Avaliação de atividades de programação submetidas em MOOC com emprego de técnicas de visualização

Emerson Yudi Nakashima[1], Wagner Aparecido Monteverde[2], Narci Nogueira da Silva[1,2], Aretha Barbosa Alencar[1], Marco Aurélio Graciotto Silva[1,2]

[1] Universidade Tecnológica Federal do Paraná (UTFPR)
Departamento Acadêmico da Computação (DACOM)
Campo Mourão – PR – Brasil

[2]Universidade Tecnológica Federal do Paraná
Programa de Pós-Graduação em Informática
Cornélio Procópio – PR – Brasil
{emersonnakashima, wagnermonteverde}@alunos.utfpr.edu.br
{narci, arethaalencar, magsilva}@utfpr.edu.br

Resumo. *MOOC (Massive Open Online Course) é um tipo de plataforma de e-learning em ampla escala, apresentando uma quantidade elevada de atividades de aprendizagem a serem realizadas e avaliadas. Observando-se que muitas submissões são semelhantes, pode-se explorar as variações de implementações por meio de agrupamentos, diminuindo o tempo gasto em correção sem prejuízo à qualidade. O objetivo deste trabalho é propor subsídios para avaliação de programas submetidos em disciplinas introdutórias à computação, utilizando técnicas de mineração e visualização de dados para apresentar agrupamentos de submissões semelhantes. Os subsídios consistem em ferramentas para extração de características e para mineração visual de dados. Estes subsídios foram utilizados para a avaliação de atividades de programação de um curso introdutório à Computação. Os resultados sugerem de que o uso de técnicas de visualização pode ser benéfico ao processo de avaliação de programas.*

Abstract. *MOOC (Massive Open Online Course) is a type of large scale e-learning platform, presenting a consirable amount of learning activities to be carried out and evaluated. Noting that many submissions are similar, one can explore the variations of implementations through clustering, reducing the time spent for evaluation without impairing quality. The objective of this work is to propose subsidies for the evaluation of programming assignments submitted in introductory programming courses, using data mining and visualization techniques to present groups of similar submissions. The subsidies consist of tools for extraction of characteristics, and for visual data mining. These subsidies were used in an study to evaluate programming activities of an introductory programming course. The results suggest that the use of visualization techniques may be beneficial to the process of programming assignments evaluation.*

1. Introdução

Massive Open Online Courses (MOOC) é uma plataforma de ensino online com cursos massivos e abertos, que visa oferecer os mais diversos cursos a nível global, geralmente gratuitamente e oferecendo aos usuários a liberdade de planejar os horários em

DOI: 10.5753/cbie.wcbie.2017.952

que realizarão as aulas. O caráter massivo refere-se a capacidade do MOOC em suportar uma grande quantidade de alunos [Klobas et al. 2014]. Tal quantidade é bem superior ao número de discentes que uma sala tradicional pode acomodar ou o total de participantes em um curso online antes do surgimento do MOOC. Por exemplo, em um curso de Inteligência Artificial ofertado, online e gratuitamente, pela Universidade de Stanford em 2011, possuía 160.000 estudantes inscritos [Rodriguez 2012].

Principalmente em MOOCs de ensino de programação, tratando-se de uma ferramenta de ensino a nível mundial, deve-se considerar um grande número de usuários. Com isso, torna-se necessária a utilização de mecanismos de avaliação automática ou semiautomática [Schmidt e McCormick 2013]. No entanto, as atividades de programação necessitam que seus algoritmos sejam analisados quanto às saídas geradas pela sua execução, projeto do algoritmo, facilidade de compreensão, dentro outros requisitos, a fim de retornar o resultado da avaliação. Como há uma grande quantidade de submissões em cursos de programação, consequentemente acarreta alguns problemas como avaliar todas as submissões, tempo requerido e aumento de custo de correção por parte do professor.

Considerando que muitas submissões podem ser semelhantes, os professores podem explorar e compreender as variações de implementações enviados pelos estudantes [Yin et al. 2015], a fim de diminuir o tempo gasto para correção dos códigos-fontes submetidos, por meio do uso de técnicas de visualização e mineração de dados.

Para favorecer o acesso à informação de qualquer conjunto de dados, é preciso ampliar a capacidade cognitiva do ser humano em processos de exploração de dados massivos. Esta é justamente uma das propostas da Visualização de Informação [Chen 2006], disciplina que surgiu a partir da disponibilidade de interfaces gráficas mais eficientes com recursos de interação adequados. Dado um conjunto de dados ou padrões extraídos previamente, a visualização de dados produz modelos gráficos e representações visuais a fim de utilizar a capacidade cognitiva do ser humano, por meio da percepção visual, para colaborar com a exploração e obtenção de informações úteis presente nos dados [Oliveira e Levkowitz 2003]. Ademais, a visualização de dados é intuitiva, permitindo explorar os dados mais rapidamente e fornecendo melhores resultados na maioria das vezes.

Técnicas de projeção multidimensional têm sido empregadas para gerar representações visuais globais de conjuntos de dados massivos e de alta dimensionalidade que podem ser embutidos em espaços métricos, ou para os quais uma matriz de distâncias entre as instâncias pode ser calculada [Paulovich et al. 2008]. Essas técnicas mapeiam dados de alta dimensionalidade em um espaço de baixa dimensionalidade, tipicamente 2D, buscando preservar, na medida do possível, as relações de (dis)similaridade entre os dados. É possível calcular a similaridade entre códigos-fontes. Primeiramente, realiza-se a extração de características desses códigos compreendendo as variações de implementação por meio da análise estática [Yin et al. 2015,Glassman et al. 2014,Taherkhani et al. 2012] ou análise dinâmica [Glassman et al. 2015]. Posteriormente, calcula-se a similaridade entre os códigos com base nas características extraídas. Este processo permite a aplicação técnicas de projeção multidimensional a conjuntos de códigos-fontes, facilitando o aspecto cognitivo na visualização das relações de similaridade entre os códigos.

Devido a favorecerem a percepção de similaridade entre os códigos-fontes, esses mapas gerados por projeções multidimensionais permitem identificar grupos de códigos-

fontes altamente relacionados (similares) e fronteiras entre grupos. Com isso, o agrupamento das submissões semelhantes pode auxiliar na correção ao permitir a correção de uma amostra do agrupamento, dado que todos os outros códigos desse são similares quanto às características consideradas na análise, permitindo que o professor dedique o tempo poupado na correção de diversas implementações para aprimorar as correções de poucos trabalhos, além de possibilitar menor custo para o MOOC.

O objetivo deste trabalho foi estabelecer subsídios para avaliar a visualização para avaliação de submissões em disciplinas introdutórias à computação, utilizando técnicas de mineração e visualização de dados para construir e apresentar agrupamentos de código-fonte semelhantes. Como metas, estabeleceu-se o desenvolvimento de uma ferramenta para recuperação de dados a partir de códigos-fontes e da alteração da ferramenta para mineração e visualização de dados ScienceView [Alencar et al. 2012]. Para extrair as características do estilo de escrita e a complexidade ciclomática dos dados, foi utilizada uma versão alterada da ferramenta `flake8` [Cordasco et al. 2010] para análise o código das atividades de programação.

Por meio desses subsídios obtivemos indícios que a visualização pode auxiliar o professor a corrigir diversas submissões. Sua eficiência será evidenciada conforme aumentar a frequência de utilização da ferramenta *ScienceView*, possibilitando o uso desses subsídios em uma turma presencial para identificar os erros mais frequentes e adaptar o conteúdo das aulas para sanar tais erros.

O restante deste artigo está organizado da seguinte forma. A Seção 2 apresenta trabalhos relacionados à visualização e avaliação de implementações submetidos em MOOC. A Seção 3 apresenta o método de pesquisa. As Seções 4 e 5 apresentam os resultados deste estudo, respectivamente a ferramenta de visualização para auxílio à avaliação e o relato de um estudo exploratório sobre a avaliação de atividades de programação com auxílio de técnicas de visualização. A Seção 6 conclui este artigo e apresenta trabalhos futuros relativos a esta pesquisa.

2. Trabalhos Relacionados

Os MOOCs disponibilizam diversos cursos por meio de plataformas de ensino online e para grande quantidade de pessoas. Considerando a grande quantidade de usuários e atividades avaliativas, pesquisadores buscam desenvolver ferramentas para auxiliar na correção dessas atividades, utilizando técnicas de mineração e visualização.

Yin et al. extraem uma árvore de sintaxe abstrata de cada atividade implementada pelos alunos e verificam a similaridade par a par por meio da distância de edição de árvore [Yin et al. 2015]. Tal abordagem pode agrupar algoritmos corretos e incorretos. Por exemplo: pode ocorrer erro lógico devido a uma ou mais instruções estarem invertidas, caso em que o cálculo de similaridade pode apontar que as implementações são semelhantes e agrupar códigos-fontes corretos com incorretos. Entretanto, essa abordagem possui alto custo computacional.

Glassman et al. extraem 60 características de submissões implementadas em Python, classificando as características em duas dimensões: alto nível e baixo nível [Glassman et al. 2014]. Primeiramente, realizam o agrupamento com a técnica *k-means*, utilizando as características de alto nível e, internamente nesses agrupamentos, executam o

agrupamento com a mesma técnica, mas para as características de baixo nível. O agrupamento em dois níveis é interessante devido à possibilidade de obter agrupamentos com erros semelhantes apenas das características de alto nível e, após a realização do segundo agrupamento, das características de baixo nível, permitindo a realização do *feedback* em duas etapas.

Taherkhani et al. apresentam a ferramenta Aari para classificar e reconhecer algoritmos de ordenação [Taherkhani et al. 2012]. Para isso, extraem diversas características: numéricas, descritivas, e outras típicas de algoritmos de ordenação. Entretanto, é necessário que treine a ferramenta para reconhecer os algoritmos desejados, caso contrário, o classificador não estará apto e ocorrerão diversos erros de classificação.

Glassman et al. apresentam a ferramenta OverCode que extrai características por meio do histórico de execução da análise dinâmica [Glassman et al. 2015]. A similaridade das submissões é calculada por meio da comparação entre blocos do programa. A partir disso é utilizado o conceito de pilha para apresentar o agrupamentos das implementações semelhantes e, ao clicar em um agrupamento, é possível verificar as linhas de código que formaram aquele agrupamento. Essa abordagem proporciona o desenvolvimento de vários recursos na ferramenta, como reescrever as variáveis que possuem a mesma atribuição, e mostrou que a combinação de blocos de código e análise dinâmica pode produzir bons agrupamentos.

Wei e Wu realizam a extração de características por meio da normalização do código-fonte e da análise do estilo de escrita [Wei e Wu 2015]. Para isso, separaram cada implementação em pedaços de código, condizentes a uma função, para verificar a similaridade entre eles. Com isso, o instrutor corrigiria pedaços do código ao invés de corrigir todo o algoritmo. O autor encontra a menor *substring* do bloco a partir de comparações *token* a *token* para verificar a similaridade.

Embora quase todos os trabalhos utilizem algum algoritmo de agrupamento, apenas um emprega técnicas de visualização para auxiliar a correção [Yin et al. 2015]. Os demais contam com diferentes graus de suporte, com auxílio de ferramentas. Desta forma, dificulta-se o entendimento da similaridade entre as atividades de programação e, portanto, a qualidade da correção dos programas.

3. Método

Após a revisão da literatura, verificamos que o nosso estudo pode ser dividido em algumas etapas. Primeiramente é necessário escolher qual linguagem de programação será considerada para as atividades de programação a serem avaliadas. Em seguida, deve-se verificar quais características podem ser extraídas das implementações submetidas utilizando a linguagem especificada. Posteriormente, é necessário construir a base de dados por meio da submissão de implementações, realizar a extração de características, minerar os dados extraídos das submissões e verificar a similaridade entre as implementações. Por fim, realizar a projeção multidimensional e visualizar os possíveis agrupamentos formados na projeção.

A linguagem de programação refere-se a definição de qual linguagem será considerada para as submissões a serem avaliadas. Foi definida a utilização da linguagem de programação `Python`. A escolha foi devido a disponibilidade de cursos sobre introdução a programação utilizando Python em diversos MOOCs e cursos de graduação.

Com base em todos os trabalhos relacionados, podemos perceber que os três tipos de análise – estática, estilo de escrita e dinâmica – obtiveram bons resultados. Considerando a combinação de características originadas de tipos de análises distintas, utilizamos uma combinação de aspectos provenientes da análise estática e do estilo de escrita. A escolha da análise estática deu-se pela possibilidade de implementações para solucionar o mesmo problema possuírem o mesmo tamanho. Adicionalmente, a organização do código, realizado pelo programador, pode impactar no entendimento da solução do problema: por isso a escolha da análise do estilo de escrita.

As características a serem extraídas da análise estática foram a quantidade de linhas de código e a complexidade ciclomática da implementação. Em relação a análise do estilo de escrita, foram consideradas todas as violações do estilo de escrita definido pelo ferramenta `PEP8` [van Rossum et al. 2001].

Em seguida, construímos a base de dados que será utilizada no experimento. É necessário que o conjunto de implementações obtenha uma quantidade considerável de códigos-fontes a fim de avaliar a utilização da ferramenta para MOOC. A base de dados desse estudo possui implementações desenvolvidas em Python de um curso introdutório à Computação e Programação.

A partir da base de dados, realizamos a extração de características referentes ao estilo de escrita e à análise estática (quantidade de linhas de código-fonte e a complexidade ciclomática). A análise e medida dessas características foram obtidas com auxílio da ferramenta `flake8` [Cordasco et al. 2010].

Com a finalidade de encontrar relações de similaridade entre os códigos-fontes com base nos valores das características extraídas para eles, utilizamos a técnica de similaridade do cosseno. Após a extração de características, obtém-se uma representação matricial dos dados. Esta é, tipicamente, uma matriz esparsa (com grande quantidade de zeros), dado que os estudantes buscam implementar corretamente as soluções das atividades, o que reduz a quantidade de erros referentes ao estilo de escrita considerados neste trabalho. Nesse tipo de situação, a similaridade do cosseno é mais utilizada por não favorecer comparações entre zeros.

Nosso objetivo, por meio dessas etapas, consistiu no desenvolvimento de subsídios para avaliação de programas, utilizando técnicas de visualização para auxiliar os professores a corrigirem todas as submissões, considerando o tempo de correção e a qualidade do *feedback* referente a visualização da ferramenta *ScienceView* para o professor.

O subsídio de avaliação foi uma adaptação da ferramenta `ScienceView` que, originalmente, verifica as mudanças nas relações de artigos científicos com o decorrer do tempo, utilizando técnicas de mineração visual embasadas na técnica de projeção multidimensional chamada T-LSP [Alencar et al. 2012].

Para validar o tempo gasto para todas as correções, foi realizado um estudo piloto com participação de 4 pessoas e organizado em quatro etapas. A primeira consistiu de treinamento quanto à utilização da ferramenta ScienceView para uma base de atividades de programação específica para esta finalidade. Nas duas etapas seguintes do estudo, os participantes corrigiram trabalhos de uma outra base de atividades, sendo organizados em dois grupos: na segunda etapa, o primeiro grupo avaliou as atividades utilizando a ferramenta e o segundo grupo realizou a correção manualmente, utilizando as medidas

extraídas pelas ferramentas e com um auxílio de uma planilha; na terceira etapa, ocorreu a inversão das atividades realizadas entre os grupos. Finalmente, na última etapa do estudo, os participantes responderam um questionário.

O treinamento foi realizado na forma de tutorial com a apresentação dos critérios de avaliação (estilo de escrita e complexidade ciclomática) e da ferramenta, com duração de 55 minutos. Especificamente quanto à ferramenta, foram apresentados todos os passos referentes ao seu funcionamento, desde a inserção de uma nova coleção no banco de dados até a visualização das submissões com a visualização da projeção criada a partir da coleção. Em seguida, foi apresentado como utilizar a ferramenta para auxiliar na avaliação das implementações: visualizar os agrupamentos formados ao longo das projeções, selecionar um determinado conjunto de implementações, identificar as características semelhantes de tais grupos e abrir mais de uma implementação simultaneamente. Durante o treinamento, foi utilizada uma base de dados distinta para não enviesar o estudo.

Encerrado o treinamento, procedeu-se para correção das submissões utilizando a ferramenta ScienceView ou de forma manual, conforme previamente descrito. Para esta etapa, foi utilizada uma base de 152 submissões em Python, de 5 problemas distintos, referentes a um curso introdutório à Computação com 49 alunos, trabalhados durante um semestre. Desta base, foram considerados os tipos de erros apontados na extração de características e a criação de um relato com os programas avaliados e comentários de correção. Cada uma das duas etapas da correção (manual e com a ferramenta) teve duração de 20 minutos, permitindo comparar a quantidade de correções efetuadas por período. Das 4 pessoas que participaram do estudo, uma não conseguiu utilizar a ferramenta ScienceView durante esta etapa do estudo. Assim, relatamos os resultados referentes a 3 correções com auxílio da ScienceView e 4 correções manuais, utilizando a tabela de medidas.

4. Ferramenta ScienceView para avaliação de atividades de programação

A ferramenta *ScienceView* foi desenvolvida para criar mapas de documentos temporais que exibam as mudanças nas relações de similaridade entre artigos científicos ao longo do tempo, utilizando a projeção multidimensional dinâmica T-LSP a partir de artigos nos formatos *ISI*, *Endnote* ou *BibTeX* [Alencar et al. 2012]. A adaptação dela para os propósitos deste trabalho exigiram alterações quanto aos formatos de dados de entrada e na manipulação dos tipos de valores nas implementações das técnicas de mineração e visualização empregadas.

A extração das características do programas foi realizada com a ferramentas de código-aberto `flake8` [Cordasco et al. 2010]. No entanto, o formato de saída dos resultados das ferramentas não era adequado. Portanto, foi necessário modificações no `flake8` para gerar os dados em um formato adequado para utilização na ferramenta ScienceView. Optou-se por gerar os dados no formato CSV, dada a facilidade de sua geração e manipulação. A ScienceView foi então alterada para permitir a importação de dados nesse formato.

Originalmente, a ScienceView suportava apenas números inteiros em sua representação interna, o que é razoável dado os tipos de dados de entrada originais, o modelo de representação utilizado e o compromisso de reduzir os requisitos de memória. No entanto, alguns dos valores importados do formato CSV podem ser números reais (por

exemplo, complexidade ciclomática média). Desta forma, foram alteradas as formas de representação e cálculos para utilizar números em ponto flutuante de precisão dupla.

As Figuras 1 e 2 apresentam a visualização de uma base com 152 atividades de programação em Python. Cada ponto da visualização é referente a uma submissão, e a técnica de projeção posiciona essas instâncias de submissão de forma que submissões similares permaneçam próximas no mapa da projeção. Ao clicar em um dos pontos, uma interface exibe sua implementação e as características extraídas para aquela submissão. Também é possível ordenar as colunas de características *Quantity* e *Normalized* em ordem crescente ou decrescente para visualizar as características que mais ocorreram.

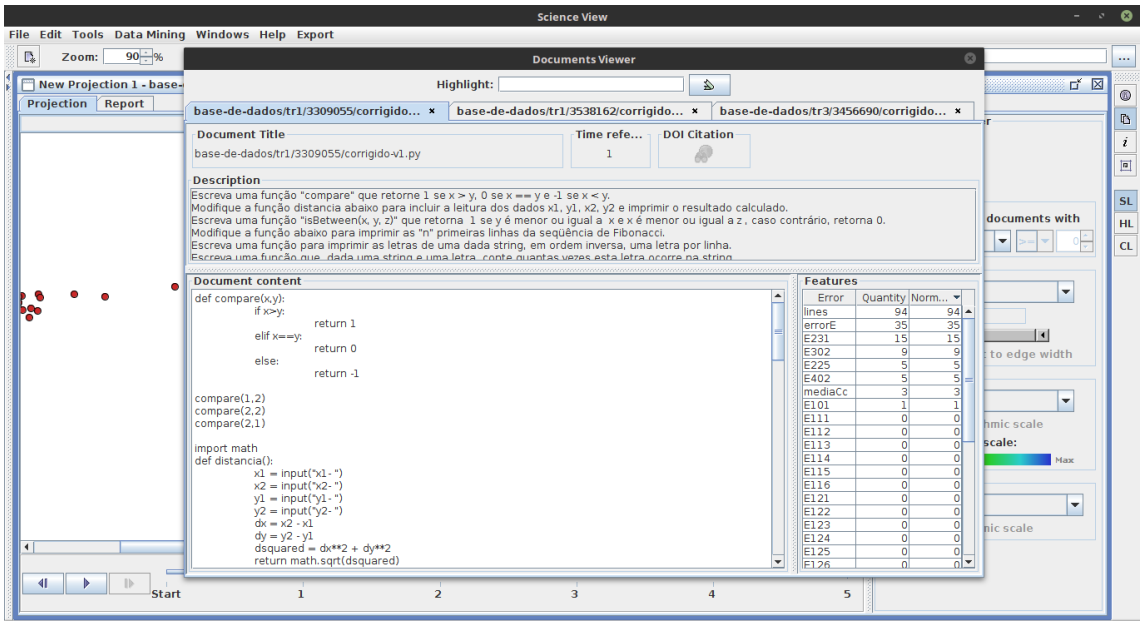

Figura 1. Visualização dos agrupamentos gerado pela ScienceView.

Uma das funcionalidades da ferramenta é a extração de tópicos ao decorrer das projeções. Desta forma é possível visualizar os agrupamentos encontrados pela *ScienceView*, bem como os tópicos ou características mais proeminentes de cada grupo extraídos por uma técnica baseada em covariância. Os agrupamentos são obtidos automaticamente pela ferramenta pelo uso do algoritmo de agrupamento DBSCAN no resultado da projeção. A Figura 2 exibe os agrupamentos formados pela ferramenta no decorrer da projeção no instante de tempo 3. É possível verificar os agrupamentos formados, pois eles são visualmente delimitados por polígonos na cor cinza e, para cada agrupamento, são apresentadas as características mais relevantes àquele agrupamento (tópico).

A ferramenta também realiza o rastreamento dos grupos simultaneamente. Consequentemente, além de gerar os grupos para cada projeção na sequência, a *ScienceView* também rastreia esses grupos ao longo do tempo. No entanto, isso não foi explorado no âmbito deste artigo.

5. Resultados

O estudo foi realizado com quatro participantes, organizado em quatro etapas: treinamento, correções (manual e com auxílio da ferramenta) e submissão de questionário,

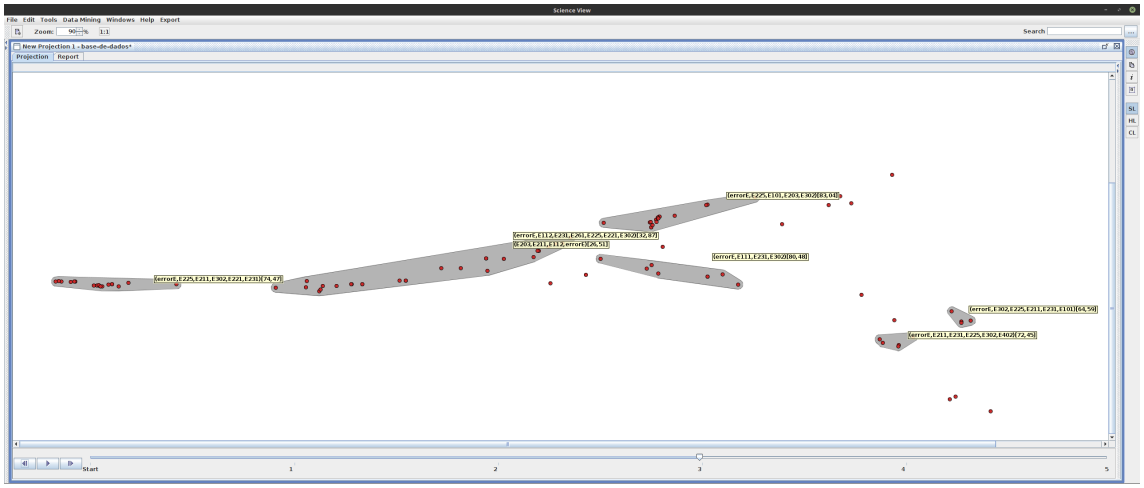

Figura 2. Exibição das projeções com a extração de tópicos ativada.

conforme apresentado na Seção 3. Apresentamos os resultados referentes às correções e ao questionário.

Na primeira parte da correção (etapa 2 do estudo), 2 voluntários realizaram a correção manual, utilizando a tabela de medidas extraídas das submissões, enquanto os outros 2 utilizaram a ferramenta ScienceView para auxiliar na correção. Na segunda parte da correção (etapa 3 do estudo), um dos contribuintes que realizou a correção manual passou a utilizar a ScienceView enquanto os outros 2, que utilizaram a ferramenta na etapa anterior, realizaram a correção manual.

A Tabela 1 apresenta a quantidade de submissões corrigidas por cada participante. A primeira linha identifica qual participante realizou as correções e a primeira coluna indica a forma de correção realizada pelos voluntários. Então, é apresentado a quantidade de correções que cada participante conseguiu realizar.

	Voluntário 1	Voluntário 2	Voluntário 3	Voluntário 4
Correção tradicional	3 códigos-fontes	4 códigos-fontes	3 códigos-fontes	4 códigos-fontes
ScienceView	Não utilizou a ferramenta	11 códigos-fontes	6 códigos-fontes	4 códigos-fontes

Tabela 1. Quantidade de correções realizadas pelos participantes do estudo.

Utilizando a ferramenta ScienceView para auxiliar nas correções, todos os participantes do estudo conseguiram corrigir, pelo menos, a mesma quantidade de submissões e dois desses corrigiram mais submissões. Desta forma, podemos observar que a ferramenta permite reduzir o tempo de correção das submissões.

Complementando a análise dos resultados das correções, prosseguimos com a exposição dos resultados do questionário. Ele continha questões objetivas e dissertativas para estabelecer o perfil do voluntário e avaliar a qualidade do treinamento e da ferramenta. Enquanto as questões objetivas visavam as informações profissionais e a qualidade do treinamento da ferramenta e se é possível utilizá-la para auxiliar na correção, as questões dissertativas visavam encontrar lacunas observadas pelo participante para uma

possível atualização da ferramenta. Um dos participantes não respondeu o questionário.

Sobre o nível de conhecimento em Python, 2 voluntários afirmaram possuir conhecimento intermediário sobre a linguagem de programação, enquanto 1 voluntário declarou ter conhecimento básico. Outra questão foi quanto ao tempo de experiência como professor, o que poderia interferir na quantidade de correções realizadas com a ferramenta. Dentre eles, 2 voluntários possuíam menos que 1 ano de experiência, enquanto 1 possui mais de 10 anos de experiência como professor.

Ao serem questionados se o treinamento ajudou a utilizar a ferramenta, todos os respondentes afirmaram que o treinamento em forma de tutorial da ScienceView contribuiu para sua utilização. Obtivemos êxito com a adaptação da interface que apresenta o código-fonte e a tabela de erros, visto que, ao serem questionados se faltava alguma informação nessa interface, todos responderam negativamente.

Ao relatarem sobre o *feedback* recebido da ScienceView por meio da visualização da projeção, 2 participantes afirmaram que os agrupamentos auxiliaram na verificação de implementações com características similares. O outro participantes constatou a possibilidade de utilizar a ferramenta em uma turma fechada de alunos a fim de verificar quais os principais erros deles por meio da extração de tópicos da ferramenta.

Sobre a experiência em relação ao uso da ferramenta, um dos participantes admitiu que, como não utilizou muito a ferramenta, foi mais rápido realizar as correções manuais. Contudo, outra resposta afirmou que a utilização mais frequente da ScienceView possibilitará analisar as implementações de forma mais rápida.

6. Conclusões

Por meio dos resultados obtidos, observamos indícios da eficácia dos mecanismos de visualização e de sua utilidade para auxiliar a correção de atividades de programação. No entanto, durante o estudo também foi possível observar algumas importantes limitações. Uma delas está relacionada com as características extraídas das atividades de programação. Por enquanto, elas estão restritas a estilo de escrita e complexidade, o que é um importante limitador para a correção efetiva de programas. Outro ponto é quanto ao uso da ferramenta para avaliação. Foi possível observar que os participantes realizaram poucas correções de submissões que estavam no mesmo agrupamento. Embora o treinamento tenha provido informações sobre os agrupamentos, não foi fornecida uma estratégia explícita para correções considerando a visualização.

Desta forma, em continuidade a este trabalho, é necessário investigar mais características quanto às submissões. Por exemplo, características relacionadas à análise dinâmica e quanto à execução de casos de teste podem ser consideradas, principalmente aquelas que podem ser obtidas dos mecanismos de avaliação automática utilizados atualmente em MOOCs. O estabelecimento de estratégias para avaliação das atividades também precisa ser abordado, considerando o tamanho dos grupos e a evolução deles durante o oferecimento das disciplinas.

Finalmente, este estudo deve considerar uma base de dados que represente adequadamente as submissões existentes em MOOCs. Atualmente encontra-se em construção uma base de dados considerando programas submetidos em MOOCs de programação em Python, obtidos a partir do Github. Com uma base de dados mais representativa e das

melhorias sugeridas nesta seção, espera-se o desenvolvimento de uma abordagem efetiva para a avaliação de atividades de programação.

Referências

Alencar, A. B., Börner, K., Paulovich, F. V., Oliveira, M. C. F. d. (2012). Time-aware visualization of document collections. In *27th Annual ACM Symposium on Applied Computing*, p. 997–1004, New York, NY, EUA. ACM.

Chen, C. (2006). *Information Visualization: Beyond the Horizon*. Springer-Verlag, Secaucus, NJ, EUA.

Cordasco, I. et al. (2010). Flake8: Your tool for style guide enforcement. Programa de computador.

Glassman, E. L., Scott, J., Singh, R., Guo, P. J., Miller, R. C. (2015). OverCode: Visualizing variation in student solutions to programming problems at scale. *Transactions on Computer-Human Interaction*, 22(2):7:1–7:35.

Glassman, E. L., Singh, R., Miller, R. C. (2014). Feature engineering for clustering student solutions. In *1st ACM Conference on Learning @ Scale Conference*, p. 171–172, New York, NY, EUA. ACM.

Klobas, J. E., Mackinsosh, B., Murphy, J. (2014). The anatomy of MOOCs. In Kim, P., editor, *Massive open online courses: the MOOC revolution*, cap. 1, p. 1–22. Routledge, New York, NY, EUA.

Oliveira, M. Levkowitz, H. (2003). From visual data exploration to visual data mining: a survey. *IEEE Transactions on Visualization and Computer Graphics*, 9(3):378–394.

Paulovich, F. V., Nonato, L. G., Minghim, R., Levkowitz, H. (2008). Least Square Projection: A fast high-precision multidimensional projection technique and its application to document mapping. *IEEE Transactions on Visualization and Computer Graphics*, 14(3):564–575.

Rodriguez, C. O. (2012). MOOCs and the AI-Stanford like courses: Two successful and distinct course formats for massive open online courses. *European Journal of Open, Distance and E-Learning*, 15(2):1–13.

Schmidt, D. C. McCormick, Z. (2013). Producing and delivering a coursera MOOC on pattern-oriented software architecture for concurrent and networked software. In *Conference on Systems, Programming, & Applications: Software for Humanity 2013*, p. 167–176, New York, NY, EUA. ACM.

Taherkhani, A., Korhonen, A., Malmi, L. (2012). Automatic recognition of students' sorting algorithm implementations in a data structures and algorithms course. In *12th Koli Calling International Conference on Computing Education Research*, p. 83–92, New York, NY, EUA. ACM.

van Rossum, G., Warsaw, B., Coghlan, N. (2001). PEP8: Python style guide checker. Programa de comptuador. 1.5.7.

Wei, Z. Wu, W. (2015). A peer grading tool for MOOCs on programming. In *Intelligent Computation in Big Data Era: International Conference of Young Computer Scientists, Engineers and Educators*, volume 503, p. 378–385, Berlin, Alemanha. Springer.

Yin, H., Moghadam, J., Fox, A. (2015). Clustering student programming assignments to multiply instructor leverage. In *2nd ACM Conference on Learning @ Scale*, p. 367–372, New York, NY,EUA. ACM.