Investigando o uso de múltiplas soluções de referência para classificar códigos usando algoritmos de similaridade

João Vitor Silva Oliveira[1], Alexandre de Andrade Barbosa[1]

[1]Universidade Federal de Alagoas (UFAL)
Arapiraca – AL – Brazil

joao130693@gmail.com, alexandre.barbosa@arapiraca.ufal.br

Abstract. *In context of programming learning environments, many students have difficulties to understand the contents, causing a lack of performance in the discipline. This way, in many researches methodologies and tools are proposed to provide fast and effective feedback. In this investigation, was verified if the number of reference solutions have positive influence in classification of solutions. At the end of the investigation, it was possible to notice that, in a few cases, similarity algorithms provide an fair agreement with the experts, the use of a larger number of reference solutions did not constitute a significant improvement.*

Resumo. *No contexto de ensino e aprendizagem de programação, muitos alunos têm apresentado dificuldades na compreensão dos conteúdos, o que ocasiona mau desempenho na disciplina. Visto isso, muitas pesquisas investigam metodologias e propõem ferramentas para fornecer um feedback rápido e efetivo. Nessa investigação, buscou-se identificar se a quantidade de soluções de referência adotadas influência positivamente a classificação das soluções realizadas por um classificador automático. Ao final da investigação, foi possível notar que em poucos casos o classificador automático fornece uma concordância equivalente aos especialistas, e o uso de uma maior quantidade de soluções de referência não consistiu em um ganho significativo.*

1. Introdução

Os conceitos relacionados com a atividade de programação são extremamente importantes para o aluno de computação no decorrer do curso [Cristovão 2008], [Barbosa 2014]. De acordo com o currículo de referência da Sociedade Brasileira de Computação [SBC 2005], diversas disciplinas que tratam do ensino de algoritmos e linguagens de programação devem estar presentes no currículo de qualquer curso de computação. Durante o processo de aprendizagem de programação, muitos alunos apresentam dificuldades em entender os conceitos envolvidos e acabam obtendo um mau desempenho nestas disciplinas [Moreira e Favero 2009], [Mason e Cooper 2014].

O cenário nacional foi analisado na revisão sistemática da literatura apresentada em [Ramos 2015], onde a taxa média de reprovação nas disciplinas de programação é de aproximadamente 32,6%. Outros locais apresentam taxas de reprovação de aproximadamente 50% [Barbosa 2014], [Barbosa 2011], [Noschang 2014].

Atividades práticas de codificação são fundamentais para o aprendizado de programação, os estudantes devem ter o hábito de solucionar exercícios e direcionarem

seus estudos à codificação [Araujo 2013], [Paes 2013]. Diversos pesquisadores tentam desenvolver metodologias de ensino ou ferramentas para diminuir as dificuldades presentes no ensino/aprendizagem de programação [Cristovão 2008]. O uso de ferramentas que fornecem avaliadores automáticos em disciplinas de programação vem sendo bastante pesquisado. Alguns exemplos de soluções pesquisadas são: avaliadores de códigos, ferramentas para animação de algoritmos e identificadores de plágio.

Nesta pesquisa, foram usados algoritmos de avaliação de similaridade para classificar códigos com base na taxonomia SOLO (*Structure of the Observed Learning Outcomes*). Dois objetivos foram adotados, sendo um comparar as classificações geradas automaticamente com as classificações fornecidas por um conjunto de especialistas, e o segundo verificar se a variação da quantidade de soluções de referência adotadas tem impacto na qualidade das classificações geradas automaticamente.

Este artigo está organizado da seguinte maneira, na Seção 2 são apresentados os fundamentos teóricos necessárias para compreensão da pesquisa. Na Seção 3 são apresentados os trabalhos relacionados ao contexto desta pesquisa. A metodologia adotada, bem como os resultados obtidos e a discussão relacionada são exibidos na Seção 4. Por fim, na Seção 7 são descritas conclusões relativas a essa pesquisa.

2. Fundamentação Teórica

2.1. Algoritmos de Avaliação de Similaridade

Para este artigo foram utilizados algoritmos de similaridade com a finalidade de medir o grau de semelhança entre duas entidades. Foram usadas quatro estratégias para a identificação de similaridade, são elas: Coeficiente de *Jaccard*, similaridade baseada na edição de texto (*Text Edit Distance*), similaridade baseada na edição de árvore (*Tree Edit Distance*) e a frequência de termos (*Term Frequency-Inverse Document Frequency*) (TFIDF). A seguir são apresentadas as descrições de cada estratégia utilizada:

Coeficiente de Jaccard é uma técnica que é utilizada para calcular a similaridade entre duas ou mais instancias. O valor de similaridade está contido dentro de um intervalo de [0,1]. Dado dois conjuntos A e B, a divisão do tamanho da intersecção de A e B pelo tamanho da união de A e B, resultará no valor de similaridade entre os dois conjuntos. Tal como é exibido na Equação 1:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} \quad (1)$$

Similaridade baseada na edição de texto corresponde ao cálculo do custo mínimo de uma sequência de operações que transformam uma cadeia X em outra cadeia Y. Segundo [Konstantinidis 2007], o cálculo de similaridade é definido pela função (S), onde dado duas strings A e B distintas, o resultado de $S_{A,B} = 1 - dl_{A,B} / max(|A|,|B|)$, onde dl representa a distância de *Levensthein* e o max calcula o maior tamanho entre $|A|$ e $|B|$. O resultado de $F_{A,B}$ é o menor número de operações de inserção, remoção e substituição para

tornar duas cadeias iguais.

Similaridade baseada na edição de árvore é uma técnica que dado um par de árvores (A,B) é possível determinar quantos passos são necessários para transformar determinada árvore em outra aplicando as operações de edição, são elas: inserção, remoção e substituição de nós. O valor obtido corresponde a dissimilaridade (ds) entre os conjuntos, a similaridade s é dada pelo complemento $(1 - ds)$.

Frequência de termos é uma medida estatística que utiliza a frequência com que um termo/palavra é exibido em uma *string* para determinar sua importância. Quanto maior a frequência da palavra em um código qualquer maior será sua relevância/importância. Existem diferentes estratégias na literatura para computar tal medida. Neste artigo, foi utilizado a estratégia baseada na medida cosseno [Gaudencio 2014]. Para determinar a similaridade entre dois códigos, são criados dois vetores x_i e x_j para representar cada código. A distância Cosseno (DC) entre x_i e x_j, é a distância se dá pelo ângulo θ formado entre os dois vetores, tal como na Equação 2.

$$cos(x_i, x_j) = \frac{x_i \bullet x_j}{|x_i||x_j|} \quad (2)$$

2.2. Medidas de Comparação

Nessa pesquisa foram utilizadas métricas de comparação *Cohen's Kappa* e Distância Euclidiana com o objetivo de medir o quão próximo são as classificações dos algoritmos quando comparados com a dos especialistas. A seguir as descrições de cada métrica:

Kappa de Cohen é um método estatístico que avalia o nível de concordância além do que seria esperado tão somente pelo acaso entre dois conjuntos de classificações. Sejam dados dois conjuntos de classificações fornecidos por dois especialistas $E1$ e $E2$, deseja-se saber o quanto $E1$ e $E2$ concordam entre si. A intensidade da concordância entre $E1$ e $E2$ pode ser dada pela medida do coeficiente Kappa de Cohen. Este critério mede o grau de concordância além do que seria esperado pelo acaso. O Kappa de Cohen é adequado para obter a concordância entre classificações fornecidas por especialistas par a par. Os valores de Kappa de Cohen variam em um intervalo de [-1,1], onde os valores negativos representam discordância, os valores positivos representam concordância, onde quanto maior o valor obtido maior será a concordância entre os conjuntos, e o valor 0 (zero) que representa que a concordância foi exatamente a esperada pelo acaso.

Distância Euclidiana corresponde à distância em linha reta entre dois pontos em um espaço euclidiano n-dimensional. Sejam dados dois pontos P e Q em um espaço n-dimensional, a distância euclidiana entre estes pode ser calculada de acordo com a Equação 3, onde p_i e q_i representam respectivamente uma coordenada do ponto P e Q.

$$DistanciaEuclidiana = \sqrt{\sum_{i=1}^{d} |p_i - q_i|^2} \quad (3)$$

2.3. Taxonomia SOLO

Desenvolvida por [Biggs e Collis 1980], a taxonomia SOLO é um modelo que estabelece um conjunto de categorias, que permitem classificar a complexidade da compreensão/clareza sobre determinados assuntos. A taxonomia SOLO está dividida em cinco classes crescentes de complexidade, que são: *Pre-structural, Unistructural, Multi-structural, Relational e Extended abstract*. No Quadro 1 segue a descrição de cada classe pertencente a taxonomia SOLO e a interpretação adotada para cada uma delas.

Quadro 1. Quadro do modelo adaptado da 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.

Uma vez que era desejado apenas identificar se uma solução atendia todos os requisitos solicitados, sem necessidade de análise do uso de conceitos mais avançados, não foram diferenciadas as soluções *Relational* e *Extended abstract*. Dessa forma, caso uma solução tenha atendido todos os critérios solicitados, independente dos constructos utilizados, ela deveria ser classificada como *Relational*.

3. Trabalhos Relacionados

Nesta seção são descritos os artigos de [Barbosa 2016], [Pelz 2012] e [Maciel 2012] que apresentaram maior relação com a presente pesquisa. A proposta usada neste artigo se difere destes trabalhos principalmente nas medidas utilizadas para calcular a distância entre os arquivos para determinar a similaridade. Enquanto nos trabalhos relacionados de [Pelz 2012] e [Maciel 2012] o grau de similaridade é dado pelo cálculo de distância das medidas *Levenshtein* e Cosseno, neste artigo são utilizadas um conjunto maior de medidas para calcular o grau de similaridade. Em relação ao trabalho de [Barbosa 2016] foram utilizadas soluções de referência com diferentes níveis de corretude, no presente artigo todas as soluções de referências utilizadas são soluções totalmente corretas.

Na pesquisa apresentada em [Barbosa 2016], foi levantado um questionamento se a utilização de algoritmos de similaridade poderiam fornecer uma classificação equivalente a avaliação de um especialista de acordo com a taxonomia SOLO. Foram

realizadas duas rodadas de experimentação, sendo estas: o uso dos algoritmos de similaridade baseando-se em uma única solução de referência; e o uso dos algoritmos de similaridade baseando-se em quatro soluções de referência com diferentes graus de corretude. A comparação das classificações ocorreu através do uso de medidas estatísticas de comparação entre as classificações obtidas com os algoritmos e aquelas fornecidas por especialistas. Ao final do experimento, os autores descrevem que os resultados obtidos pelos algoritmos tanto com uma solução quanto com quatro soluções de referência foram bastante similares, e que os algoritmos de similaridade não tiveram uma boa concordância com nenhum dos especialistas.

No artigo desenvolvido por [Pelz 2012], é proposta uma técnica para a avaliação automática de exercícios práticos de programação. Para isso, a avaliação dos algoritmos foram divididas em quatro etapas, são elas: verificação sintática dos algoritmos, verificação da presença de comandos obrigatórios (*loops*, condicionais), a similaridade entre os códigos dos alunos e as soluções do professor e, por fim, a execução de programas para verificar se as saídas geradas correspondem com as saídas esperadas. Para a validação do mecanismo, foi realizado um experimento com 93 alunos de três turmas de ingressantes no curso de Ciência da Computação. Para realizar as correções, foi utilizada a métrica de cálculo de distância de *Levenshtein* para medir o grau de similaridade dos algoritmos. Ao final do experimento, os autores descrevem que com os resultados obtidos é possível fornecer a avaliação de pequenos exercícios práticos de programação.

O artigo proposto por [Maciel 2012], fornece uma avaliação para medir a sensibilidade do algoritmo *Sherlock* 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 (valores literais, nomes de variáveis, comentários). Para isso, foram abordadas quatro técnicas de normalização, são elas: remoção de linhas e espaços vazios, remoção de todos os caracteres situados entre aspas, remoção de todas as palavras reservadas e remoção de todos os valores literais e variáveis. Para validar a proposta, os testes foram analisados de duas formas, a primeira com base em códigos gerados manualmente e, a segunda com algoritmos desenvolvidos por alunos. Ao decorrer de um semestre foi utilizada uma ferramenta de análise de similaridade, denominada BOCA Lab, em conjunto com o *Moodle*. Os resultados obtidos foram que a ferramenta *Sherlock* não é eficaz para análise de similaridade sem antes ter feito uma normalização adequada dos códigos.

4. Metodologia

4.1. Questões de pesquisa

A condução desta pesquisa foi norteada para responder as questões de pesquisa:

QP1 - Adotando N soluções de referências classificadas como *Relational*, a avaliação fornecida é equivalente a avaliação dos especialistas?

- **H_0:** A classificação automática é igual a dos especialistas.
- **H_1:** A classificação automática não é igual a dos especialistas.

QP2 - A variação da quantidade de soluções de referências adotadas tem impacto na

qualidade das classificações geradas automaticamente?

- **H_0:** As classificações são iguais independente da variação da quantidade de soluções de referências adotadas.
- **H_1:** As classificações melhoram quando mais soluções de referências são adotadas.

4.2. Descrição dos dados

O conjunto de dados utilizado nessa pesquisa é composto por: um conjunto de enunciados de exercícios de programação; um conjunto de códigos submetidos pelos discentes como soluções aos enunciados; um conjunto de algoritmos propostos como soluções de referência e um conjunto de classificações fornecido pelo especialistas.

O conjunto de enunciados de exercícios de programação é composto por quatro enunciados, estes foram fornecidos como uma atividade prática em uma disciplina de introdução a programação.

O conjunto de códigos submetidos pelos discentes representa um total de 438 (quatrocentos e trinta e oito) soluções, sendo 131 para a primeira questão, 118 para a segunda questão, 108 para a terceira e 81 para a quarta questão. As questões possuem níveis de complexidade diferentes, sendo a primeira questão a mais fácil solução e a última questão a mais difícil. Todas as soluções foram desenvolvidas na linguagem de programação *Python* na versão 3.

O conjunto de soluções de referência é composto por 32 programas, de modo que cada questão possui 8 soluções de referências classificadas como *Relational*. Sendo assim, um conjunto de quatro soluções possui similaridade abaixo de 0.5, enquanto o outro conjunto de quatro soluções possui similaridade acima de 0.5.

O conjunto de classificações foram fornecidos por três especialistas, sendo dois destes professores e um tutor. 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 que deveriam classificar.

4.3. Descrição do experimento

Foram realizados três rodadas de execução no experimento, onde nestas foi alterada a quantidade de soluções de referência e a similaridade entre estas soluções. Na primeira rodada, foi utilizada apenas uma solução de referência classificada como *Relational*. Na segunda rodada, foram utilizadas quatro soluções de referência classificadas como *Relational*, onde a média de similaridade entre as soluções foi menor que 0.5. Na terceira rodada, foram utilizadas outras quatro soluções de referência classificadas como *Relational*, onde a média de similaridade entre elas foi maior que 0.5. Cada rodada foi executada em três etapas: execução dos algoritmos de similaridade sobre os códigos submetidos, geração dos valores referente a cada métrica de comparação e uma análise dos dados.

Todas as classificações receberam correspondentes numéricos, no seguinte modo: *Relational*, *Multistructural*, *Unistructural* e *Prestructural* agora são 4, 3, 2 e 1 respectivamente. As soluções foram classificadas da seguinte forma, quando uma solução

foi classificada como *relational* quando a similaridade estava no intervalo (0.75, 1.00], como *multistructural* quando no intervalo (0.50, 0.75], como *unistructural* quando no intervalo (0.25, 0.50] e como *prestructural* quando no intervalo (0.0, 0.25].

5. Resultados e Discussão

Nesta seção serão apresentados os resultados obtidos na pesquisa, além de uma série de discussões. Nas figuras 1, 2, 3 e 4 são exibidos os gráficos *boxplots* com os resultados obtidos para o conjunto de especialistas e os algoritmos utilizados. Em todos os gráficos, foram mantidos os resultados dos especialistas e da classificação obtida usando apenas uma solução de referência.

Foram adotados as seguintes nomenclaturas: *ref1* o uso de apenas uma solução de referência, *ref4* o uso de quatro referências e *ref4_E2* o uso de quatro referências na rodada 2. Desta maneira, por exemplo, *Tree_ref1* se refere ao resultado do algoritmo *Tree* quando apenas uma solução foi utilizada, *Tree_ref4* refere-se ao resultado do algoritmo *Tree* quando quatro soluções foram utilizadas, e *Tree_ref4_E2* refere-se ao resultado do algoritmo *Tree* quando quatro soluções foram utilizadas na rodada 2.

Para a medida de comparação *Kappa*, tomando como base a análise das medianas exibidas nas Figuras 1 e 2. Percebe-se que a concordância entre os especialistas é superior a concordância de todos os algoritmos, em todas as rodadas. Somente o algoritmo *Tree* obteve, na primeira e segunda rodadas (”*Tree_ref1*”e ”*Tree_ref4*”), valores máximos (Kappa = 0.48) e (Kappa = 0.48) superiores à mediana obtida entre especialistas (Kappa = 0.405). Para todos os outros algoritmos os valores máximos obtidos foram inferiores à mediana obtida entre os especialistas.

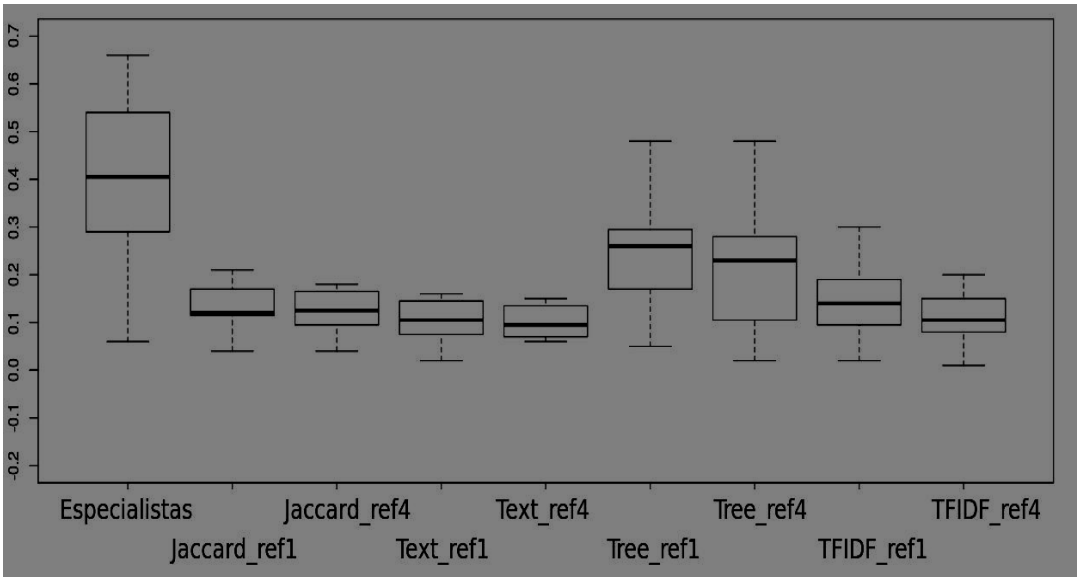

Figura 1. Resultados da primeira rodada para a medida de comparação Cohen's Kappa.

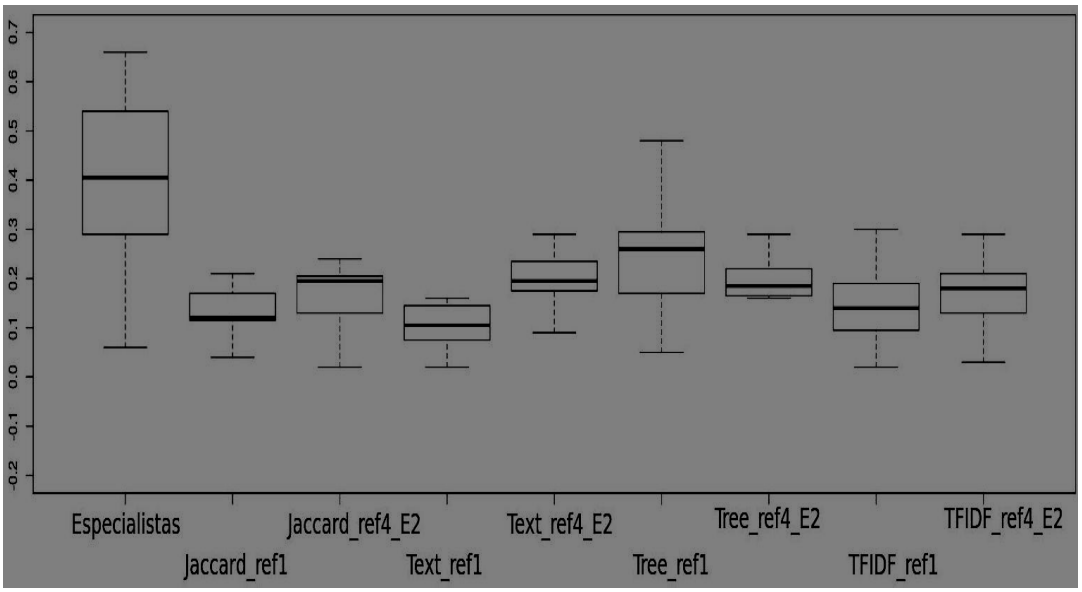

Figura 2. Resultados da segunda rodada para a medida de comparação Cohen's Kappa.

Em todas as execuções do experimento o algoritmo *Tree Edition Distance* foi o que obteve melhores resultados, tanto com uma solução quanto com quatro soluções de referência. Em relação a variação do número de soluções, na segunda rodada, observa-se que o uso de um número maior de soluções resultou em um impacto negativo, muitos dos resultados foram inferiores aos obtidos com uma solução de refeência apenas. Já na terceira rodada, com exceção do algoritmo *Tree*, o impacto do uso de um maior número de soluções foi positivo, pois os valores de máximo, mínimo e mediana, são em geral maiores que os obtidos com uma solução apenas.

Para a medida Distância Euclidiana, tomando como base a análise das medianas exibidas na Figura 3 e 4. Com exceção dos algoritmos *Jaccard* e *Text*, todos os algoritmos tiveram medianas bem próximas à mediana obtida entre os especialistas. Além disso, na terceira rodada, o algoritmo "*Tree_ref4_E2*" (mediana DE = 8.065) foi o único a obter uma mediana superior à mediana entre os especialistas (mediana DE = 8.245). Ao analisar os valores de máximo, mediana e mínimo, percebe-se que o algoritmo *Tree* foi o que obteve melhores resultados no experimento.

Em relação ao número maior de soluções de referências utilizadas, para a segunda rodada, com exceção do algoritmo *Tree*, o impacto foi positivo, pois em geral os valores de máximo, mínimo e mediana obtiveram melhores resultados quando comparados com os algoritmos que utilizaram apenas uma solução de referência. Já na terceira rodada, os resultados foram melhores quando comparados com o uso de uma solução apenas. Desta forma, é possível sugerir que o uso de uma maior quantidade de soluções de referência melhora a classificação fornecida pelos algoritmos.

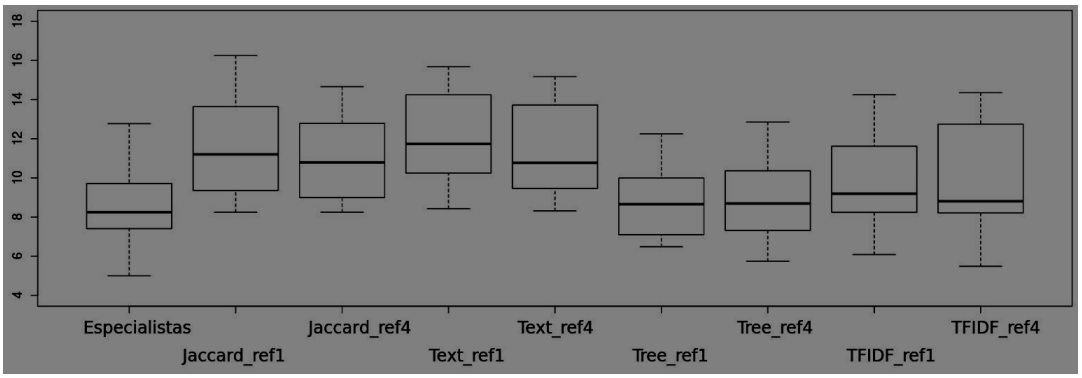

Figura 3. Resultados da primeira rodada para a medida de comparação Distância Euclidiana.

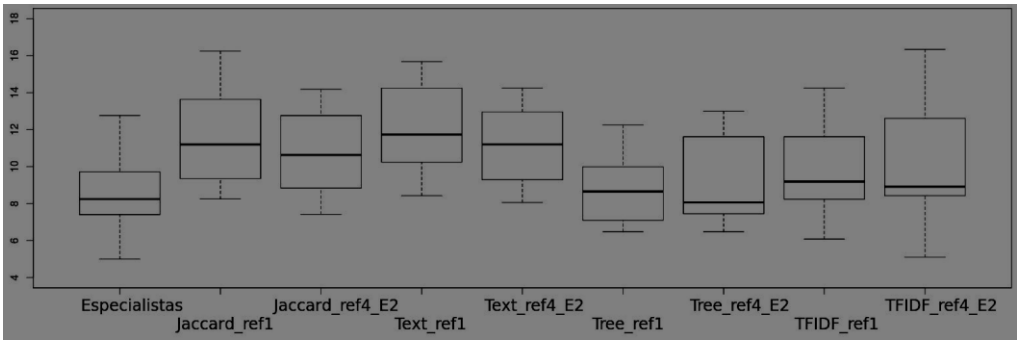

Figura 4. Resultados da segunda rodada para a medida de comparação Distância Euclidiana.

5.1. Análise da convergência das submissões

Nesta seção são apresentados dados relacionados com a convergência das submissões dos alunos em relação ao conjunto de soluções de referência utilizados. Observando os dados é possível verificar como ficaram divididas as soluções submetidas, separadas pelo algoritmo de similaridade adotado e pela solução de referência utilizada. Foram utilizadas 404 soluções do universo de 438, pois algumas soluções possuíam erros de sintaxe ou não haviam sido classificadas por todos os especialistas. Na Tabela 1, são exibidos os dados da análise de convergência, sendo R2, a segunda rodada do experimento, e R3 a terceira rodada do experimento.

Tabela 1. Análise de convergência das soluções.

Algoritmos	Solução A		Solução B		Solução C		Solução D	
	R2	R3	R2	R3	R2	R3	R2	R3
Jaccard	304	99	64	153	2	37	34	115
Text	316	110	65	186	4	38	19	70
Tree	348	124	33	149	11	35	12	96
TFIDF	322	113	62	212	3	35	16	44

Na Tabela 1, na segunda rodada percebe-se uma convergência para a solução A, proposta pelo professor da disciplina, em relação as outras soluções, criadas para realização do experimento. Isso demonstra que a solução de referência A obteve mais concordância com os algoritmos propostos pelos alunos. Na terceira rodada, as soluções propostas tiveram uma melhor divisão entre as soluções de referências utilizadas. A solução B foi a que obteve uma maior concordância entre as soluções utilizadas e os códigos submetidos. A mudança de convergência não gerou alteração significativa em relação à classificação das soluções.

6. Ameaças à validade

As seguintes ameaças à validade se fazem presentes neste trabalho:

- A pequena quantidade de questões – a baixa quantidade de questões utilizadas não permite a generalização dos resultados para outras questões;
- A quantidade de especialistas – apenas três especialistas forneceram classificações, tal quantidade não é representativa do universo de especialistas;
- A linguagem adotada - apenas códigos na linguagem *Python* foram observados, códigos de outra linguagem podem alterar os valores de similaridade obtidos e portanto gerar resultados distintos daqueles aqui exibidos.

7. Conclusões

Neste estudo, foi apresentada uma investigação sobre o uso de algoritmos de avaliação de similaridade com o objetivo de classificar uma solução de acordo com a taxonomia SOLO. Em relação à qualidade da classificação, os resultados obtidos sugerem que o uso destes algoritmos de forma isolada não reflete a classificação de um especialista. Em relação à influência da quantidade de soluções de referência adotadas, os resultados obtidos foram bastante próximos dado qualquer conjunto de soluções de referência.

Referências

Alexandre, Correia, A. L., and de Barros Costa, E. (2015). Um mapeamento sistemático sobre analisadores de código em disciplinas de programação. In Anais do SBIE.

Araujo, E. C., Gaudencio, M., Menezes, A., Ferreira, I., Ribeiro, I., Fagner, A., Ponciano, L., Morais, F., Guerrero, D. S., and Figueiredo, J. A. (2013). O papel do hábito de estudo no desempenho do aluno de programaç ao.

Barbosa, A., Costa, D., Correia, A., Moura, D., and Costa, E. (2016). Uso de algoritmos de similaridade para classificar códigos de acordo com a taxonomia solo em disciplinas de programação introdutória. In Anais dos Workshops do Congresso Brasileiro de Informática na Educação, volume 5, page 1107.

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

Barbosa, L. S., Fernandes, T. C., and Campos, A. M. (2011). Takkou: Uma ferramenta proposta ao ensino de algoritmos. In Anais do CSBC/WEI.

Biggs, J. and Collis, K. F. (1980). Solo taxonomy. volume 17, pages 19–23.

Cristovão, H. M. (2008). Aprendizagem de algoritmos num contexto significativo e motivador: um relato de experiência. In Anais do CSBC/WEI.

Faceli, K., Lorena, A. C., Gama, J., and Carvalho, A. (2011). Inteligência Artificial: Uma Abordagem de Aprendizado de Máquina. Livros Técnicos e Científicos.

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, pages 21–26. ACM.

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

Maciel, D. L., Soares, J. M., França, A. B., and Gomes, D. G. (2012). Análise de similaridade de códigos-fonte como estratégia para o acompanhamento de atividades de laboratório de programação. volume 10.

Mason, R. and Cooper, G. (2014). Introductory programming courses in australia and new zealand in 2013 - trends and reasons. In Proc. of the Sixteenth Australasian Computing Education Conference - Volume 148, ACE, pages 139–147, Darlinghurst, Australia.

Moreira, M. P. and Favero, E. L. (2009). Um ambiente para ensino de programação com feedback automático de exercícios. In Workshop sobre Educação em Computação (WEI 2009), volume 17.

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.

Paes, R. B., Malaquias, R., Guimaraes, M., and Almeida, H. (2013). Ferramenta para a avaliação de aprendizado de alunos em programação de computadores. In Anais do II Congresso Brasileiro de Informática na Educação (CBIE 2013) Workshops (WCBIE 2013), Campinas, SP.

Pelz, F. D., de Jesus, E. A., and Raabe, A. L. (2012). Um mecanismo para correção automática de exercícios práticos de programação introdutória. In Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE), volume 23.

Ramos, V., Freitas, M., Galimbert, M., Mariani, A. C., and Wazlawick, R. (2015). A comparação da realidade mundial do ensino de programação para iniciantes com a realidade nacional: Revisão sistemática da literatura em eventos brasileiros. In Anais do SBIE.

SBC (2005). Currículo de referência da sociedade brasileira de computação para cursos de graduação em computação e informática.