Uso de algoritmos de similaridade para classificar códigos de acordo com a taxonomia SOLO em disciplinas de programação introdutória

Alexandre A. Barbosa[1,3], **Danilo V. B. da Costa**[1], **Allan L. Correia**[1], **Douglas L. L. Moura**[1], **Evandro B. Costa**[2,3]

[1]Universidade Federal de Alagoas - Campus Arapiraca (UFAL)
Caixa Postal 61 - Arapiraca - AL - Brasil

[2]Instituto de Computação - Universidade Federal de Alagoas (UFAL)
Caixa Postal 15.064 - 91.501-970 - Maceió - AL - Brasil

[3]Dep. de Sistemas e Computação - Universidade Federal de Campina Grande (UFCG)
Caixa Postal 10.106 - Campina Grande - PB - Brasil

{alexandre.barbosa, danilo.costa, allan.coreia}@arapiraca.ufal.br

{douglasllmoura, ebc.academico}@gmail.com

Abstract. *Traditionally, in programming disciplines practical coding activities are adopted. The analysis of students solutions is not trivial, it can be time consuming, and it is error prone and biased by the evaluator. Many researches have proposed the use of automatic mechanisms for code analysis. Some of these mechanisms evaluates the degree of similarity between two codes, usually the objective is the detection of plagiarism. In this work, similarity evaluation algorithms are used in order to classify a solution according to the SOLO taxonomy. It was possible to observe that with 1 or 4 reference solutions, the similarity algorithms had a very similar accuracy. Moreover, regardless of the number of solutions, the agreement between algorithms and experts wasn't satisfactory.*

Resumo. *Tradicionalmente, em disciplinas de programação são adotadas atividades práticas de codificação. A análise das soluções dos alunos não é trivial, pois pode ser demorada, sujeita ao viés e aos erros do avaliador. Visto isso, muitas pesquisas propuseram o uso de mecanismos automáticos para análise de códigos. Alguns mecanismos avaliam o grau de similaridade de dois códigos, em geral com a finalidade de detectar plágio. Neste trabalho, tais algoritmos são usados para classificar uma solução na taxonomia SOLO. Foi possível notar que, com 1 ou 4 soluções de referência, os algoritmos de similaridade retornaram uma acurácia muito semelhante. Além disso, independente da quantidade de soluções, a concordância entre algoritmo e especialista não foi satisfatória.*

1. Introdução

A programação é uma das competências fundamentais na área da computação. Contudo, apesar de sua importância, as disciplinas de programação têm apresentado um alto índice de reprovações e desistências. Na revisão sistemática da literatura apresentada em [Ramos et al.] a taxa média de reprovação em tais disciplinas é de aproximadamente

DOI: 10.5753/cbie.wcbie.2016.1107

32,6%. Os cenários descritos em [Barbosa et al. 2014] e [Noschang et al. 2014] citam reprovações de aproximadamente 50% dos alunos de disciplinas de programação.

Segundo diversos pesquisadores, a dificuldade do professor em fornecer um acompanhamento adequado a cada um dos alunos é um dos fatores motivadores dos diversos problemas existentes nas disciplinas de programação [Mota et al.].

Uma das principais estratégias usadas nas disciplinas de programação é a realização de atividades práticas de codificação. A análise das soluções propostas pelos alunos nestas atividades não é trivial, podendo essa avaliação ser demorada, sujeita ao viés e aos erros do avaliador. Para avaliar um programa, além de identificar se este é capaz de gerar as saídas adequadas para um conjunto de entrada, diversos parâmetros podem ser observados, tais como, estrutura do código, documentação, eficiência e manutenibilidade.

Dado o cenário descrito, muitas pesquisas têm sido desenvolvidas com o intuito de propor métodos ou ferramentas para facilitar o acompanhamento das atividades em disciplinas de programação. Muitas destas pesquisas, tais como, [Mota et al.], [Pelz et al. 2012], [Silva et al.] e [Raabe et al. 2015], propuseram o uso de mecanismos automáticos para análise de códigos, tendo como objetivo a redução da carga de trabalho do professor e a possibilidade de fornecer *feedback* de maneira mais ágil.

Neste trabalho, foram usados algoritmos de avaliação de similaridade para classificar códigos na taxonomia SOLO (*Structure of the Observed Learning Outcomes*), com a intenção de comparar as saídas geradas com as avaliações fornecidas por especialistas (professores e tutores), com respeito a acurácia da classificação, quando comparadas as avaliações dos professores, no contexto de ensino de programação.

Este trabalho segue a seguinte estrutura, na seção 2 são apresentados os conceitos necessários a compreensão da pesquisa, na seção seguinte são apresentados os trabalhos relacionados ao tema pesquisado, na seção 4 são exibidos os detalhes do experimento, a análise dos resultados e a discussão relativa à estes, finalmente na última seção são exibidas as conclusões e os trabalhos futuros.

2. Fundamentação teórica

2.1. Algoritmos de avaliação de similaridade

Na literatura podem ser encontradas diversas técnicas para avaliação de similaridade. Alguns trabalhos utilizam tais meios para realizar a detecção de plágio, enquanto em outros trabalhos estes meios são usados para descrever o grau de similaridade de duas entidades.

Nesta pesquisa, foram usadas quatro estratégias de detecção similaridade: Coeficiente Jaccard, *Text Edition Distance*, *Tree Edition Distance* e *Term Frequency-Inverse Document Frequency*. Todas têm como base a sintaxe de *tokens* extraídas da *Abstract Syntax Tree of a Python code*[1], que é a representação em árvore da estrutura sintática de um algoritmo escrito em *Python*. A descrição de tais estratégias é apresentada a seguir:

Coeficiente de Jaccard é uma técnica utilizada para calcular a semelhança entre conjuntos. Sua fórmula consiste em: dados dois conjuntos A e B, a divisão do tamanho da interseção de A e B pelo tamanho da união de A e B gera um

[1] https://docs.python.org/2/library/ast.html

valor que representa a similaridade existente entre os conjuntos. Sendo assim, $Coef_{Jaccard}(A, B) = \frac{|A \bigcap B|}{|A \bigcup B|}$.

***Term Frequency-Inverse Document Frequency* (TFIDF)** é uma medida estatística que considera a importância de um termo em um código. Nesta estratégia a importância de um termo aumenta a medida em que este se torna mais frequente. Para computar tal medida, conforme descrito em [Gaudencio et al. 2014], são criados vetores, x_i e x_j, para representar cada código. A distância Cosseno (D_C) entre x_i e x_j, dada por $\cos(x_i, x_j) = \frac{x_i \bullet x_j}{|x_i||x_j|}$, ou seja a distância corresponderá ao valor de cosseno do angulo Θ formado pelos dois vetores. Interpretando a medida da distância como a dissimilaridade entre dois código, a similaridade (S_C) é obtida pelo cálculo do complemento deste valor, ou seja, S_C = 1 - D_C.

Text Edition Distance é uma estratégia que dadas duas ou mais *strings*, irá gerar o número mínimo de operações (*delete*, *insert* e *rename*) que é necessário para transformar uma *string* na outra [Herranz et al. 2011]. O cálculo da similaridade é definido pela função de similaridade (S). Neste caso, dado A e B onde $S(A, B)$ = 1 - $dl(A, B)$ / $max(|A|, |B|)$, onde dl corresponde a distância de Levenshtein [Konstantinidis 2007] e max computa o maior tamanho entre $|A|$ e $|B|$.

Tree Edition Distance é uma forma de quantificar o quão dissimilar é um conjunto de *strings*. Cada conjunto é transformado em uma árvore e é contada a quantidade mínima de operações (*delete*, *insert* e *rename*) necessárias para transformar uma árvore em outra. O valor obtido corresponde a dissimilaridade (ds) entre os conjuntos, a similaridade s é então o complemento $(1 - ds)$.

2.2. Medidas de comparação

As medidas de comparação Kappa de Cohen (*Cohen's Kappa*) e distância euclidiana foram adotadas para analisar o quão próximas são as classificações dos algoritmos, quando comparadas as dos especialistas. Tais medidas são descritas a seguir:

Kappa de Cohen Sejam dados dois conjuntos de classificações, a intensidade da concordância entre estes pode ser dada pela medida do coeficiente Kappa de Cohen. Este critério mede o grau de concordância além do que seria esperado pelo acaso. $Kappa = \frac{P_o - P_c}{P - P_c}$, onde P_o é a proporção de observações, P_c é a proporção de concordância ao acaso e P é o número de instâncias. Os valores de Kappa podem variar no intervalo [-1, 1], onde: 0 (zero) indica nenhuma concordância, ou concordância foi exatamente a esperada pelo acaso; valores negativos sugerem discordância, sem interpretação de intensidade; e valores positivos sugerem concordância, onde quanto maior o valor obtido maior a intensidade de concordância.

Distância Euclidiana Sejam dados dois pontos em um espaço n-dimensional, a distância euclidiana é dada por $DE = \sqrt{\sum_{i=1}^{d} |P_i - Q_i|^2}$, onde P e Q são pontos e o i representada cada coordenada destes pontos. Para [Meyer 2002] esta é uma medida de dissimilaridade que representa a distância entre duas instâncias, cujas posições são determinadas por suas coordenadas.

2.3. Taxonomia SOLO

De acordo com [Biggs and Collis 1980], a taxonomia SOLO permite classificar a complexidade da formalização do pensamento em um sistema de categorias. O modelo original

concebido, segundo [Biggs and Collis 2014], trata de cinco níveis, que são: *extended abstract, relational, multistructural, unistructural* e *prestructural*. Na Tabela 1 é exibida a descrição para classificação de um elemento em cada classe existente na taxonomia.

Tabela 1. Descrição das classes existentes na Taxonomia SOLO

Classificação SOLO	Descrição
Extended abstract	Resposta cumpre com 100% dos requisitos solicitados e vai além, empregando conceitos mais avançados
Relational	Resposta cumpre com 100% dos requisitos solicitados
Multistructural	Resposta atinge alta porcentagem dos requisitos solicitados, mas não os cumpre totalmente
Unistructural	Resposta atinge porcentagem intermediária dos requisitos solicitados, só os cumpre parcialmente
Prestructural	Resposta atinge baixa porcentagem dos requisitos solicitados, limitando-se aos aspectos mais básicos.

Para os objetivos deste trabalho foram adotados apenas os níveis *relational, multistructural, unistructural* e *prestructural*. Justificamos esta escolha, pois como as soluções analisadas são códigos de um curso introdutório, não eram esperadas soluções que utilizassem conceitos mais avançados.

2.4. Testes estatísticos

Dois testes estatísticos foram utilizados neste trabalho, sendo estes o teste de *Shapiro-Wilk*, que verifica se os dados possuem distribuição normal, e o teste de *Mann-Whitney-Wilcoxon*, que é um teste de hipótese aplicável em dados que não possuem distribuição normal. Dados dois conjuntos de dados, o teste *Mann-Whitney-Wilcoxon* verifica se os conjuntos tendem a um mesmo valor (hipótese nula), ou se um destes conjuntos possui valores maiores que os presentes no outro conjunto (hipótese alternativa). A interpretação dos p-valores em ambos os testes, ocorre da seguinte maneira, caso o p-valor obtido no teste seja menor que α (tipicamente α = 0.05) a hipótese nula pode ser negada, caso contrário, se o p-valor é maior que α, não é possível negar a hipótese nula.

3. Trabalhos relacionados

Nesta seção são descritos [Gaudencio et al. 2014], [Silva et al.], [Pelz et al. 2012] e [Raabe et al. 2015], outros trabalhos foram encontrados, porém estes são os que apresentaram maior relação com a presente pesquisa.

Em [Gaudencio et al. 2014] é levantado o questionamento se avaliadores automáticos (algoritmos de avaliação de similaridade) podem comparar códigos de alunos tão bem quanto especialistas. Na pesquisa, o experimento conduzido buscou fornecer para professores e avaliadores automáticos um conjunto de códigos e compará-los. A comparação foi realizada da seguinte maneira, dado um trio de códigos (C1, R, C2), onde R é uma solução de referência e C1 e C2 são as soluções que devem ser comparadas, indique C1 como mais semelhante a R, C2 como mais semelhante a R ou considere que ambos são igualmente similares ou diferentes de R. Como resultados do experimento realizado, obteve-se uma concordância entre professores no intervalo de 62% a 95%, e entre a maioria professores e algoritmos as taxas de concordância com o melhor resultado dos algoritmos é sempre superior a 90%.

Na pesquisa conduzida em [Silva et al.] é proposto um arcabouço capaz de combinar diferentes tipos de analisadores de código. Nos analisadores utilizados podem ser observados, construtos, estrutura do código, corretude sintática, corretude semântica e outros parâmetros. Para os dois primeiros parâmetros foram utilizados a métrica de distância de Levenshtein e a técnica *Tree Inclusion* respectivamente. O arcabouço foi usado para avaliar 816 códigos Python, propostos como soluções submetidas por discentes de um curso de Sistemas de Informação. As soluções foram analisadas com base nos seguintes tópicos: variáveis e comandos de entrada e saída; operadores e expressões; estruturas de seleção; e estruturas de repetição. Como resultado, foi possível identificar grupos de estudantes, como alunos que não conseguiram construir soluções estruturalmente coesas. Também foi possível identificar alunos que superam as expectativas como aluno iniciante em programação.

No trabalho [Pelz et al. 2012], é proposto um mecanismo para correção automática de códigos. Esse mecanismo realiza a correção em quatro etapas, são elas: verificação sintática, verificação da presença de comandos obrigatórios (condicionais, laços de repetição), similaridade entre o código do aluno e as soluções do professor e a execução do programa para verificar se as saídas estão corretas. Para validar a proposta foi conduzido um experimento que contou com 93 alunos, neste experimento as correções automáticas estáticas utilizaram o cálculo de distância de Levenshtein para determinar a similaridade dos códigos. Como resultado os autores descrevem que é possível detectar códigos criados apenas para satisfazer o mecanismo de correção. Contudo, descreve-se também que a solução funciona apenas para códigos simples.

A proposta descrita em [Raabe et al. 2015], consiste em construir um gerador de dicas sobre erros cometidos por estudante em exercícios de programação. Para isso foi utilizado uma combinação entre análise estática e dinâmica na correção dos trabalhos. Durante a análise estática, um ponto chave foi a inserção de padrões de erros comuns que os estudantes praticam. Esse erros foram inferidos de modo semiautomático a partir de uma base de dados com 1429 problemas que já tinham sido resolvidos pelos estudantes. Para analisar os erros, algoritmos *Tree Walkers* foram utilizados para percorrer a estrutura sintática do código e com isso gerar uma mensagem de ajuda ao aluno. Os experimentos foram realizados com 23 estudantes, com objetivo de coletar o impacto que as dicas geraram durante a resolução dos problemas. Por fim, os autores notaram que as dicas não tiveram um impacto satisfatório.

4. Metodologia da pesquisa

4.1. Questões de pesquisa e hipóteses

A condução desta pesquisa foi norteada para responder as seguintes questões de pesquisa:
QP1 - Utilizando os avaliadores de similaridade, a classificação fornecida é semelhante a classificação de um especialista?

- **Hipótese nula**: Os algoritmos fornecem uma classificação com alta concordância em relação aos especialistas;
- **Hipótese alternativa**: Os algoritmos fornecem uma classificação com baixa concordância em relação aos especialistas;

QP2 - A quantidade de soluções de referência adotadas para identificar o grau de similaridade tem influência na concordância entre as classificações?

- **Hipótese nula**: A concordância na classificação não difere independente da quantidade de soluções de referência adotada;
- **Hipótese alternativa**: Quanto maior seja a quantidade de soluções de referência, melhor será a concordância na classificação;

4.2. Descrição dos dados

O conjunto de dados utilizado para a pesquisa é formado por: um conjunto de enunciados de problemas de programação; um conjunto de códigos submetidos por discentes como soluções aos enunciados; um conjunto de códigos propostos como soluções de referência; e um conjunto de classificações fornecido pelos especialistas.

O conjunto de enunciados de problemas de programação é formado por quatro enunciados, estes foram fornecidos como exercícios práticos em uma disciplina introdutória de programação.

O conjunto de códigos submetidos pelos discentes consiste em um total de 438 soluções, sendo 131 para a primeira questão, 118 para a segunda, 108 para terceira e 81 para quarta questão. Todas as soluções foram criadas utilizando a linguagem Python em sua versão 3. As questões possuem níveis de dificuldade diferentes, sendo a primeira questão de mais fácil solução e a quarta a mais complexa dentre os exercícios.

O conjunto de códigos propostos como soluções de referência é composto de 16 programas, de modo que cada questão possua um código de referência para cada classe da taxonomia SOLO. Sendo assim, existe um código *relational*, um *multistructural*, um *unistructural* e um *prestructural* para cada questão.

O conjunto de classificações foi fornecido por três especialistas, sendo dois professores e um tutor de um curso à distância. As classificações foram realizadas em um momento anterior a realização do experimento. Todos os especialistas receberam uma descrição da taxonomia SOLO, o conjunto de enunciados e os códigos para classificação.

4.3. Descrição do experimento

O experimento foi realizado em duas rodadas, em ambas os algoritmos de similaridade foram executados sobre todos os códigos propostos pelos discentes, na primeira rodada apenas uma solução de referência foi utilizada e na segunda para cada questão quatro soluções foram usadas. A comparação das classificações fez uso das medidas de comparação Kappa de Cohen e Distância Euclidiana.

Na primeira rodada a classificação foi obtida através de intervalos especificados para o coeficiente de similaridade obtido. Uma solução foi classificada como *relational* quando a similaridade estava no intervalo $(0.75, 1.00]$, como *multistructual* quando no intervalo $(0.50, 0.75]$, como *unistructual* quando no intervalo $(0.25, 0.50]$ e como *prestructual* quando no intervalo $(0.0, 0.25]$.

Na segunda rodada o valor máximo obtido no calculo de similaridade determinou a classe SOLO de cada solução. Desta forma, por exemplo, caso o maior coeficiente de similaridade tenha sido obtido em relação a solução de referência *relational* o código do discente seria classificado como *relational*, o mesmo para as outras classes.

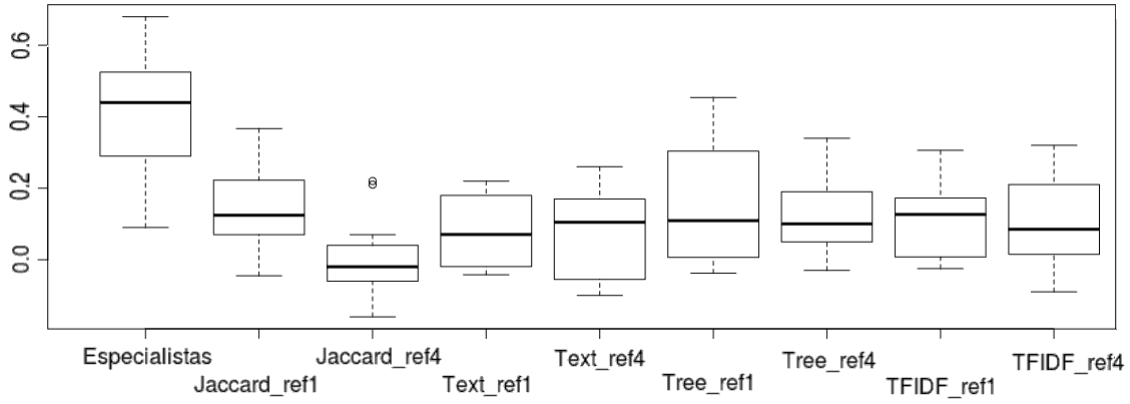
4.4. Resultados e discussão

Nesta seção serão apresentados os resultados obtidos na pesquisa desenvolvida, além de uma série de discussões relacionadas a estes resultados. É importante observar que o Kappa de Cohen é uma medida de concordância, sendo assim quanto maior o valor obtido, maior será a concordância existente no conjunto e vice-versa. Já a Distância Euclidiana é uma medida de dissimilaridade, para essa medida quanto menor o valor mais similar são os conjuntos comparados.

Nas Figuras 1 e 2 são exibidos os gráficos boxplot que resumem os resultados obtidos para o conjunto de especialistas e para cada algoritmo. As duas rodadas de execução do experimento, ou seja quando apenas uma solução de referência foi adotada e quando quatro soluções de referência foram adotadas, são indicadas pelos termos 'ref1' e 'ref4'. Sendo assim, por exemplo, Jaccard_ref1 se refere ao resultado do algoritmo Jaccard quando apenas uma solução de referência foi utilizada, e Jaccard_ref4 se refere ao resultado do algoritmo Jaccard quando as quatro soluções de referência foram usadas.

Para a medida Kappa, tomando como base a observação das medianas exibidas na Figura 1, é possível sugerir que a concordância entre os especialistas é superior a concordância de todos os algoritmos, seja para primeira ou segunda rodada dos experimentos. Apenas o algoritmo Tree obteve, na primeira rodada ("*Tree_ref1*"), um valor máximo (Kappa=0.454), superior a mediana obtida entre os especialistas (Kappa=0.440). Para todos os outros algoritmos o máximo dos valores obtidos foram inferiores a mediana dos especialistas. Desta forma, de acordo com a observação da medida Kappa, sugere-se que os algoritmos fornecem uma classificação com baixa concordância em relação aos especialistas.

Figura 1. Resultados da medida de comparação Kappa de Cohen.

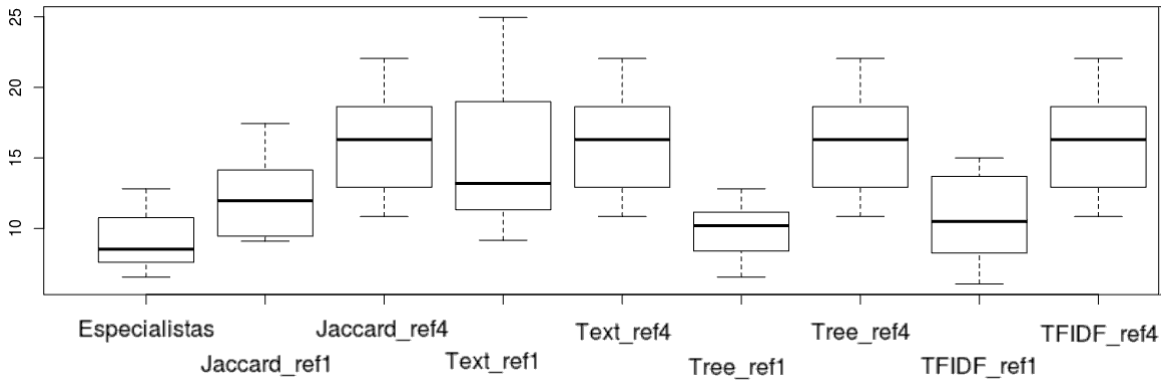


Em relação a variação sobre a quantidade de soluções de referência adotadas, em geral, tem-se um cenário onde as medianas estão bastante próximas. Para a maioria dos algoritmos, com exceção de "*Text*", o uso de uma maior quantidade de soluções de referência teve impacto negativo, pois os valores de máximo, quartis, mediana e mínimo, são em geral menores do que os obtidos com uma única solução de referência. Desta forma, de acordo com a observação da medida Kappa, sugere-se que o uso de uma maior quantidade de soluções de referência não melhora a classificação fornecida pelos algoritmos.

Para a medida Distância Euclidiana, tomando como base a observação das medi-

anas exibidas na Figura 2, é possível sugerir que a dissimilaridade entre os especialistas é menor que a existente em relação aos algoritmos. Quando apenas uma solução de referência foi utilizada, os algoritmos "*Tree_ref1*" (mediana DE=10.20) e "*TFIDF_ref1*" (mediana DE=10.50) foram os que obtiveram valores mais próximos aos obtidos pelos especialistas (mediana DE=8.54). Ao analisar os resultados também pode-se perceber que o máximo e o mínimo dos especialistas e do algoritmo "*Tree_ref1*" são iguais [6.56, 12.81]. O valor mínimo obtido com o algoritmo "*TFIDF_ref1*" (mínimo DE=6.08) é menor do que o mínimo obtido pelos os especialistas (mínimo DE=6.56). Assim, ao analisar os resultados sugere-se que o algoritmo *Tree* usando apenas uma solução de referência foi o que obteve os melhores resultados. Desta forma, de acordo com a observação da medida Distância Euclidiana, sugere-se que os algoritmos, com exceção de *Tree*, fornecem uma classificação com maior dissimilaridade do que os especialistas.

Figura 2. Resultados da medida de comparação Distância Euclidiana.



Em relação à variação sobre a quantidade de soluções de referência adotadas, em todos os casos ao adicionar as quatro soluções de referência, as medianas dos resultados obtidos pelos algoritmos é menor do que os obtidos na primeira rodada dos experimentos. Desta forma, de acordo com a observação da medida Distância Euclidiana, sugere-se que o uso de uma maior quantidade de soluções de referência não melhora a classificação fornecida pelos algoritmos.

Com estes resultados, é possível sugerir que os especialistas não adotam uma estratégia de comparação que se baseie em apenas uma solução de referência. Uma vez que existem várias maneiras diferentes de construir um algoritmo que resolva um problema, é natural observar o uso dos algoritmos de similaridade não tenha alta concordância em relação aos especialistas quando apenas uma solução de referência é adotada. Também é possível sugerir que uma maior quantidade de soluções de referência não melhorou a concordância entre as classificações fornecidas pelos algoritmos, quando estas são comparadas as de especialistas. Tal resultado difere do comportamento que era esperado, uma vez que mais soluções de referência foram adotadas era de se esperar que a classificação melhorasse, pois um universo maior de códigos aceitáveis estava sendo fornecido.

Para verificar as hipóteses associadas as questões de pesquisa apresentadas foi utilizado o teste Mann-Whitney-Wilcoxon, pois de acordo com o teste de normalidade de Shapiro-Wilk os dados não apresentavam distribuição normal. Os p-valores obtidos são exibidos na Tabela

Sendo assim, para QP1 a hipótese nula pode ser rejeitada, sugerindo que a hipótese alternativa seja válida, ou seja, os algoritmos fornecem uma classificação com baixa con-

Tabela 2. Resultados do teste de Mann-Whitney-Wilcoxon para as hipóteses.

	p-valor ($\alpha = 0.05$) (hipóteses para QP1)	p-valor ($\alpha = 0.05$) (hipóteses para QP2)
Jaccard	0.009813	6.157e-09
Text	2.2e-16	3.661e-16
Tree	0.04466	0.1887
TF-IDF	2.2e-16	0.9999

cordância em relação aos especialistas. Para QP2, é possível a hipótese nula para dois algoritmos (Jaccard e Text) e não é possível rejeitar a hipótese nula para os outros dois (Tree e TFIDF). Sendo assim, para Jaccard e Text, a classificação fornecida pelos algoritmos é diferente quando há variação na quantidade de soluções de referência adotada, pela análise gráfica apresentada sugere-se que a acurácia da classificação piorou. Já para Tree e TFIDF a classificação fornecida pelos algoritmos não é diferente quando há variação na quantidade de soluções de referência adotada.

4.5. Ameaças à validade

As seguintes ameaças à validade estão presentes nesta pesquisa:

- Quantidade de questões - Foram usados apenas quatro questões, onde somente os conceitos mais básicos são explorados. Uma maior quantidade de questões pode cobrir mais tópicos e gerar resultados distintos daqueles aqui exibidos;
- Quantidade de especialistas - O universo de especialistas é bastante vasto para o contexto de programação, nesta pesquisa foram capturadas classificações de apenas três especialistas, tal quantidade pode não ser representativa;
- Soluções de referência - A comparação dos códigos dos discentes foi realizada com soluções de referência nas quatro classes SOLO, é possível que a adoção de mais soluções da classe *relational* gere resultados distintos daqueles aqui exibidos;
- Métodos de classificação - Os intervalos de classificação usados em relação ao valor de similaridade obtido podem não ter sido os intervalos ideais, outros intervalos podem gerar resultados distintos daqueles aqui exibidos;
- Linguagem - Apenas códigos em Python 3 foram observados, códigos escritos em outra linguagem podem alterar o cálculo de similaridade e gerar resultados distintos daqueles aqui exibidos;

4.6. Replicação do experimento

Para possibilitar a replicação do experimento, os dados utilizados, a implementação dos algoritmos, as planilhas de resumo dos resultados e o código R da análise, estão disponíveis em um repositório [2] do GitHub.

5. Conclusões e trabalhos futuros

Neste trabalho, foram apresentados experimentos que utilizavam algoritmos de avaliação de similaridade com a finalidade de classificar uma solução de acordo com a taxonomia SOLO. O uso de tais algoritmos quando apenas uma solução de referência foi adotada, foi

[2] https://github.com/nemo-research/walgprog-2016

bastante similar aos resultados obtidos com o uso de mais soluções de referência. Na segunda etapa do experimento, para cada questão foram utilizadas referências pertencentes as classes SOLO *relational, multistructural, unistructural* e *prestructural*. A classificação fornecida em qualquer um dos cenários, em geral, não obteve uma boa concordância com os especialistas.

Novas questões de pesquisa surgiram após a conclusão da investigação apresentada neste artigo. Uma possibilidade de investigação se dá na adoção de uma maior quantidade de soluções de referência classificadas apenas como *relational*, ou seja soluções totalmente corretas, indagamos se com isso a classificação poderá melhorar. Outra possibilidade de pesquisa levantada a partir dos resultados obtidos é o uso de algoritmos de similaridade em combinação com outras estratégias, dessa forma outros critérios adotados no processo de correção/avaliação de códigos poderiam ser contemplados.

Referências

Barbosa, A. d. A., Ferreira, D. Í., and Costa, E. B. (2014). Influência da linguagem no ensino introdutório de programação. In *Anais do SBIE*, pages 612–621.

Biggs, J. and Collis, K. F. (1980). Solo taxonomy. *Education News*, 17(5):19–23.

Biggs, J. B. and Collis, K. F. (2014). *Evaluating the quality of learning: The SOLO taxonomy (Structure of the Observed Learning Outcome)*. Academic Press.

Gaudencio, M., Dantas, A., and Guerrero, D. D. (2014). Can computers compare student code solutions as well as teachers? In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, SIGCSE '14, pages 21–26.

Herranz, J., Nin, J., and Sole, M. (2011). Optimal symbol alignment distance: A new distance for sequences of symbols. *IEEE Transactions on Knowledge and Data Engineering*, 23(10):1541–1554.

Konstantinidis, S. (2007). Computing the edit distance of a regular language. *Information and Computation*, 205(9):1307–1316.

Meyer, A. d. S. (2002). *Comparação de coeficientes de similaridade usados em análises de agrupamento com dados de marcadores moleculares dominantes*. PhD thesis, Universidade de São Paulo.

Mota, M. P., de Brito, S. R., et al.

Noschang, L. F., Fillipi Pelz, E. A., and Raabe, A. L. (2014). Portugol studio: Uma ide para iniciantes em programação. In *Anais do CSBC/WEI*, pages 535–545.

Pelz, F. D., de Jesus, E. A., and Raabe, A. L. (2012). Um mecanismo para correção automática de exercícios práticos de programação introdutória. In *Anais do SBIE*.

Raabe, A., de Jesus, E. A., Hodecker, A., and Pelz, F. (2015). Avaliação do feedback gerado por um corretor automático de algoritmos. In *Anais do SBIE*, page 358.

Ramos, V., Freitas, M., et al. A comparação da realidade mundial do ensino de programação para iniciantes com a realidade nacional: Revisão sistemática da literatura em eventos brasileiros.

Silva, M. T., Costa, E. D. B., et al. Um Arcabouço para Construção de Mecanismos de Análise de Códigos de Programação Introdutória.