Uma abordagem adaptativa para gerar agrupamento de códigos em disciplinas de programação introdutória

Alexandre de A. Barbosa [1,3], **Evandro de B. Costa** [2,3], **Patrick H. Brito** [1,2]

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

[2]Instituto de Computação - Universidade Federal de Alagoas (UFAL)
Maceió - AL - Brasil

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

alexandre.barbosa@arapiraca.ufal.br, ebc.academico@gmail.com

patrick@ic.ufal.br

Abstract. *Despite the importance of introductory programming disciplines, it is quite common to find problems. In such environments, we easily find unmotivated students with some doubts and that do not understand basic concepts. Monitoring each one of the students is not trivial because there are many students and it is necessary to observe many characteristics of each code submited for practical activities. In this research, clustering of codes to minimize the effort of evaluation is investigated. The results vary from reasonable to perfect concordances, considering the semiautomatic evaluations obtained with the clustering and the expert evaluations.*

Resumo. *Apesar da importância das disciplinas introdutórias de programação, é bastante comum encontrar problemas de ensino e aprendizagem. Facilmente, em tais ambientes, encontramos alunos desmotivados, com dúvidas não esclarecidas e que não entendem conceitos básicos. O acompanhamento individualizado dos alunos não é trivial, pois existem muitos alunos e é necessário observar muitas características de cada código proposto nas diversas atividades práticas adotadas. Nesta pesquisa, investigou-se o agrupamento de soluções, visando minimizar o esforço de avaliação despendido. As concordâncias obtidas nos resultados variam de razoável a perfeita, considerando as avaliações semiautomáticas obtidas com os agrupamentos e as avaliações de especialistas.*

1. Introdução

Os períodos iniciais dos cursos da área de computação, em geral, englobam diferentes disciplinas que possuem como foco o estudo de algoritmos e a implementação de programas. Facilmente encontramos nestas disciplinas alunos desmotivados, com dúvidas não esclarecidas e que não conseguem entender os conceitos necessários para a prática de programação. Além disso, muitos dos alunos aprovados não possuem as competências necessárias ao curso e à vida profissional [McCracken et al. 2001].

DOI: 10.5753/cbie.sbie.2017.1427

No contexto de ensino e aprendizagem de programação, tipicamente são adotadas atividades práticas de codificação. A análise das soluções propostas é bastante trabalhosa. Para avaliar um código, além de identificar se as saídas corretas são geradas para um conjunto de entrada, diversos parâmetros podem ser observados, tais como, estrutura do código, eficiência, manutenibilidade, documentação (ex. comentários), outros. Sendo a mencionada avaliação demorada, ela é sujeita ao viés e aos erros de cada avaliador, caracterizando então um problema, pois, o *feedback* rápido e eficiente é de extrema importância para possibilitar o aprendizado de qualquer conceito [Stegeman et al. 2014].

O aluno, sem o *feedback* em tempo adequado, desconhece se seu entendimento é correto. Dessa forma, possíveis dúvidas não são apresentadas e novos conteúdos passam a ser ministrados pelo professor sem que os anteriores tenham sido totalmente compreendidos. É comum que o professor, mesmo quando auxiliado por monitores, não consiga realizar avaliações adequadas de forma rápida, pois caso busque fornecer uma avaliação rápida, provavelmente ela carecerá de qualidade, por outro lado, ao tentar prover *feedback* de alta qualidade, ocorrerá uma sobrecarga de trabalho e demora na resposta.

Dado o contexto anterior, é possível afirmar que o acompanhamento individualizado é inviável no cenário descrito. Alguns pesquisadores apontam que a dificuldade do professor em fornecer um acompanhamento mais próximo aos alunos, agrava e potencializa a ocorrência de diversos fatores causadores de dificuldades de ensino e aprendizagem de programação [Mota et al. 2009] [Stegeman et al. 2014].

Ao longo da avaliação dos códigos propostos como solução para um problema, o professor observa soluções muito próximas. Contudo, é improdutivo buscar estas soluções similares manualmente. Nesta pesquisa, para atacar parte do problema descrito, investigou-se uma de suas facetas, concernente à possibilidade de agrupar soluções em código, utilizando características extraídas das próprias implementações. Como principal contribuição desta pesquisa, é esperado que o agrupamento de soluções permita ao professor agilizar o processo de avaliação e fornecer *feedback* mais detalhado para os discentes.

O restante deste artigo está organizado da seguinte forma, na Seção 2 são apresentados os conceitos necessários à compreensão da pesquisa; Na Seção 3 são descritos os trabalhos relacionados ao tema pesquisado, buscando-se caracterizar onde este trabalho se insere, bem como descrever quais os principais diferenciais abordados nesta pesquisa em relação à outras propostas; Na Seção 4 são exibidas a proposta e os detalhes da condução das investigações realizadas até o momento; Finalmente, na Seção 5 são apresentadas as conclusões originadas do estado atual desta pesquisa.

2. Fundamentação

O conjunto de conceitos necessários para compreensão da pesquisa se constitui do algoritmo de agrupamento *Kmeans* [Piech 2013], um conjunto de métricas de software [Sommerville 2010] e um conjunto de medidas estatísticas de comparação.

Algoritmos de agrupamento organizam grupos de elementos levando em conta sua similaridade. Desta forma, é correto afirmar que os elementos de um mesmo grupo são mais similares entre si do que em relação aos elementos de outros grupos. O algoritmo de agrupamento *Kmeans* foi utilizado nesta pesquisa, pois é possível determinar a quantidade 'K' de grupos que devem ser criados fornecendo um parâmetro ao algoritmo. O

agrupamento realizado por *Kmeans* se baseia em um calculo de distância, tipicamente a distância euclidiana, que toma como base um vetor de propriedades. Nesta pesquisa, diferentes vetores de propriedades foram adotados, cada vetor possuía um subconjunto de métricas de software extraídas dos códigos.

Uma métrica de software representa um meio de verificar se um software possui determinada propriedade. As seguintes métricas foram adotadas na condução deste trabalho: corretude funcional [Sommerville 2010], complexidade ciclomática [McCabe 1996], a quantidade de operadores distintos [Halstead 1977], a quantidade de operandos distintos [Halstead 1977], similaridade baseada no coeficiente de Jaccard [Rego et al. 2014], similaridade baseada na distância de edição de texto [Rego et al. 2014] e similaridade baseada na distância de edição de árvore [Rego et al. 2014].

As medidas de comparação Kappa de Cohen (*Cohen's Kappa*) [Cohen 1960] [Gwet 2008] e distância euclidiana foram adotadas para analisar o quão próximas, ou distantes, são dois conjuntos de avaliações/classificações. Sejam dados dois conjuntos de classificações, a intensidade da concordância entre estes pode ser dada pela medida do coeficiente Kappa de Cohen. Os valores de Kappa podem variar no intervalo [-1, 1], onde valores negativos sugerem discordância, 0 (zero) indica concordância dada ao acaso, e valores positivos indicam a intensidade de concordância, com as seguintes interpretações textuais, concordância leve, razoável, moderada, substancial, quase perfeita e perfeita ($kappa = 1$). A distância euclidiana representa a distância entre dois pontos em um espaço n-dimensional. Desta forma, sejam dados dois vetores cujas coordenadas correspondam ao conjunto de avaliações/classificações dos avaliadores, o valor obtido no calculo da distância euclidiana representará o quão distantes são suas avaliações.

3. Trabalhos relacionados

Trabalhos relacionados ao contexto de geração automática de avaliações, *automatic grading* e *automatic feedback*, não são recentes. O trabalho de Hext e Winings [Hext and Winings 1969], além do trabalho de Forsythe e Wirth [Forsythe and Wirth 1965], por exemplo, datam da década de sessenta. Muitas das pesquisas recentes continuam explorando ideias similares aos trabalhos de Hext e Winings.

A grande maioria dos avaliadores de código, dentre os quais citamos [Paes et al. 2013, Yulianto and Liem 2014], utilizam juízes online, ou seja a avaliação é baseada em testes de aceitação. Algumas destas propostas complementam a indicação de sucesso, ou falha, fornecida pelos testes com dicas que auxiliem o aluno a identificar os erros. Tais dicas são associadas a um caso de teste com base na experiência do testador em determinar a provável causa da falha.

Diversas pesquisas, como exemplo citamos [Rego et al. 2014, Biggers and Kraft 2011, Li et al. 2016], utilizam a análise de similaridades com outras finalidades que não sejam a detecção de plágio. Os trabalhos de Li [Li et al. 2016] e Biggers [Biggers and Kraft 2011] exploram a identificação de similaridades sob uma ótica mais próxima do ensino de Engenharia de Software, o intuito é, por exemplo, identificar a aderência do código ao que foi especificado como requisito. O trabalho de Rego [Rego et al. 2014], investiga se avaliadores automáticos (algoritmos de avaliação de similaridade) podem comparar códigos de alunos tão bem quanto especialistas.

As propostas descritas em [Raabe et al. 2015, Pelz et al. 2012] exploram meios

para prover *feedback* automático aos alunos, cada uma explorando diferente de técnicas combinando análise estática e dinâmica. Em [Raabe et al. 2015], é apresentado um gerador de dicas sobre erros cometidos, e em [Pelz et al. 2012] um análisador de códigos com verificação da presença de comandos considerados como obrigatórios nas soluções propostas.

Outras pesquisas, dentre os quais citamos [Silva et al. 2014, Srikant and Aggarwal 2014, Choudhury et al. 2016, Yin et al. 2015], buscam agrupar ou classificar soluções de código. Em cada uma destas pesquisas é adotadado um conjunto diferente de técnicas. No trabalho de Silva [Silva et al. 2014], é proposto um arcabouço apaz de combinar diferentes tipos de analisadores de código, onde pode-se observar corretude sintática, corretude semântica e a presença (ou ausência) de variáveis e comandos de entrada e saída, operadores e expressões, estruturas de seleção e estruturas de repetição. Nos trabalhos de Choudhury e Yin [Choudhury et al. 2016, Yin et al. 2015] são utilizados o algoritmo de agrupamento *OPTICS*, a métrica ABC [Fitzpatrick 2000] e similaridade baseada em distância de árvores, com o intuito de auxiliar, através de dicas, estudantes matriculados em cursos de Engenharia de Software a produzir código com maior 'qualidade'. No trabalho de Srikant, em [Srikant and Aggarwal 2014], é proposto um meio de aplicar aprendizagem de máquina (regressão linear, *random forests* e *Support Vector Machines - SVMs*) para aprender quais propriedades que são levadas em consideração na avaliação de um exercício de programação.

A principal contribuição existente na pesquisa descrita neste artigo é minimizar o esforço de avaliação de forma adaptada aos critérios de cada especialista. Outros trabalhos similares utilizam critérios fixos, ou seja, parte-se do suposto que todos os avaliadores utilizam a mesma forma de avaliação. Nenhum dos outros trabalhos relacionados encontrados utilizam técnicas de aprendizagem de máquina para fornecer uma avaliação adaptável, em relação ao professor/especialista que fornece esta avaliação.

4. Agrupamento de códigos em disciplinas de programação introdutória

A cada especificação de exercícios em disciplinas de programação o professor se depara com dois objetivos, prover aos alunos um conjunto de exercícios úteis ao aprendizado e ponderar o custo e esforço relacionados com avaliação dos códigos que serão submetidos. O professor precisa selecionar um conjunto de exercícios que torne possível realizar uma avaliação adequada em um curto espaço de tempo, pois, como descrito anteriormente o *feedback* rápido é um importante fator para o aprendizado.

Compreendendo avaliação como o processo de fornecer uma classificação, tipicamente um critério (ex. A, B, C, D) ou nota (ex. valores entre 0 e 10) para a solução apresentada, e um *feedback*, que pode variar de professor para professor, correspondendo, por exemplo, a um conjunto de observações para melhoria do código ou às justificativas para o critério ou nota fornecido. A condução da investigação deste trabalho, focou-se apenas no aspecto de classificação de uma solução, sendo esta classificação representada por uma nota no intervalo [0;10]. Sendo então um dos objetivos, identificar se a avaliação semiautomática obtida com o uso de agrupamento de códigos é semelhante à avaliação de um especialista.

Uma das estratégias utilizadas para facilitar e agilizar as atividades de avaliação de código é a divisão de trabalho. O trabalho de avaliação é dividido entre o professor e

um grupo de monitores, ou quando possível, entre todos os professores que ministram um conjunto de disciplinas relacionadas. Visto isso, outro objetivo consiste em averiguar se a concordância das avaliações obtidas com agrupamento e as avaliações completas de um especialista possuem alta concordância em comparação às avaliações de dois especialistas humanos.

4.1. Proposta de abordagem adaptativa para agrupamento de códigos

O agrupamento adaptativo de códigos proposto nesta pesquisa, é realizado através das seguintes etapas: (1) Extração das métricas sobre os códigos; (2) Identificação dos critérios adotados pelo especialista; (3) Geração de agrupamentos; (4) Especificação de avaliações.

A extração das métricas sobre os códigos corresponde à computação dos valores inseridos no vetor de propriedades utilizado pelo algoritmo de agrupamento. Para as métricas de similaridade, apenas o valor de similaridade em relação a uma solução de referência foi utilizado.

Para identificar os critérios adotados por um especialista, foi utilizada uma abordagem exaustiva, ou seja, sem técnicas de Inteligência Artificial. Sendo assim, todas as combinações possíveis para o conjunto de métricas adotado foram geradas. Uma vez que foram utilizadas sete métricas, tem-se $2^7 - 1 = 127$ diferentes conjuntos de métricas, ou 127 diferentes vetores de propriedades. Sendo este conjunto de métricas representado por $C = (c_1, c_2, ..., c_{127})$. O elemento c_i deste conjunto é selecionado com base no grau de concordância com o especialista em relação as avaliações fornecidas por este.

A combinação de todas as métricas visava capturar, com algum grau de aproximação, os critérios adotados pelos especialistas. Como exemplo, se um especialista observa apenas saídas corretas para as entradas fornecidas, seu critério seria capturado por um elemento c_i do conjunto C, onde a única métrica adotada seria a corretude funcional. Por outro lado, outro especialista que observe saídas corretas para as entradas fornecidas e outro conjunto de características do código teria seu critério capturado por um elemento c_j, onde a métrica de corretude funcional estaria combinada com outras métricas.

A geração de agrupamentos para os códigos foi realizada fazendo-se uso do algoritmo *Kmeans*, utilizando os valores $k = 5$ e $k = 10$. Justifica-se a escolha destes valores pois, os conjuntos de códigos avaliados possuíam um mínimo de 23 submissões, e sendo assim no pior caso o esforço de avaliação seria reduzido aproximadamente pela metade.

Em todas as execuções foram usados parâmetros de entrada *default* para o algoritmo *Kmeans*. Para cada possível vetor de propriedades gerado na etapa anterior, foi criado um agrupamento para cada valor de k. Sendo então, o conjunto de agrupamentos representado pelo conjunto $G = (g_1, g_i, g_j, g_{254})$. Onde os elementos g_i, para $1 \leq i \leq 127$, correspondem ao uso de $k = 5$ e as diferentes combinações de métricas e os elementos g_j, para $128 \leq i \leq 254$, correspondem ao uso de $k = 10$ e as diferentes combinações de métricas.

A especificação de avaliações, ocorreu de modo que cada especialista atribuísse uma nota (classificação) a um elemento representativo do grupo. Sendo este elemento aquele que possui a menor distância da solução de referência geral. Essa nota é então generalizada para todos os elementos de um mesmo grupo. Desta forma, quando *kmeans* foi executado fornecendo $k = 5$ foram necessárias apenas cinco avaliações do especialista,

do mesmo modo, para $k = 10$ foram necessárias dez avaliações do especialista. Todas as outras avaliações eram generalizadas para todos os elementos do grupo.

4.2. Metodologia de avaliação da abordagem

4.2.1. Descrição dos dados

O conjunto de dados utilizado para a pesquisa é formado por um conjunto de problemas (exercícios) de programação, um conjunto de códigos submetidos por discentes como soluções aos exercícios, e um conjunto de avaliações fornecido por especialistas em programação. O conjunto de exercícios é formado de seis problemas, sendo estes: 'salário com bônus' (32 submissões de uma turma da UFAL); 'distância entre 2 pontos' (23 submissões de uma turma da UFAL); 'aprovado' (43 submissões de uma turma da UFAL); 'eleitor (40 submissões de uma turma da UFPB); '*loop* de ímpares' (41 submissões de uma turma do IFAL); e 'divisível por 3' (44 submissões de uma turma da UFAL). O conjunto de códigos submetidos pelos discentes consiste em um total de 223 submissões de soluções propostas, ou de tentativas de soluções. Apenas códigos na linguagem *Python* foram utilizados na pesquisa. A Figura 1 ilustra a apresentação de um dos exercícios utilizados, o problema 'Aprovado'.

Figura 1. Tela de descrição de um problema para um especialista e listagem de avaliações realizadas.

O conjunto de avaliações foi fornecido por 8 especialistas, sendo 2 professores de uma instituição de ensino superior, 3 graduados que foram monitores de disciplinas

de programação e 3 alunos de graduação que concluíram as disciplinas introdutórias de programação. Cada especialista realizou os procedimentos de avaliação no tempo e ambiente em que julgou mais adequado. Ao realizar uma avaliação era necessário fornecer: (a) Uma descrição dos critérios adotados (exibido na Figura 1); (b) Uma nota, no intervalo $[0; 10]$, (exibido na Figura 1); (c) Um *feedback* correspondendo ao que seria informado ao autor do código (exibido na Figura 1); e (d) Uma observação, fornecida ao pesquisador, relatando dúvidas ou problemas relacionados com a avaliação de um código.

Observando as descrições dos critérios adotados foi possível identificar diferentes especialistas adotam diferentes conjuntos de critérios na avaliação de um mesmo problema. Nesta pesquisa os critérios de um especialista eram capturados por um conjunto de métricas de software. Observando os conjuntos de métricas identificados, notou-se que um mesmo especialista varia suas observações a depender do problema que está sendo avaliado. Postas estas afirmações, podem ser derivadas duas importantes observações. A primeira observação consiste no fato de que não há garantias de que durante o processo de avaliação um especialista não tenha alterado seu conjunto de critérios, seja com a adoção de outro(s) critério(s) além daqueles informados, ou ainda que tenha deixado de utilizar algum dos critérios informados. A segunda observação se dá na variação dos critérios adotados por um especialista na avaliação de soluções fornecidas para um mesmo problema, ou seja, quando o especialista avalia de maneira diferente submissões muito similares, ou até mesmo iguais. A abordagem proposta neste trabalho pode capturar critérios 'implícitos' ao especialista. Contudo, a segunda situação, quando para um especialista ocorre a variação dos critérios de avaliação em um mesmo problema, levaria a obtenção de uma menor concordância da abordagem com o especialista.

4.2.2. Avaliação da abordagem

Para avaliar a abordagem proposta, foram comparados dois vetores para cada problema e para cada especialista, sendo um vetor de notas gerado de maneira semiautomática através dos agrupamentos e o outro correspondendo ao conjunto completo de notas do especialista. Além disso, os vetores de notas de dois especialistas, par a par, também foram comprados. Tais comparações fizeram uso das medidas de comparação descritas anteriormente, Kappa de Cohen e distância euclidiana.

A medida Kappa de Cohen indicava o grau de concordância entre as notas fornecidas. Nesta medida uma concordância só ocorre quando exatamente a mesma nota é fornecida nos dois vetores. Notas próximas, mas que não são idênticas, não são computadas como uma concordância. O uso da distância euclidiana é então justificado, pois com essa medida é possível observar o quanto os vetores de notas, interpretados como pontos em um espaço n-dimensional, são distantes.

4.2.3. Resultados e discussão

Os resultados obtidos foram computados com base nos valores máximos, mínimos e as médias para as medidas comparativas adotadas de forma separada por problema em relação a cada especialista. Os melhores resultados foram obtidos quando o algoritmo *Kmeans* foi executado com 'K=10'. Os resultados gerais obtidos são apresentados na

Tabela 1.

Kappa de Cohen			**Dist. Euclidiana**		
Máx.	Min.	Méd.	Máx.	Min.	Méd.
1.00	0.39	0.76	19.57	0.00	5.95

Tabela 1. Resultados gerais obtidos para os agrupamentos de códigos

Sendo assim, com o intuito de identificar se a avaliação semiautomática obtida com o uso de agrupamento de códigos é semelhante a avaliação de um especialista, considerando a média, obteve-se uma concordância substancial ($kappa = 0.76$) entre a avaliação baseada no agrupamento e a avaliação de especialistas. Este resultado é fortalecido pelos valores obtidos com relação à distância euclidiana, que teve um valor médio de 5.95, o que indica uma proximidade entre as avaliações. Assim, dada as concordâncias obtidas, é possível sugerir que a avaliação baseada nos agrupamentos fornece uma avaliação semelhante a dos especialistas.

O gráfico exibido na Figura 2 contém a representação em *boxplots* das concordâncias obtidas com a abordagem de agrupamento em relação aos especialistas e das concordâncias gerais para cada especialista em relação aos seus pares considerando todas as avaliações realizadas.

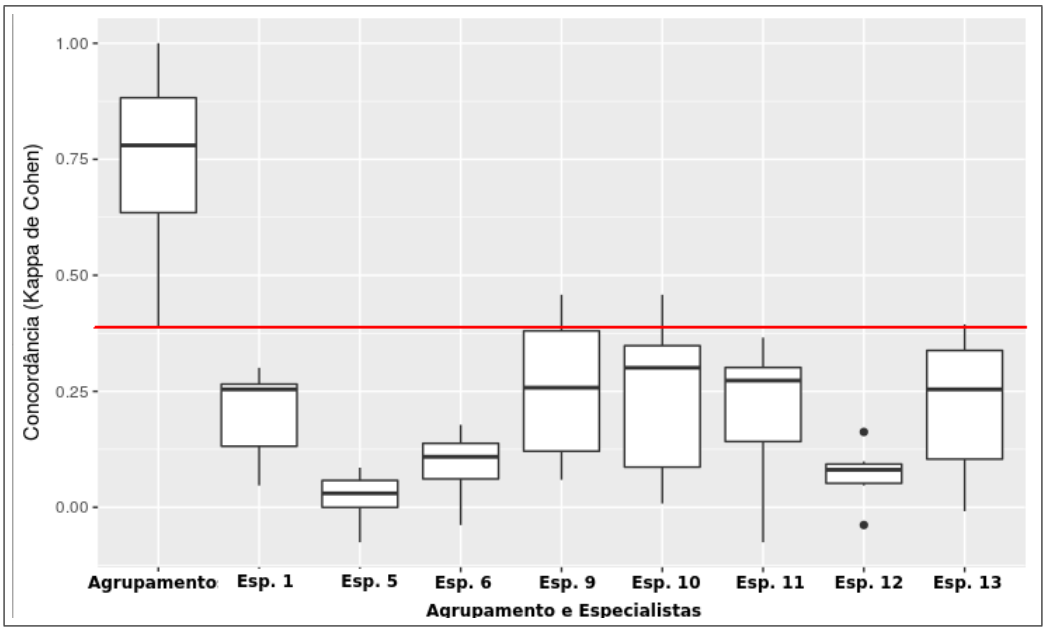

Figura 2. Concordâncias obtidas com a abordagem de agrupamento e de cada especialista em relação aos seus pares. Especialistas representados por um *id*.

Observando o gráfico de concordâncias, é possível notar que o valor máximo de concordância obtido entre especialistas (concordância moderada, $kappa = 0.45$) é bastante próximo ao valor mínimo obtido, enfatizado pela linha horizontal que corta o gráfico, em relação a avaliação baseada nos agrupamentos (concordância razoável, $kappa = 0.39$), sendo ligeiramente superior. Considerando as médias, ou seja o quanto um especialista concorda com seus pares e o quanto a abordagem de agrupamento gera

avaliações com concordância em relação ao especialista, a concordância entre especialistas (concordância razoável, $kappa = 0.2539$) foi bastante inferior à concordância obtida com a abordagem de agrupamento (concordância substancial, $kappa = 0.70$). Sendo assim, pode ser sugerido que a avaliação baseada nos agrupamentos fornece uma concordância superior àquela obtida entre dois especialistas.

5. Conclusões e trabalhos futuros

Neste artigo foi proposto o uso de algoritmos de agrupamento como meio para minimizar o esforço despendido na avaliação de códigos em disciplinas introdutórias de programação. Os resultados obtidos até o momento sugerem que é possível minimizar o esforço de avaliação despendido. Duas observações apoiam essa afirmação. Primeiramente, os critérios de correção de um especialista parecem ser capturados por métricas da Engenharia de Software com um bom grau de corretude. Além disso, as avaliações de soluções similares se motraram bastante próximas para os cenários observados.

Muito trabalho ainda é necessário na condução desta pesquisa, comparar o uso de diferentes técnicas de agrupamento, utilizar outras medidas representativas dos códigos, ampliar o universo de dados, verificar a minimização do esforço de avaliação em relação ao *feedback* para o aluno, entre outras atividades. Dado o sucesso do uso de outras técnicas em trabalhos com finalidades similares, é esperado que tais mudanças e adições contribuam com a melhoria da abordagem proposta.

Referências

Biggers, L. R. and Kraft, N. A. (2011). Quantifying the similiarities between source code lexicons. In *Proceedings of the 49th Annual Southeast Regional Conference*, ACM-SE '11, pages 80–85, New York, NY, USA. ACM.

Choudhury, R. R., Yin, H., Moghadam, J., Chen, A., and Fox, A. (2016). Autostyle: Scale-driven hint generation for coding style. In *Proceedings of the 13th International Conference on Intelligent Tutoring Systems*, ITS 201, pages 122–132.

Cohen, J. (1960). A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20(1):37–46.

Fitzpatrick, J. (2000). More c++ gems. chapter Applying the ABC Metric to C, C++, and Java, pages 245–264. Cambridge University Press, New York, NY, USA.

Forsythe, G. E. and Wirth, N. (1965). Automatic grading programs. *Commun. ACM*, 8(5):275–278.

Gwet, K. L. (2008). Computing inter-rater reliability and its variance in the presence of high agreement. *British Journal of Mathematical and Statistical Psychology*, 61(1):29–48.

Halstead, M. H. (1977). *Elements of Software Science (Operating and Programming Systems Series)*. Elsevier Science Inc., New York, NY, USA.

Hext, J. B. and Winings, J. W. (1969). An automatic grading scheme for simple programming exercises. *Commun. ACM*, 12(5):272–275.

Li, S., Xiao, X., Bassett, B., Xie, T., and Tillmann, N. (2016). Measuring code behavioral similarity for programming and software engineering education. In *Proceedings of the*

38th International Conference on Software Engineering Companion, ICSE '16, pages 501–510, New York, NY, USA. ACM.

McCabe, T. J. (1996). Cyclomatic complexity and the year 2000. *IEEE Software*, 13(3):115–117.

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y. B.-D., Laxer, C., Thomas, L., Utting, I., and Wilusz, T. (2001). A multi-national, multi-institutional study of assessment of programming skills of first-year cs students. In *Working Group Reports from ITiCSE*, pages 125–180, New York, USA.

Mota, M. P., de Brito, S. R., Moreira, M. P., and Favero, E. L. (2009). Ambiente integrado a plataforma moodle para apoio ao desenvolvimento das habilidades iniciais de programação. In *Anais do XX SBIE 2009*, Florianópolis, SC, Brasil.

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 CBIE - 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 *Anais do SBIE*.

Piech, C. (2013). Kmeans. `http://stanford.edu/~cpiech/cs221/handouts/kmeans.html`. Último acesso em fevereiro de 2017.

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*, pages 358–366.

Rego, M. G., Dantas, A., and Guerrero, D. S. (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.

Silva, M. T., Costa, E. D. B., Barbosa, P. H., and Cavalcante, J. D. C. (2014). Um Arcabouço para Construção de Mecanismos de Análise de Códigos de Programação Introdutória. In *Anais do SBIE*, pages 1083–1092.

Sommerville, I. (2010). *Software Engineering*. Addison-Wesley Publishing Company, USA, 9th edition.

Srikant, S. and Aggarwal, V. (2014). A system to grade computer programming skills using machine learning. In *Proc. of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 1887–1896. ACM.

Stegeman, M., Barendsen, E., and Smetsers, S. (2014). Towards an empirically validated model for assessment of code quality. In *Proc. of the 14th Koli Calling Int. Conf. on Computing Education Research*, Koli Calling, pages 99–108, New York, USA.

Yin, H., Moghadam, J., and Fox, A. (2015). Clustering student programming assignments to multiply instructor leverage. In *Proceedings of the Second (2015) ACM Conference on Learning @ Scale*, L@S '15, pages 367–372. ACM.

Yulianto, S. V. and Liem, I. (2014). Automatic grader for programming assignment using source code analyzer. In *Data and Software Engineering (ICODSE), 2014 International Conference on*, pages 1–4. IEEE.