva, ao se comparar com o tempo em que o Sherlock retorna os resultados. Desenvolvido em linguagem C, apresenta bom desempenho e realiza a análise de semelhança léxica entre documentos textuais, incluindo códigos-fonte, enquanto que as outras ferramentas foram concebidas para comparar documentos em linguagens de programação específicas. Para encontrar trechos duplicados de cada documento, é gerada uma assinatura digital calculando valores *hash* para palavras e sequência de palavras. Ao final, comparam-se as assinaturas geradas e identifica-se o percentual de semelhança.

O Sherlock funciona por meio da comparação de palavras separadas por

espaços. Decorrente dessa característica, por exemplo, a expressão "`for(`" é avaliada diferentemente da expressão "`for (`", com espaço, apesar de essa diferença não ser significativa para a compilação. Logo, foi necessário desenvolver técnicas para manipular os códigos com a ideia inicial de uniformizá-los, buscando-se garantir que expressões equivalente sejam consideradas de maneira coerente.

3. Critérios e Técnicas para normalização de código

A análise de similaridade acontece em duas etapas: o pré-processamento (normalização) dos códigos e a comparação pelo Sherlock (Pike, 2012). A normalização atua principalmente na identificação e remoção de trechos de código sem relevância, além de padronizar os códigos para facilitar a comparação. Além disso, visa desconsiderar modificações inseridas para encobrir plágios. Essas modificações, em geral, não são complexas, sendo limitadas a mudanças que não alteram o funcionamento do programa.

Com a finalidade de investigar as melhores técnicas para análise de similaridade, adotaram-se quatro abordagens de normalização, com graus de intervenção crescentes. Assim, o código gerado com o uso da primeira técnica é o que mais se assemelha ao da versão original, enquanto aquele gerado pela última técnica realiza alterações mais invasivas. As quatro técnicas são descritas na Tabela 1.

Tabela 1 - Descrição das técnicas de normalização

Código Original
<Sem modificações>
	`// Imprime resposta`
	`for (i=1;i<= n;i++){`
	
	`if (i %3== 1){`
	`printf("Resp %d", v[i]/x+z);`
	`}`
	`}`

Normalização 1
Remoção de linhas e espaços vazios;
Remoção de todos os comentários;
Remoção das referências aos arquivos externos (bibliotecas);
Inclusão (ou remoção) de espaços em branco entre expressões, declaração de variáveis e outras estruturas.
Regras específicas para aproximar e afastar caracteres para normalização 1.
	`for( i=1; i<=n; i++ ) {`
	`if( i % 3 ==1 ) {`
	`printf ( " Resp %d ", v[ i ] / x +z );`
	`}`
	`}`

Normalização 2
Aplicação da normalização 1;
Remoção de todos os caracteres situados entre aspas.
Regras específicas para aproximar e afastar caracteres para normalização 2.
	`for( i=1; i<=n; i++ ) {`
	`if( i % 3 ==1 ) {`
	`printf( , v[ i ] / x +z );`
	`}`
	`}`

Normalização 3
Aplicação da normalização 2;
Remoção de todos os valores literais e variáveis.
Regras específicas para aproximar e afastar caracteres para normalização 3.
	`for( = ; <= ; ++ ) {`
	`if( % == ) {`
	`printf( , [] / +);`
	`}`
	`}`

Normalização 4
Aplicação da normalização 3;
Remoção de todas as palavras reservadas.
Regras específicas para aproximar e afastar caracteres para normalização 4.
	`( =; <=; ++ ) {`
	`( %== ) {`
	`( , [] / +);`
	`}`
	`}`

As técnicas propostas são formadas por algumas dezenas de regras especificadas a partir de análise lógica e empírica. Observando a técnica de normalização 1 aplicada ao código exemplo da Tabela 1, verificou-se, por exemplo, que palavras reservadas indicativas de funções parametrizadas e estruturas condicionais, como: "`printf (`", e "`if (`", são melhor comparadas quando se eliminam os espaços antes do parêntese, resultando em "`printf(`" e "`if(`". As consequências ocorridas na comparação feita pelo Sherlock em relação a separação pelo espaço em branco de elementos de estruturas logicamente associadas requer que, na normalização 4, os caracteres especiais sejam unidos, como em "`(,);`", ao invés de "`( , ) ;` ", para se obter o melhor resultado.

Em alguns experimentos, observou-se empiricamente que, para expressões como "`i%3==1`", obtêm-se melhores resultados quando o valor numérico está junto ao operador do lado direito, e os elementos do lado esquerdo estão separados, com a seguinte configuração: "`i % 3 ==1`". Contudo, para declaração de variáveis, por exemplo, "`int i=1;`", a organização que resulta em melhores resultados é "`int i = 1;`".

Cada normalização representa em si uma estratégica única, que herda da anterior algumas propriedades, que podem ser diferenciadas nas regras que levam a aproximar ou afastar determinados caracteres. Por exemplo, para a normalização 3 os resultados são melhores quando "`% ==`" ficam separados, contudo na normalização 4 convém deixar "`%==`" juntos. O conjunto dessas pequenas regras produzem resultados eficazes, conforme será demostrado na próxima seção.

4. Testes e Resultados

Utilizamos duas abordagens para validar a proposta apresentada. A primeira baseia-se em códigos gerados de modo manual, e, na segunda, códigos desenvolvidos por alunos.

4.1. Análise com códigos-fonte gerados manualmente

Para verificar a análise de similaridade através de um procedimento de simulação, foram realizadas modificações manualmente em 2 códigos de teste. Eles foram escolhidos por reunirem a maioria dos conceitos presentes em um primeiro curso de graduação para a linguagem C. A partir de cada código base, foi criado um conjunto de 8 variações, seguindo os padrões de plágio mais frequentemente encontrados, adaptando-se aqueles que são elencados por Ahmadzadeh *et al*. (2011). As alterações inseridas no código foram realizadas da seguinte maneira: (I) Código base; (II) Cópia do código base; (III) Inclusão/edição de comentários; (IV) Mudança de nomes de variáveis; (V) Troca de posição de variáveis e funções; (VI) Mudança de escopo de variável; (VII) Alteração na indentação; (VIII) Inclusão de informação inútil: incluir bibliotecas, variáveis, comentários; (IX) Rearranjo de expressões; (X) Todas as alterações combinadas. As alterações realizadas não ocorrem de forma incremental, ao contrário, são originadas do mesmo código base, com exceção da última, que reúne alterações de todos os itens anteriores.

As Tabelas 2a e 2b listam todos os resultados obtido pelas ferramentas JPlag (Prechelt, 2002), MOSS (2012) e o Sherlock (Pike, 2012) após a aplicação das técnicas de normalização 1 (T1), 2 (T2), 3 (T3) e 4 (T4) propostas. A primeira representa alterações em um código base com 30 linhas e a segunda em outro com 70 linhas. Supondo que as ferramentas sejam ideais, o resultado esperado em cada célula da tabela é 100% de similaridade, já que os códigos preservam em sua totalidade, a lógica contida no programa de base, contudo verifica-se que as ferramentas apresentam limitações.

Conforme concluído por Kleiman (2007), também verificamos que a normalização é mais importante do que a própria ferramenta de comparação.

Tabela 2 - Comparação de similaridade a partir de 2 códigos base

(a)

	JPlag	MOSS	T1	T2	T3	T4
I	-	-	-	-	-	-
II	100%	92%	100%	100%	100%	100%
III	100%	92%	100%	100%	100%	100%
IV	100%	92%	100%	50%	42%	100%
V	50%	0%	50%	50%	50%	33%
VI	70.8%	69%	100%	100%	100%	50%
VII	100%	92%	100%	100%	100%	66%
VIII	31.1%	17%	50%	50%	11%	40%
IX	100%	92%	50%	50%	100%	66%
X	34.7%	0%	33%	0%	7%	16%

(b)

	JPlag	MOSS	T1	T2	T3	T4
I	-	-	-	-	-	-
II	100%	97%	100%	100%	100%	100%
III	100%	97%	100%	100%	100%	100%
IV	100%	97%	19%	11%	36%	100%
V	62%	55%	75%	70%	83%	75%
VI	70.9%	65%	100%	100%	100%	100%
VII	100%	97%	100%	100%	81%	66%
VIII	46%	50%	75%	70%	50%	50%
IX	100%	82%	75%	70%	76%	75%
X	50.3%	53%	19%	11%	30%	100%
X(2)	33.3%	35%	20%	4%	18%	40%

Pela análise da Tabela 2a, das técnicas propostas, a T1 e T3 destacam-se com os melhores resultados. Das 9 comparações T1 apresenta resultados melhores que o MOSS em 8 casos (II, III, IV, V, VI, VII, VIII e X), e com relação ao JPlag, 7 resultados são equivalentes ou melhor (II, III, IV, V, VI, VII, VIII e X), perdendo apenas em IX. Enquanto T3 apresenta resultados mais favoráveis que o MOSS em 7 casos (II, III, V, VI, VII, IX e X) e com resultados piores em 2 (IV e VIII).

Para a Tabela 2b, na linha X, obteve-se um resultado inesperado, em que a técnica de normalização 4, aplicada a uma variação do código base que acumulava todas as alterações, apresentou um resultado muito diferente dos demais. Para realizar uma análise mais aprofundada, foi gerada nova versão para o código X, conforme a linha X(2), na qual alterações mais profundas foram realizadas e o teste repetido. Ainda dessa forma, a técnica de normalização 4 apresentou resultados ainda melhores do que do JPlag e MOSS. Comparando os desempenhos, a T4 se destaca, apresentando resultados inferiores ao MOSS e JPlag apenas em duas situações (V e IX), enquanto as demais, apresenta resultados inferiores em média em 4 dos 9 casos.

Dos dados apresentados, conclui-se que as técnicas propostas possuem comportamentos complementares, o que leva a sugerir que uma solução combinada pode apresentar resultados melhores que as duas principais ferramentas e com as vantagens citadas na seção 2.

4.2. Análise com códigos-fonte submetidos por alunos

A análise de similaridade também foi avaliada em uma turma de introdução à programação, em trabalhos desenvolvidos com a linguagem C, durante o 1º semestre de 2012. A turma é formada por 97 alunos, divididos em 4 grupos. Cada grupo realizou 4 atividades contendo 2 problemas diferentes, perfazendo, ao todo, 32 problemas.

Os resultados da análise de similaridade usando as quatro técnicas de normalização foram comparados entre si e também aos resultados das ferramentas JPlag e MOSS. A tabela 3 relaciona a quantidade de códigos que apresentam índice de similaridade maior do que 20% e 70% para cada tipo de normalização. A análise da tabela mostra que, em geral, a sensibilidade à identificação de similaridade apresenta um comportamento crescente da técnica 1 para a técnica 4. Entretanto, pode-se observar que, para alguns problemas, como o de número 8, essa característica não se confirma em relação às técnicas 2, 3 e 4, sendo, na verdade, invertidas. Isso mostra que as

técnicas podem revelar semelhanças que vão além da organização estrutural do código, fornecendo elementos de análise diferenciados para o professor.

Tabela 3 - Grupo A: comparação das 4 técnicas com JPlag e MOSS

	Norm 1		Norm 2		Norm 3		Norm 4		JPlag		MOSS	
	> 20%	> 70%	>20%	> 70%	> 20%	> 70%	> 20%	> 70%	> 20%	>70%	> 20%	>70%
Problema 1	13	2	33	1	52	0	10	1	7	4	15	3
Problema 2	5	0	22	0	19	1	4	0	2	2	2	1
Problema 3	5	0	23	1	32	4	12	10	45	38	14	4
Problema 4	4	0	12	0	30	0	30	3	47	9	15	3
Problema 5	15	0	45	4	38	3	72	10	106	34	65	6
Problema 6	13	0	59	5	57	3	101	22	115	37	62	5
Problema 7	12	3	42	4	65	4	15	6	15	6	12	3
Problema 8	16	0	120	55	71	1	10	5	9	5	6	2

Sobre as diferenças observadas entre os resultados da análise de similaridade usando-se normalizações distintas, pode-se interpretar que: (i) a normalização é necessária para que se possa obter resultados significativos com o uso do Sherlock em comparação de códigos-fonte; (ii) normalizações diferentes podem revelar características de similaridade diferentes.

As justificativas para as diferentes ocorrências entre as aplicações da normalização poderiam também levar a duas hipóteses: (i) existe pouca ocorrência de plágio e, na verdade, a normalização 4 é constituída por falsos positivos; (ii) a normalização 1 não é eficiente e a 4 é a que apresenta melhores resultados. Entretanto, para aprofundar um pouco mais essa análise, foram gerados os resultados para as mesmas questões com o uso do JPlag e do MOSS, ilustrada na tabela 3 e 4.

A tabela 4 apresenta os comparativos do número de ocorrência de similaridades para o grupo D da turma de programação, levando-se em consideração apenas as técnicas de normalização 1 e 4, além do JPlag e do MOSS. Pela baixa incidência de similaridade apontada pela técnica de normalização 1 em relação às demais, infere-se que a técnica 1 produz os resultados menos significativos. Além disso, o JPlag, praticamente em todos os problemas, indica o maior número de casos de similaridade, e, por outro lado, o MOSS sempre aponta o menor número de similaridades registradas. Isso demonstra que a sensibilidade utilizada pelo MOSS é a mais conservadora para indicação de similaridade. A técnica de normalização 4, com o uso do Sherlock, apresenta resultados intermediários entre os do MOSS e do JPlag em 62% dos problemas para os registros de similaridades acima de 70%, demonstrando, assim, constituir-se em uma promissora ferramenta de análise. Um aspecto importante da técnica 4 associada ao Sherlock é a redução expressiva do tamanho do código a ser comparado devido à eliminação de grande parte do código, o que, na maioria das vezes, permitirá o processamento mais rápido da comparação.

O uso do Sherlock com a técnica de normalização 4, em algumas situações, pode apresentar maior ocorrência de similaridade do que o JPlag, como pode ser observado pelo problema 31, que propõe a impressão por extenso do valor de um algarismo inteiro. A análise visual dos códigos dos alunos mostrou que quase todos desenvolveram uma solução com o uso do *swicth*, com código bastante semelhante, o que foi feito por orientação do professor (e poderia também ter sido conseqüência do uso de algum exemplo encontrado em fontes de referência). Na perspectiva da similaridade, os resultados não são, portanto, considerados falsos positivos para essa

questão. Assim, verifica-se que os resultados do Sherlock com a normalização 4 foram mais significativos do que os apresentados pelo JPlag e MOSS, característica também observada na Tabela 2.

Tabela 4 - Grupo D: relação da técnica 1 e 4 com JPlag e MOSS

	N01		N04		JPLAG		MOSS	
	>20%	>70%	> 20%	>70%	> 20%	>70%	> 20%	>70%
Probl 25	26	1	30	7	42	12	33	3
Probl 26	7	0	34	3	12	4	12	7
Probl 27	11	1	16	11	29	15	12	11
Probl 28	6	1	27	15	45	21	26	7
Probl 29	8	1	16	3	41	4	29	2
Probl 30	6	1	50	4	70	14	44	1
Probl 31	9	0	28	28	29	10	22	4
Probl 32	9	1	1	0	4	1	1	0

5. Ferramenta para análise de similaridade no Moodle

A ferramenta foi utilizada ao longo do semestre, tendo sido recolhidos os dados apresentados na seção anterior, permitindo ao professor melhor conhecer o comportamento dos alunos e fazer a avaliação da metodologia empregada. Ao final do semestre, o comportamento das duplas de alunos que apresentaram índices recorrentemente altos de similaridade foi estudado pelo professor e pelos monitores que acompanharam as práticas laboratoriais. Alguns alunos foram convidados a esclarecer os motivos da alta similaridade. Da análise, pôde-se inferir que os altos índices nem sempre indicaram o plágio, embora a facilidade de comunicação com o uso da internet tenha sido um elemento facilitador desse comportamento. Com base na análise e no acompanhamento presencial do professor e dos monitores em laboratório, pode-se verificar que algumas ocorrências de similaridade se deram entre duplas de alunos que estudam regularmente em conjunto e que, por isso, recorre usualmente ao mesmo tipo de suporte, além de compartilharem ideias e soluções, atitude que não pode ser condenada. Em relação ao plágio, alguns elementos metodológicos podem contribuir com a redução da prática, como a proposição de exercícios distintos para diferentes alunos em uma mesma aula de laboratório e apresentação de enunciados apenas no seu início, pois os problemas, em sua maioria, eram propostos com bastante antecedência. Entretanto, faz-se necessário a realização de novas experimentações para alcançar o modelo mais conveniente.

Ainda sob a perspectiva metodológica, a experiência permitiu ao professor conhecer alguns dos recursos utilizados pelo aluno para o aprendizado. Verificou-se que grande parte da turma utiliza redes sociais para compartilhar informações e, muitas vezes, soluções para as atividades propostas. Se, por um lado, a utilização de recursos virtuais pode auxiliar no processo de aprendizagem, fomentando discussões coletivas sobre os problemas e suas soluções, em alguns casos, acabou servindo de suporte aos alunos que apresentam postura mais passiva, e que buscam respostas prontas.

Todos os cenários descritos e as observações realizadas estão sendo continuamente reavaliadas, visto que a ferramenta continua sendo utilizada e gerando novos dados. Espera-se que, com a contribuição da ferramenta para análise do comportamento dos alunos seja possível ao professor melhor conhecer os seus alunos e interferir de maneira mais eficiente para um melhor aprendizado e, consequentemente, para a redução da evasão e da reprovação.

6. Considerações finais e trabalhos futuros

Neste trabalho, identificamos que a ferramenta Sherlock não é eficaz para análise de similaridade de código-fonte sem que se faça, *a priori*, uma normalização criteriosa do código a ser processado. Para lidar com essa limitação, foram propostas e desenvolvidas técnicas de normalização que, de acordo com os resultados apresentados, são comparáveis as duas principais ferramentas de detecção de plágio referenciadas na literatura. As técnicas de normalização desenvolvidas, somadas à característica *open source* do Sherlock, que permite a sua inserção como um componente de um ambiente virtual preparado para práticas laboratoriais de programação, viabilizam a sua utilização de maneira eficaz na ferramenta BOCA-LAB.

Constituindo-se em um instrumento de análise para o professor de programação, a verificação de similaridade para todas as quatro técnicas de normalização propostas pode ser de grande utilidade, visto que elas apresentam variações que afetam a sensibilidade da comparação. Apesar de a técnica 4 ser a mais sensível em termos estruturais, é possível que seja importante para o professor, por exemplo, considerar os nomes de funções ou de palavras reservadas na comparação entre códigos, o que o remete para as outras técnicas. Além disso, é importante frisar que a ferramenta pode ser utilizada para finalidades diferentes da detecção de plágio. É possível, em especial nas turmas de iniciantes em programação, que a similaridade seja até mesmo desejável. Para exemplificar, o professor pode usar a comparação para verificar se as soluções apresentadas foram baseadas em um exemplo fornecido ou em uma orientação dada aos alunos ou se foram usadas lógicas e estruturas diferentes daquelas que foram propostas.

Como trabalhos futuros, pretende-se aprimorar as regras de normalização, investigando outras abordagens e formas de interferência no algoritmo Sherlock. Além disso, novos testes serão realizados para problemas mais avançados e que utilizem mais de um código fonte. Os dados obtidos neste trabalho serão usados para cruzar se existe relação entre a alta similaridade encontrada em alguns alunos e os índices de evasão e reprovação.

Referências Bibliográficas

Ahmadzadeh, M.; Mahmoudabadi, E.; Khodadadi F. (2011) "Pattern of Plagiarism in Novice Students' Generated Programs: An Experimental Approach" In Journal of Information Technology Education, v10.

Ahtiainen, Aleksi; Surakka, Sami; Rahikainen, Mikko. (2006). "Plaggie: Gnu-licensed source code plagiarism detection engine for java exercises". In Baltic Sea '06: Proceedings of the 6th Baltic Sea conference on Computing education research, pages 141–142, New York, NY, USA, 2006. ACM.

Bin-Habtoor, A. S.; Zaher, M. A. (2012) "A Survey on Plagiarism Detection Systems" International Journal of Computer Theory and Enginee-ring vol. 4, no. 2, pp. 185-188.

Burrows, S.;Tahaghoghi, S.M.M.;Zobel, J. (2007)."Efficient and effective plagiarism detection for large code repositories". Software-Practice & Experience, 37(2), 151-175.

Chen, Xin; Francia, Brent; Li, Ming; Mckinnon, Brian; Seker, Amit (2004). "Shared information and program plagiarism detection," IEEE Transactions on Information Theory, vol. 50, no. 7, pp. 1545–1551, 2004.

Dougiamas, M. & Taylor, P. (2003).“Moodle: Using Learning Communities to Create an Open Source Course Management System”. In D. Lassner& C. McNaught (Eds.), Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications 2003 (pp. 171-178).

Djuric, Z., Gasevic, D., "A Source Code Similarity System for Plagiarism Detection" The Computer Journal, 2012.

Grune, D. and Vakgroep, M. 1989. Detecting copied submissions in computer science workshops. Tech. rep., Informatica Faculteit Wiskunde & Informatica, Vrije Universiteit.

França, A. B.; Soares, J. M. (2011). “Sistema de apoio a atividades de laboratório de programação via Moodle com suporte ao balanceamento de carga”. In: XXII Simpósio Brasileiro de Informática na Educação, Aracaju - SE. Anais do XXII SBIE, 2011. p. 710-719.

Hage, J.; Rademaker, P.; Vugt, N. V. “A comparison of plagiarism detection tools.” Utrecht Uni-versity. Utrecht, The Netherlands, p. 28. 2010.

Kleiman, Alan B. (2007) “Análise e comparação qualitativa de sistemas de deteção de plágio em tarefas de programação.” Universidade Estadual de Campinas. Dissertação de Mestrado.

Lancaster, Thomas; Culwin, Fintan. (2001). "Towards an error free plagarism detection process".In Proceedings of the 6th annual conference on Innovation and technology in computer science education (ITiCSE '01).ACM, New York, NY, USA, 57-60.

MOSS (2012).“MOSS (Measure Of Software Similarity) plagiarism detection system”. Univ. California, Berkeley. Disponível em "http://theory.stanford.edu/~aiken/moss/". Acesso em: 01 de agosto de 2012.

Mota, M. P., Brito, S. R., Moreira, M. P., Favero, E. L. (2009) “Ambiente Integrado à Plataforma Moodle para Apoio ao Desenvolvimento das Habilidades Iniciais de Programação.” In: XX Simpósio Brasileiro de Informática na Educação. Florianópolis: SBC.

Onlinejudge (2012). Disponível em: https://github.com/hit-moodle/onlinejudge. Acesso em 01 de agosto de 2012.

Prechelt, L.; Malpohl, G. & Phlippsen, M. (2002). “Finding plagiarisms among a set of programs with JPlag”.J. UCS – Journalof Universal Computer Science.

Pike, R. (2012). The Sherlock Plagiarism Detector. Disponível em: “http://sydney.edu.au/engineering/it/~scilect/sherlock/” Acesso em: 01 de agosto de 2012.

Schleimer, S., Wiljerson, D. S., & Aiken, A. (2003) "Winnowing: local algorithms for document fingerprinting". In Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data (SIGMOD '03).ACM, New York, NY, USA.

VPL (2012). “Virtual Programming Lab” Disponível em: “http://vpl.dis.ulpgc.es/”. Acesso em: 01 de agosto de 2012.