UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE COMPUTAÇÃO
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

EMERSON YUDI NAKASHIMA

AVALIAÇÃO DE ATIVIDADES DE PROGRAMAÇÃO SUBMETIDAS EM MOOC COM EMPREGO DE TÉCNICAS DE VISUALIZAÇÃO

MONOGRAFIA

CAMPO MOURÃO
2017

EMERSON YUDI NAKASHIMA

AVALIAÇÃO DE ATIVIDADES DE PROGRAMAÇÃO SUBMETIDAS EM MOOC COM EMPREGO DE TÉCNICAS DE VISUALIZAÇÃO

Trabalho de Conclusão de Curso de graduação apresentado à disciplina de Trabalho de Conclusão de Curso 2, do Curso de Bacharelado em Ciência da Computação do Departamento Acadêmico de Computação da Universidade Tecnológica Federal do Paraná, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Marco Aurélio Graciotto Silva

Coorientador: Prof[a]. Dr[a]. Aretha Barbosa Alencar

CAMPO MOURÃO
2017

ATA DA DEFESA DO PROJETO DE TCC

Às **13:50** do dia **20/06/2017** foi realizada na sala **E103** da UTFPR-CM a sessão pública da defesa do projeto de Trabalho de Conclusão do Curso de Bacharelado em Ciência da Computação do(a) acadêmico(a) **Emerson Yudi Nakashima**. Estavam presentes, além do(a) acadêmico(a), os membros da banca examinadora composta por:

- Orientador: **Prof. Dr. Marco Aurálio Graciotto Silva**;
- Membro 1: **Profa. Dra. Aretha Barbosa Alencar;**
- Membro 2: **Prof. Dr. Igor Scaliante Wiese;**
- Membro 3: **Prof. Dr. Lucio Geronimo Valentin**.

Inicialmente, o(a) acadêmico(a) fez a apresentação do seu trabalho, sendo, em seguida, arguida pela banca examinadora. Após as arguições, sem a presença do(a) acadêmico(a), a banca examinadora o(a) considerou aprovado na disciplina de Trabalho de Conclusão de Curso **2** e atribuiu, em consenso, a nota 9,0 (nove). Este resultado foi comunicado ao(à) acadêmico(a) e aos presentes na sessão pública e, posteriormente, deverá ser registrado no sistema acadêmico pelo professor responsável de TCC. Em seguida foi encerrada a sessão e, para constar, foi lavrada a presente Ata que segue assinada pelos membros da banca examinadora, após lida e considerada conforme.

Observações:

_____

_____

_____

_____

_____

_____

Campo Mourão, 20/06/2017

Profa. Dra. Aretha Barbosa Alencar Membro 1	Prof. Dr. Igor Scaliante Wiese Membro 2	Prof. Dr. Lucio Geronimo Valentin Membro 3	Prof. Dr. Marco Aurálio Graciotto Silva Orientador

A ata de defesa assinada encontra-se na coordenação do curso.

Resumo

Nakashima, Emerson Y. Avaliação de atividades de programação submetidas em MOOC com emprego de técnicas de visualização. 2017. 64. f. Monografia (Curso de Bacharelado em Ciência da Computação), Universidade Tecnológica Federal do Paraná. Campo Mourão, 2017.

Contexto: Em disciplinas de algoritmos ou programação, é necessário criar e implementar uma solução para um problema. Entretanto, há a possibilidade de possuir diversas formas de solucionar o problema e, consequentemente, formas de implementações. Desta forma, a quantidade de implementações possíveis é vasta, dificultando a avaliação delas pelo professor quanto ao custo, tempo e qualidade da avaliação. Para agravar essa dificuldade, cursos massivos, abertos e *online* (MOOC) possuem uma grande quantidade de usuários, inviabilizando a correção individual das submissões.
Objetivo: O objetivo deste trabalho é propor subsídios para a avaliação de programas submetidos em disciplinas introdutórias à computação, utilizando técnicas de mineração e visualização de dados para construir e apresentar agrupamentos de submissões semelhantes. Os subsídios propostos consistem na utilização de ferramentas para extração de características, padronização no armazenamento dessas características e a utilização de técnicas de agrupamento e visualização, com o auxílio de uma ferramenta.
Método: A primeira etapa consistiu na identificação das características que podem ser extraídas conforme o tipo de análise utilizado: características estáticas referentes a estilo de escrita e complexidade. Após a identificação, foi necessário o desenvolvimento de ferramentas para coletar tais medidas de forma que pudéssemos utilizá-las para realizar a projeção e visualização. Com isso, desenvolvemos uma ferramenta para analisar as informações disponíveis, realizando os agrupamentos e gerando uma visualização dos programas submetidos. Para avaliar a visualização de submissões com auxílio da ferramenta, utilizamos uma base de dados de implementações com soluções de cinco problemas distintos. A avaliação ocorreu em duas etapas: mediante a qualidade das visualizações, considerando as técnicas de mineração e visualização de dados; e verificando o *feedback* da ferramenta para o professor por meio de um questionário.
Resultados: Com uma base de dados de 152 implementações, obtivemos boa avaliação da qualidade dos agrupamentos. Quanto à qualidade da visualização para fins de avaliação

das submissões, realizamos um treinamento com a apresentação dos critérios de avaliação e da ferramenta. O estudo procedeu da utilização da ferramenta *ScienceView*, criando uma nova base de dados, e dividindo as correções em 2 grupos: um grupo realizara a correção tradicional e o outro utilizara a ferramenta para auxiliar na correção. Em seguida, esses dois grupos inverteram o modo como foi realizado as correções. Ao final do estudo, os voluntários avaliaram o treinamento e a ferramenta positivamente.

Conclusões: Considerando a preservação de vizinhança, a qualidade da projeção é compatível com outras projeções feitas com a técnica LSP e similares, apresentando resultados similares relatados na literatura. Em relação ao emprego de visualização para avaliação de programas, os resultados foram limitados devido ao emprego pouco eficiente da ferramenta e dos agrupamentos. No entanto, considerando os agrupamentos que continham programas avaliados, existem indícios de que a visualização pode ser utilizada com êxito e, se melhorarmos o treinamento, poderemos alcançar claramente nosso objetivo. Ainda assim, os participantes avaliaram como positiva a utilização da ferramenta. Como trabalho futuro, serão investigadas a utilização de outras características das submissões e o aperfeiçoamento da usabilidade e treinamento quanto ao uso da ferramenta.

Palavras-chaves: MOOC. Programação. Avaliação. Agrupamentos. Mineração de dados. Visualização.

Abstract

Nakashima, Emerson Y. Assessment of programming activities submitted in MOOC using visualization techniques. 2017. 64. f. Monograph (Undergradute Program in Computer Science), Federal University of Technology – Paraná. Campo Mourão, PR, Brazil, 2017.

Context: In algorithm or programming classes, it is necessary to create and implement a solution to a problem. However, there is the possibility of having several ways so solve a problem and, consequently, forms of implementations. This, the amount of possible implementations is vast, making it difficult for teachers to evaluate them within reasonable cost, time and quality. Moreover, massive courses, open and online (MOOC) have large numbers of users, aggravating this problem and making it unfeasible individual correction of submissions.
Objective: The objective of this project is evaluate and implement supporting tools to assess programming assignments submitted in introductory courses to computing using data mining and visualization techniques to build and show clusters of similar source codes.
Method: The first step consisted of identification of features that can be extracted according to static analysis and code style. After this identification, we developed tools to colect such measures so that we could use them for data mining and visualization. Thereby, we improved an existing tool for performing clusters and generating visualizations of submitted programming assingments. To evalute the tool, we have one collection of assignments' submissions with solutions of 5 different problems. The validation occurred in two steps: through the quality of the visualizations considering the mining techniques and data visualization; and checking the feedback of the tool by the teacher.
Results: With a database of 152 implementations, we obtained a good evaluation of the quality of the clusters. Regarding the quality of the visualization for the purpose of evaluating the submitted programs, we conducted an experimental study. The experimental study was based on the use of the ScienceView tool, creating a new database, training the study subjects with respect to the assessment criteria and tool, and assessing the programming submissions. We organized the participants into two groups: one group performed the traditional assessment and the other used the tool to aid in the assessment. These two groups then switched how corrections were made. At the end of the experimental study, the volunteers evaluated the training and the tool positively.

Conclusions: Considering the neighborhood preservation, the quality of the projection is compatible with other projections made with the LSP technique and the like, presenting similar results reported in the literature. In relation to the use of visualization for program assessment, the results were limited due to the inefficient use of the tool and the clusters. However, considering clusters containing programs evaluated by the experiment participants, there are indications that visualization can be used successfully and, if we improve training, we can clearly reach our goal. Nevertheless, the participants assessed the use of the tool as positive. As future work, we will investigate the use of other characteristics of the program and the improvement of usability and training regarding the use of the tool.

Keywords: MOOC. Programming. Assessment. Clustering. Data mining. Visualization

Lista de figuras

Lista de tabelas

Lista de acrônimos

MOOC	Massive Open Online Courses 10
IDE	Integrated Development Environment
TDD	Test Driven Development
DBSCAN	Density-based Algorithm for Discovering Clusters 16
AST	Abstract Sintax Tree 20
TED	Tree Edit Distance 22
AMI	Adjusted Mutual Information 24
OPTICS	Ordering Points To Identify the Clustering Structure 22
t-SNE	t-Distributed Stochastic Neighbor Embedding 22
k-NN	K Nearest Neighbor 30
T-LSP	Time-based Least Square Projection
GBL	Game Based learning

Sumário

Capítulo 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. Para realizar um curso ofertado pelo MOOC, é necessário apenas a conexão com a Internet e o cadastro na plataforma, geralmente gratuito. Foi popularizado em 2011 quando grandes universidades, como Instituto de Tecnologia de Massachusetts (MIT), Universidade de Harvard e Universidade de Stanford, tiveram a iniciativa mediados pelos provedores Cousera, edX e Udacity, respectivamente (MEHLENBACHER, 2012). Além de permitir explorar novos modelos de negócio (DELLAROCAS; ALSTYNE, 2013), possibilita englobar mais alunos a custos menores e com boa qualidade (SCHMIDT; MCCORMICK, 2013).

Em cursos de introdução a programação, seja MOOC ou presencial, há uma introdução sobre algoritmo seguido da seleção de uma linguagem de programação para desenvolvimento. Alguns desses cursos orientam a utilização de ferramentas, como o IDE e ensinam o funcionamento da linguagem sintaticamente. Entretanto, esse tipo de ensino tem levado o estudante a fazer seu programa baseado na tentativa e erro, visto que buscam gerar o algoritmo sem o total conhecimento lógico do problema (EDWARDS, 2003). Isso funcionou como um incentivo para que surgissem outras abordagens de ensino, como: baseado em jogos (GBL – Game Based learning) (KAPP, 2012), baseado em dicas (GLASSMAN et al., 2016), a reflexão na ação (*Reflection in action*) e o Test Driven Development para ensino (TDD) (CAMARA, 2016). A GBL utiliza recursos de jogos, como a pontução e o *ranking*, para motivar o estudante, podendo pontuar a cada problema resolvido, por exemplo. A segunda abordagem consiste na geração de dicas durante a resolução de um problema para ser apresentado a futuros estudantes. Na reflexão na ação, a implementação do *software* ocorre somente após o entendimento apropriado do problema (EDWARDS, 2004). No TDD, cumpre-se um ciclo dividido em três etapas: na primeira etapa o programador insere um teste que deve falhar no momento de sua execução; na segunda etapa é implementado a

solução para que o caso de teste escrito anteriormente seja aceito; e a última etapa refere-se a refatoração (simplificar ou melhorar) o código recém implementado (BECK, 2003).

As abordagens Test Driven Development e Reflexão na ação tornam-se interessantes para serem utilizadas em Massive Open Online Courses, visto que ambas necessitam do entendimento do problema a ser resolvido, mesmo que de formas diferente. O Test Driven Development necessita o entendimento do que não pode ser feito no problema, com isso, durante sua implementação, é possível realizar testes que possam falhar, caso viole alguma regra do problema. A Reflexão na ação é interessante devido às etapas impostas nessa abordagem. Dessa forma, é preciso entender o problema para que possa implementá-lo corretamente. Ambas as abordagens evitam que os aprendizes tentem simplesmente programar e corrigi-los conforme os erros são apresentados. Dessa forma, desenvolvem o raciocínio lógico, a interpretação dos problemas e sua implementação.

Principalmente em MOOC 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 necessário a utilização de mecanismos de avaliação automática ou semiautomática (SCHMIDT; MCCORMICK, 2013). Para alguns tipos de atividade, a avaliação automática é bem simples. Por exemplo, para verificar as soluções referentes às questões de múltipla escolha é necessário somente comparar a alternativa selecionada com o gabarito de questões (ALARIO-HOYOS et al., 2013). No entanto, implementações de programas computacionais 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 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 de agrupamentos (*clusters*). Por exemplo, tais agrupamentos podem ser realizados por meio da similaridade dos códigos. Compreendendo as variações de implementação, é possível extrair caraterísticas por meio da análise estática (YIN et al., 2015; GLASSMAN et al., 2014; TAHERKHANI et al., 2012), análise dinâmica (GLASSMAN et al., 2015) e análise do estilo de escrita (WEI; WU, 2015). Com isso, o agrupamento das submissões semelhantes infere na correção efetiva de poucos projetos, já que todos os outros códigos que estão no agrupamento serão parecidos, permitindo que o professor dedique o tempo poupado na correção de diversas implementações para aprimorar as correções de poucos trabalhos, possibilitando *feedbacks* mais precisos e 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. Os seguintes subsídios foram investigados e desenvolvidos para a avaliação de programas submetidos em MOOC:

- Agrupamento dos códigos-fontes semelhantes, utilizando técnicas de mineração;
- Seleção de uma técnica de projeção adequada para o mapeamento dos *clusters*;
- Ferramenta de visualização;
- Material para os professores sobre como corrigir as submissões utilizando a ferramenta.

Como metas, estabeleceu-se o desenvolvimento de uma ferramenta para recuperação de dados a partir de códigos-fontes e da alteração de uma ferramenta para mineração e visualização de dados (ALENCAR et al., 2012). Para extrair as características do estilo de escrita e a complexidade ciclomáticas dos dados, foi realizado modificações no código-fonte do `Flake8` (CORDASCO, 2010) e no `PEP8` (ROSSUM et al., 2001b) para que fosse possível armazenar a extração de características em um arquivo. Encontrar e modificar essas ferramentas foi fruto de trabalhos anteriores a serem utilizados no presente trabalho.

Com o auxílio dessas ferramentas, foram investigadas as características e técnicas para mineração e visualização das submissões, avaliando-se como contribuir para sua correção, o tempo em que foi necessário para que os professores corrigissem todas as submissões e a qualidade dos agrupamentos realizados pelo sistema. Decidimos pela utilização do cálculo de similaridade do cosseno a fim de verificar a similaridade das submissões e o uso do algoritmos de agrupamento DBSCAN (ESTER et al., 1996).

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 desta proposta de trabalho de conclusão de curso é organizado da seguinte forma. O Capítulo 2 apresenta o referencial teórico sobre: MOOC e visualização, além dos trabalhos relacionados à visualização e avaliação de implementações submetidos em MOOC. O Capítulo 3 apresenta o método de pesquisa, a estratégia para avaliação e as ferramentas modificadas e utilizados. O Capítulo 4 descreve dois estudos sobre a utilização da ferramenta. O Capítulo 5 conclui este estudo, apresentando considerações finais e trabalhos futuros.

Capítulo 2

Referencial Teórico

Neste capítulo são apresentados os conceitos utilizados na pesquisa, bem como os estudos que fundamentaram esse projeto. A Seção 2.1 apresenta o conceito de MOOC e as principais características da plataforma de ensino. A Seção 2.2 descreve as os principais conceitos relacionados à mineração e visualização: mineração, visualização, projeção multidimensional, agrupamento e classificador. A Seção 2.3 descreve as características que podem ser extraídas dado a escolha de um dos tipos de análise de código-fonte. Finalmente, a Seção 2.4 apresenta as pesquisas relacionadas a este estudo juntamente com suas limitações.

2.1. Massive Open Online Courses (MOOC)

Após pesquisar o conceito de Curso Massivo, Aberto e *Online*, Fassbinder et al. (2014) observam que não há uma definição comum na literatura. Encontram-se três descrições distintas para o termo. Sivamuni e Bhattacharya (2013) afirmam que o MOOC, em conformidade com o dicionário Oxford, é um curso oferecido por meio da Internet, de forma gratuita, para uma grande quantidade de alunos. Subbian (2013) reitera que o MOOC disponibiliza curso gratuito, baseado na *web*, com registro livre de taxas monetárias e compartilhamento público de currículo. Por fim, Siemens (2013) defende a tendência em inovação e experimentação do uso da tecnologia para o ensino a distância e *online* a fim de dar oportunidade de aprendizagem de forma massiva. Não obstante, observa-se a concordância quanto ao oferecimento pela Internet e para grande quantidade de pessoas (massivo).

Klobas e Murphy (2014) afirmam que massivo refere-se a capacidade do MOOC em suportar uma grande quantidade de alunos. Tal quantidade é bem superior ao número de discentes que uma sala pode acomodar ou o total de participantes em um curso *online* antes do surgimento do MOOC. Por exemplo, um curso de Inteligência Artificial foi ofertado,

online e gratuitamente, pela Universidade de Stanford em 2011, no qual houveram 160.000 estudantes inscritos (RODRIGUEZ, 2012).

Os principais recursos das plataformas utilizados para MOOC são a integração com outras aplicações, como a utilização de e-mails e fóruns, o uso de questionário relacionados com os vídeos e a inclusão de atividades para estimular e motivar os alunos (FASSBINDER et al., 2014). Os fóruns de discussão auxiliam os usuários a aprenderem sobre o problema abordado (SCHMIDT; MCCORMICK, 2013). Em adição, também é possível utilizar recursos para avaliar alguns tipos de teste, como as questões de múltipla escolha, na qual verifica-se a alternativa selecionada com a presente no gabarito de questões (ALARIO-HOYOS et al., 2013). Em relação aos cursos de programação ofertados no MOOC, por experiência, notou-se somente o uso de vídeo aulas, no qual um especialista disserta sobre a linguagem e apresenta alguns recursos, perguntas objetivas relacionadas ao vídeo e ao texto presente naquela etapa do ensino e é solicitado a implementação de um problema ao final de cada curso e, as vezes, no decorrer do curso. Devido a quantidade massiva de usuários que podem realizar e submeter as implementações dos exercícios propostos, focaremos no desenvolvimento de recursos avançados de visualização dos dados obtidos por meio das implementações submetidas em cursos de programação e na realização de *feedback* para que o usuário possa verificar seu nível de conhecimento.

2.2. Mineração e visualização de dados

A mineração de dados é uma etapa do *Knowledge Discovery in Databases* (FAYYAD et al., 1996) que busca descobrir padrões em grandes conjuntos de dados, podendo utilizar métodos de inteligência artificial, estatísticos e sistemas de banco de dados (CHAKRABARTI et al., 2006). O objeto do processo de mineração de dados consiste na extração automática de informações de um conjunto de dados e transformá-lo em uma estrutura compreensível para uso posterior (CHAKRABARTI et al., 2006). A Figura 2.1 apresenta os estágios da mineração de dados: o Conhecimento do Domínio condiz com a identificação do problema para possuir um conhecimento inicial e definir as metas e os objetivos a serem atingidos no processo de extração de conhecimento; a etapa Pré-processamento refere-se a padronização e limpeza de dados extraídos de fontes diversas e a escolha de um subconjunto representativo; a fase seguinte, Extração de Padrões, consiste na aplicação do algoritmo de mineração de dados escolhido; o estágio Pós-processamento equivale a análise dos padrões obtidos anteriormente e possibilita a extração de outros padrões; por fim, a Utilização do Conhecimento é a utilização dos dados extraídos em algum sistema ou diretamente pelo usuário. Observe que esse sistema pode se realimentar, caso seja necessário adicionar mais informações. Dessa forma, necessita-se realizar essas etapas novamente, fornecendo ao sistema novas características. A realimentação ocorre após a análise do Pós-processamento, caso os padrões adquiridos não sejam eficientes

para a aplicação. Com isso é possível retornar a etapa Pré-processamento a fim de adicionar ou remover qualquer dado.

Figura 2.1. Etapas da mineração de dados (REZENDE et al., 2003)

Muitos métodos de mineração de dados são baseados em técnicas de treinamento e teste de aprendizagem de máquina, e reconhecimento de padrões e estatísticas, como os algoritmos de classificação e agrupamentos, respectivamente (FAYYAD et al., 1996).

Classificação é a tarefa de aprendizagem de uma função alvo que mapeia cada conjunto de atributos a um dos rótulos de classe predefinidas (TAN et al., 2005a). Uma função alvo auxilia uma ferramenta, que possui informações, a distinguir os objetos de diferentes classes. A fim de realizar essas classificações, são implementados classificadores com diversas abordagens, como árvores de decisão, redes neurais e *support vector machine*, por exemplo. Esses classificadores são mais utilizados para predizer ou descrever um conjunto de dados com categorias binárias ou nominais (TAN et al., 2005a). Por exemplo, para classificar um animal como mamífero, réptil, peixe, anfíbio ou pássaro, deve-se sumarizar dados como a temperatura do corpo, característica da pele, se é uma criatura aquática, se possui patas e hiberna.

Em relação ao agrupamento, o que difere a classificação do agrupamento é que o último é formado por meio da comparação de informações entre os objetos e não existem rótulos pré-definidos, enquanto o primeiro é realizado por meio da comparação das informações do objeto com os dados contidos em cada classe predefinida da função alvo. Desta forma, os objetos dentro de um grupo devem ser similares ou relacionados entre si e diferentes ou não relacionados entre objetos de grupos diferentes, ou seja, quanto maior a similaridade dos objetos dentro de um grupo e mais diferentes são os agrupamentos, melhores ou mais distintos os agrupamentos formados (TAN et al., 2005b). O K-means (MACQUEEN et al.,

1967) e o Density-based Algorithm for Discovering Clusters (DBSCAN) (ESTER et al., 1996) são exemplos de algoritmos de agrupamento.

Após a mineração dos dados, é desejável a utilização de ferramentas para auxiliar na criação de hipóteses sobre conjuntos de dados complexos – grande conjunto de dados ou de alta dimensionalidade – para que os analistas possuam capacidade de explorá-los e compreendê-los (OLIVEIRA; LEVKOWITZ, 2003). 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; LEVKOWITZ, 2003; KEIM, 2002). Ademais, a visualização de dados é intuitiva, permitindo explorar os dados mais rapidamente e fornecendo melhores resultados na maioria das vezes (KEIM, 2002).

Entretanto, é uma tarefa complexa gerar visualizações de qualidade e intuitivas para dados numerosos ou com alta dimensionalidade. Assim sendo, uma opção é a utilização de técnicas de projeção que criam o mapeamento de dados para reconhecimento visual (FRIEDMAN; TUKEY, 1974) e, com isso, produzir a visualização. Devido a alta dimensionalidade dos dados, é recomendável a utilização de projeções multidimensionais. Essa técnica realiza a diminuição n-dimensional, sendo n uma alta dimensão, para uma espaço unidimensional, bidimensional ou tridimensional (PAULOVICH et al., 2008), sempre buscando a menor perca de informação possível, como a relação de similaridade do espaço n-dimensional no espaço unidimensional, bidimensional ou tridimensional. Essa relação é encontrada, utilizando cálculos de similaridade ou dissimilaridade entre os dados. Instâncias são representadas por marcadores visuais no espaço tipicamente bidimensional de forma que instâncias com características similares permaneçam próximas. Devido a favorecerem a percepção de similaridade entre instâncias, projeções permitem identificar grupos de instâncias altamente relacionados e fronteiras entre grupos.

Para criar essa visualização deve-se selecionar o formato de como as características extraídas serão armazenadas, como o vetor de características ou o modelo de tabela de dados (OLIVEIRA; LEVKOWITZ, 2003).

2.2.1. Similaridade dos dados

De posse de uma representação da coleção de objetos, por exemplo uma representação vetorial, é possível utilizar medidas de similaridade para quantificar a semelhança entre duas distâncias. Medidas de similaridade retornam valores entre 0 e 1, podendo ser exatamente iguais (ALENCAR, 2013). Quanto mais próximo de 1, mais parecido os objetos são. E quanto mais próximo de 0, mais diferentes serão os objetos comparados.

Para uma medida ser considerada uma métrica, é necessário conter quatro propriedades: não-negatividade, identidade, simetria e desigualdade triangular. Não obstante, há

medidas, como similaridade, que não obedecem a essas propriedades, mas que também são importante (PAULOVICH, 2008), pois resultam em um grupo maior de medidas que pode conter qualquer tipo de função que compare numericamente o quanto dois objetos podem ser diferentes (TAN et al., 2005c).

Considerando espaços de alta dimensão e esparsos, é preferível a utilização da similaridade do cosseno (PAULOVICH, 2008). pelo fato de não realizar comparações entre zeros e da limitação do conjunto de características com medidas diferentes de 0 (zero), visto que os dados são correspondentes se considerar as características que não possuem ocorrência, iguais a 0 (zero) (TAN et al., 2005c; PAULOVICH, 2008).

Além de atribuir medidas adequadas ao modelo de dados de alta dimensionalidade e esparsos, alguns cuidados são necessário quanto aos valores de cada característica e ao desbalanceamento. Por exemplo, Paulovich (2008) afirma que há dois cenários distintos que podem adulterar o resultado: quando a norma Euclidiana do vetor de característica que representa os objetos são muito diferentes; e quando as características possuem valores em escalas diferentes dos demais atributos do vetor. Tais cenários podem ser evitados. Para evitar a primeira situação, aplica-se o processo de normalização para que todos os vetores possuem norma Euclidiana unitária. A segunda situação evita-se por meio da aplicação de um processo chamado padronização (*standardization*) (TAN et al., 2005c), atribuindo novos valores com média igual a 0 (zero) e desvio padrão igual a 1 (um), por meio da transformação da média daquela característica específica.

Para isso, deve-se avaliar: a qualidade da projeção quanto a preservação das vizinhanças; e a qualidade dos agrupamentos formados com base nessas projeções, conforme apresentado na próxima seção.

2.2.2. Qualidade do agrupamento e da projeção

O objetivo dos algoritmos de agrupamento é encontrar grupos significativos presentes em um conjunto de dados (HALKIDI et al., 2001), de forma que cada agrupamento possua dados muito semelhantes entre si e discrepante com os dados de outros grupos. Independentemente do algoritmo utilizado e da estrutura dos dados, esses algoritmos sempre encontrarão grupos dentro do conjunto de dados (TAN et al., 2005b). Por isso, é necessário verificar a qualidade do agrupamento para garantir que todos os objetos de um determinado grupo estão condizentes conforme o cálculo de similaridade selecionado.

Entretanto, há uma série de problemas para a validação do agrupamento (TAN et al., 2005b): distinguir se há ocorrências de estruturas não aleatórias no conjunto de dados; determinar o número correto de agrupamentos; avaliar os resultados da análise do agrupamento sem informações externas dos dados; comparação do resultado da análise do agrupamento com resultados externos; e a comparação de dois agrupamentos distintos para determinar qual é o melhor.

Além de verificar a qualidade do agrupamento também devemos observar a qualidade da projeção. Como citado na Seção 2.2, técnicas de projeção multidimensionais transformam a alta dimensionalidade dos dados para uma, duas ou três dimensões. Contudo, é necessário conservar as relações de vizinhança (BAUER; PAWELZIK, 1992). Desta forma, a distância entre os objetos após a utilização de uma técnica de projeção multidimensional deve ser o mais próximo possível da distância entre eles no espaço de alta dimensionalidade.

Apesar do conhecimento dessas propriedades, Bauer e Pawelzik (1992) afirmam que é difícil qualificar a vizinhança após a utilização de uma técnica de projeção multidimensional. Pelo fato de, geralmente, somente ser possível verificar os erros de preservação da vizinhança por meio do controle visual da projeção.

Devido a essa dificuldade é necessário utilizar técnicas que possam verificar se um objeto foi classificado corretamente àquele grupo, como a pontuação silhueta, e verificar a qualidade da projeção, como a Preservação da vizinhança, que analisa se os vizinhos de um objeto na projeção são os mesmos vizinhos no espaço multidimensional.

Pontuação silhueta

Pontuação silhueta é um método, bem popular, que combina: coesão e separação (TAN et al., 2005b). Coesão, ou compacidade, determina que os objetos de um determinado agrupamento devem ser o mais semelhante possível. Enquanto a separação indica o quão diferentes os agrupamentos são entre si (TAN et al., 2005b; BERRY; LINOFF, 1997).

Rousseeuw (1987) e Tan et al. (2005b) definem três passos para a criação da pontuação silhueta em um determinado conjunto de dados. Primeiro, para cada objeto de um agrupamento é calculada a distância média em relação a todos os outros objetos pertencentes a esse agrupamento. O segundo passo consiste no cálculo da distância média entre um objeto e os objetos de outro agrupamento, no qual aquele não está presente. Por fim, o terceiro passo define a pontuação silhueta para um objeto pela diferença das médias do segundo passo com o primeiro passo, dividido pela maior distância média encontrado nos passos anteriores.

A pontuação silhueta de cada objeto pode variar entre -1 e 1 e seu resultado pode ter três significados distintos (ROUSSEEUW, 1987). Quanto mais próximo de 1, melhor agrupado aquele objeto foi. Quanto mais próximo de 0 (zero), o objeto pode ser considerado semelhante com um objeto de um grupo distinto, desta forma há incertezas para qual dos dois agrupamentos aquele objeto deveria ser atribuído: esse caso é conhecido como "caso intermediário". Por fim, o pior resultado, próximo a -1, indica que o objeto possui maior semelhança com o agrupamento do outro objeto com o qual foi comparado, podendo assumir que o objeto foi atribuído ao seu agrupamento incorretamente.

Após pontuar todos os objetos, é possível obter a média da pontuação silhueta de um agrupamento, fazendo a média da pontuação de todos os objetos presentes no grupo (TAN et

al., 2005b). Dado a pontuação para cada objeto, quanto mais objetos de um agrupamento possuírem pontuação próxima a 1, melhor sera a média atribuída àquele grupo.

Preservação de vizinhança

A preservação de vizinhança, ou *neighborhood preservation* (PAULOVICH; MINGHIM, 2008), tem por objetivo quantificar a qualidade de uma projeção multidimensional de um conjunto de dados, verificando as relações de distância entre os objetos são mantidas (PAULOVICH, 2008). Para cada objeto multi-dimensional, sua vizinhança (*neighborhood*) é calculada observando os k vizinhos mais próximos no plano multidimensional e os k vizinhos mais próximos na projeção, sendo n a quantidade de documentos na coleção, k pode variar entre 1 até $n-1$, adotando-se tipicamente o valor de 30 como limite superior. Com isso é possível verificar se a proporção de vizinhança é preservada (*preservation*). Seu resultado pode variar entre 0 e 1. Logo, quanto mais próximo ou igual a 0, condiz que as vizinhanças de um objeto multidimensional e de sua projeção são muito ou totalmente distintos. Quanto mais próximo de 1, melhor é a preservação da vizinhança.

2.3. Mineração e visualização de programas

Para realizar a mineração de informações nos programas submetidos, é necessário decidir como serão extraídos as características – análise sintática, dinâmica e do código de escrita – e quais dados podem ser obtidos por meio da análise escolhida (Seção 2.3.1). Independente da dimensão obtida por meio da quantidade de informações extraídas, é necessário escolher como os dados obtidos serão representados para que seja possível realizar sua visualização.

2.3.1. Características de programas

Um programa de computador é um conjunto de instruções computacionais que visa solucionar um problema. A Figura 2.2 apresenta os passos necessários para que um programa exista. O primeiro passo consiste na identificação do problema a ser solucionado. Por exemplo, um fila indiana criada aleatoriamente, no qual não é possível visualizar todos os participantes presentes nessa fila. Em seguida, deve-se elaborar um algoritmo para que seja possível resolver esse problema. O algoritmo é constituído por uma sequência de passos que resolva algum problema, condizendo com o que deve ser feito. Por fim, realiza-se a implementação do algoritmo, definindo como as instruções devem ser executadas para que seja possível visualizar todos os participantes presentes na fila indiana.

Neste trabalho, consideramos características somente das implementações. A extração de características por meio das implementações pode ocorrer das seguintes formas: análise

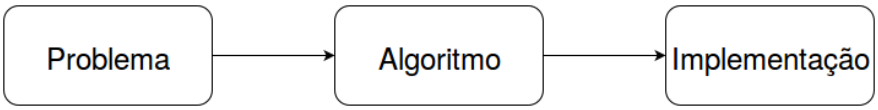

Figura 2.2. Representação da implementação de um problema

estática, análise do estilo de escrita e análise dinâmica. A análise estática ocorre por meio da observação do código-fonte, considerando apenas sua implementação, ou seja, não é necessária sua execução. Há diversas características que podem ser extraídas dessa análise. Há abordagens que extraem somente a Abstract Sintax Tree (AST), que pode ser gerada durante a análise sintática do compilador para representar o código-fonte em forma de árvore, armazenando símbolos não-terminais nos nós filhos e símbolos terminais nos nós folha, como representa a Figura 2.3 em uma declaração de condição. Outras abordagens extraem características como: a quantidade de linhas e atribuições da implementação, a complexidade ciclomática (MCCABE, 1976), quantidade de variáveis, operadores, operandos, laços de repetição e laços de repetição aninhados, por exemplo.

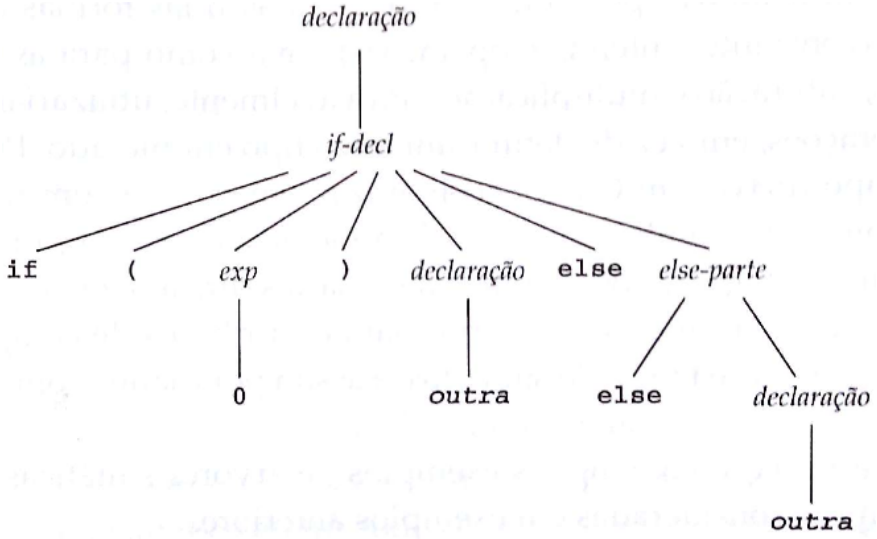

Figura 2.3. Representação de árvore de sintaxe abstrata (LOUDEN, 2004).

A análise do estilo de escrita é um tipo da análise estática. Entretanto, difere no fato das implementações estarem sintaticamente corretas. Nesse tipo de análise é considerado o estilo de escrita do programador, abrindo a possibilidade de coletar dados como: indentação, há mais que uma instrução e importação de bibliotecas por linha, ocorrência de espaços entre operando e operador, métodos separados por uma linha em branco, tamanho da instrução medido em caracteres e espaço entre operando e operador, por exemplo (Apêndice A). As Tabelas 2.1 e 2.2 exemplificam possíveis características extraídas em uma chamada de função, na qual é possível inserir ou não um espaço entre o parâmetro e os *tokens* abrir e fechar parêntese, como também a utilização de uma ou mais instruções por linha. Caso a primeira chamada de função e as duas primeiras instruções separadas por uma quebra de linha forem o padrão para o estilo de escrita, sua ocorrência não gerará nenhum aviso. Contudo, caso

ocorra as outras três chamadas de função, os quais possuem espaços em branco, ou mais de uma instrução por linha separados por ; (ponto e vírgula), é informado um tipo de erro.

Chamada de função
primo(7)
primo(7)
primo(7)
primo(7)

Tabela 2.1. Representação do estilo de escrita em uma chamada de função.

Instrução por linha
a = 5 * 3 primo(7)
a = 5 * 3; primo(7)

Tabela 2.2. Representação do estilo de escrita em instrução(ões) por linha.

A análise dinâmica do código-fonte consiste na observação da execução do programa, por meio do rastro de execução do programa (*trace*) e de teste de *software*. Analisando esse histórico, é possível verificar algumas características, como: em que momento foi realizada uma atribuição, chamada de função e recursão, qual bloco de código foi executado em uma declaração de condição, a quantidade de vezes que um laço de repetição foi executado e a saída no final da execução, entre outras situações e informações. O teste de software pode ser utilizado executando casos de teste nos quais é verificado se produção final do programa era o esperado e esta informação, se o caso de teste funcionou corretamente ou não, também pode ser uma característica do programa.

Além da possibilidade de extrair os dados citados anteriormente, é possível obter os dados de como foi realizado o processo técnico e social do desenvolvimento do programa. Para ambas abordagens, é necessário a utilização de um sistema de controle de versão para que se possa utilizar seus recursos. Um mecanismo para obter informações é em relação ao momento, data e hora em que o aluno realizou um *commit* de uma versão da sua implementação. Outro método é verificar se houve comunicações com outras pessoas durante o desenvolvimento do programa, por meio da interação em *issues* ou solicitações de *pull requests*.

2.4. Trabalhos relacionados

Os MOOCs disponibilizam diversos cursos por meio de plataformas de ensino online e para grande quantidade de pessoas. Por isso, desenvolveram ferramentas para auxiliar no ensino, como o uso de questionário relacionados com os vídeos (FASSBINDER et al., 2014), e na avaliação de questões de múltipla escolha (ALARIO-HOYOS et al., 2013). Entre os

cursos disponibilizados, há cursos relacionados a computação, no qual as atividades a serem submetidas requerem a implementação de um programa a questões objetivas. Contudo, não há recursos para auxiliar os professores a corrigirem tais implementações. Considerando a grande quantidade de usuários, pesquisadores buscam desenvolver ferramentas para auxiliar na correção, utilizando técnicas de mineração e visualização a fim de facilitar a correção de todas as implementações.

A fim de encontrar a semelhança entre os códigos, Yin et al. (2015) utilizaram a AST dos programas submetidos pelos alunos. Após a criação das árvores, é necessário o uso de métricas para verificar a similaridade entre as árvores. Desta forma, foi utilizada a Distância de Edição de Árvore (Tree Edit Distance (TED)) – ao comparar duas árvores, verificam-se quais são as movimentações (inserção, movimentação e remoção) necessárias para que as árvores fiquem iguais. Mais precisamente, foi selecionada a TED normalizada, atribuindo pesos maiores para os nós mais próximos da raiz (de menor altura) (ZHANG; SHASHA, 1989), quanto mais próximo do nó raiz, maior sua importância.

A TED normalizada realiza comparações nó a nó da raiz até as folhas, priorizando os nós mais próximos da raiz, atribuindo um peso maior a eles. A fim de evitar que pequenas diferenças na sintaxe influenciem diretamente na pontuação de similaridade, visto que a estrutura podem ser semelhantes, diferenciando-os em detalhes de baixo nível.

Os autores desse artigo utilizaram o algoritmo de agrupamento Ordering Points To Identify the Clustering Structure (OPTICS) (ANKERST et al., 1999) para agrupar os códigos semelhantes. Para visualizar os agrupamentos foi utilizado o t-Distributed Stochastic Neighbor Embedding (t-SNE) (MAATEN; HINTON, 2008) – técnica utilizada para reduzir dados de alta dimensionalidade para duas ou três dimensões preservando a estrutura local dos dados.

Na Figura 2.4 é possível observar cinco grupos distintos criados a partir da comparação das TEDs normalizadas destacados em cores diferentes. Tais cores foram obtidas, conforme o foco do trabalho para geração e fornecimento de dados, para representar os diversos valores obtidos pela métrica ABC (FITZPATRICK, 1997), no qual seu tamanho é computado contando o número de atribuições, a quantidade de ramos da árvore e as instruções de condição para um segmento do código-fonte. Contudo, a finalidade desse projeto é o fornecimento de dicas, o qual não é um tópico desta pesquisa. Cada ponto presente na visualização corresponde a uma implementação. Todas as implementações possuem a função *combine_anagrams* que possui a variável *words* como parâmetro (Figura 2.5). A implementação em vermelho obteve a menor métrica pela utilização de somente uma instrução. Os códigos-fontes roxo e laranja obtiveram pontuação semelhante, entretanto, esse obteve uma pontuação maior devido a um laço de repetição para cada chave presente na *hash* antes de terminar a função. Por fim, a implementação verde obteve a maior pontuação, visto que utilizou de mais instruções para

resolver o problema. Os pontos azuis não obtiveram similaridade suficiente para formar um agrupamento ou serem classificados em outro agrupamento, ou seja, *outliers*.

A extração de características por meio da AST é interessante, como também o uso da TED normalizada para verificar sua similaridade, devido ao seu baixo custo computacional. Entretanto, como o autor testou seu agrupamento somente com implementações para resolver um único problema, não sabemos se tal abordagem é eficiente quando houver implementações que buscam resolver diversos problemas, em razão da possibilidade dos códigos-fontes gerarem árvores parecidas ainda quando solucionam problemas distintos.

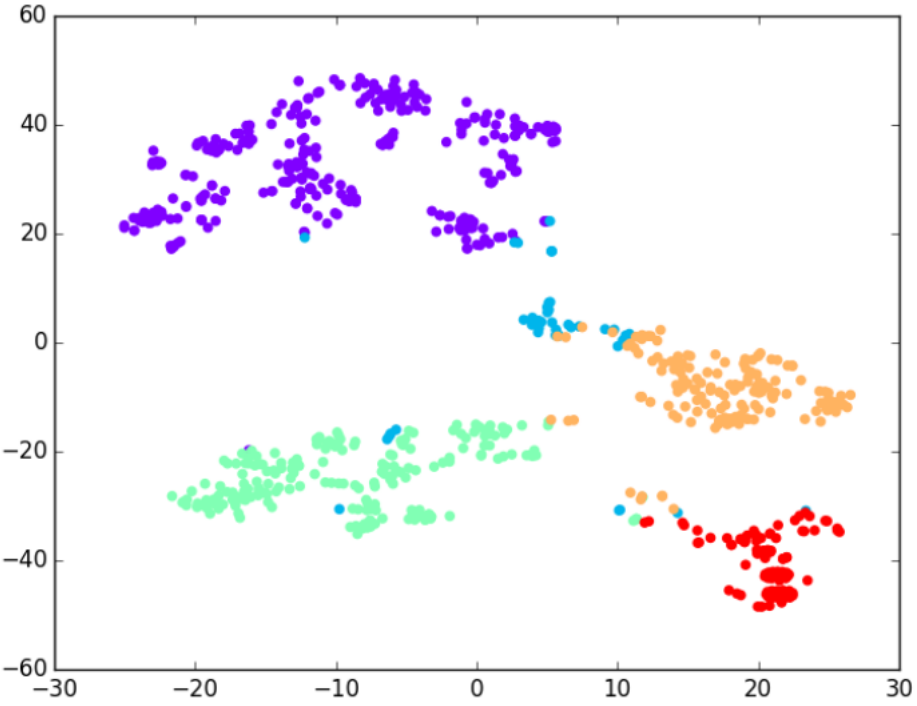

Figura 2.4. Visualização da técnica de projeção t-SNE (YIN et al., 2015)

```ruby
def combine_anagrams(words)
  words.group_by { |w|
    w.downcase.chars.sort.join }.values
end
```

```ruby
def combine_anagrams(words)
  result = []
  words.each do |w1|
    temp = []
    words.each do |w2|
      if (w1.downcase.chars.sort.join
          == w2.downcase.chars.sort.join)
        temp << w2
      end
    end
    if !result.include?(temp)
      result << temp
    end
  end
  return result
end
```

```ruby
def combine_anagrams(words)
  hash = Hash.new
  words.each do |w|
    key = w.downcase.chars.sort.join
    if hash.has_kay? key
      hash[key] += [w]
    else
      hash[key] = [w]
    end
  end
  hash.values
end
```

```ruby
def combine_anagrams(words)
  hash = Hash.new
  words.each do |w|
    key = w.downcase.chars.sort.join
    if hash.has_kay? key
      hash[key] << w
    else
      hash[key] = [w]
    end
  end
  result = []
  hash.each_key do |key|
    result << hash[key]
  end
end
```

Figura 2.5. Quatro implementações distintas que formaram os agrupamentos (YIN et al., 2015)

Glassman et al. (2014) propuseram um agrupamento hierárquico de dois níveis. No nível mais alto ocorre o agrupamento das soluções, utilizando o *K-means* com diversos valores para k, ao longo do plano de separação, considerando apenas características abstratas, como, por exemplo: posição da declaração de condicional em relação a declarações de laço de repetição (antes, dentro ou depois), profundidade de um laço de repetição (*loop*) aninhado, números de nós AST e declarações de retorno, *loops* e comparações.

Dentro de cada agrupamento de alto nível, é utilizado o *K-means* novamente, para produzir subagrupamentos destinados a capturar a dimensão generalizada, construções de linguagem de baixo nível e bibliotecas utilizadas. Os agrupamentos internos são formados por meio de 48 características concretas: operações aritméticas e lógicas, laços de repetição, funções de bibliotecas, declarações de atribuição, *loops*, condicional, número de variáveis e valores constantes, por exemplo.

A validação dos agrupamentos ocorreu por meio da comparação dos *clusters* criados pelo algoritmo de agrupamento com os que foram criados pelos professores. Para os professores foram entregues 50 códigos dos estudantes aleatoriamente distribuídos e notou-se que eles ignoraram características de baixo nível. Esses autores utilizaram a métrica Informação Mútua Ajustada (Adjusted Mutual Information (AMI)), cálculo probabilístico, para comparar cada agrupamento dos professores com os *clusters* gerado pelo *k-means* (MACQUEEN et al., 1967). Quando o valor de AMI é 0 (zero), quer dizer que os agrupamentos são independentes, entretanto, se for igual a 1, indica perfeita concordância entre os *clusters*. Quando k tinha um valor maior ou igual a 15, os agrupamentos concordaram com o agrupamento de cada professor, conforme medição do AMI.

Apesar da grande quantidade de dados extraídos das implementações, não houve nenhum alusão sobre como essas características interfeririam no cálculo de similaridade utilizada. Contudo, a abordagem de dividir as informações a serem coletadas em duas dimensões é relevante. Posto que as implementações com a mesma quantidade de laços de repetição, por exemplo, deveriam ser agrupadas facilitando a verificação se os alunos entenderam como fazer e utilizar tal instrução.

Taherkhani et al. (2012) desenvolveram uma ferramenta, denominada Aari, para classificar implementações baseado na utilização de conjuntos de treinamento e teste. Com isso, é necessário o conhecimento prévio de implementações dos problemas propostos para treinar o classificador e as características para reconhecer as submissões dos alunos. Os autores separaram as características em quatro categorias: características numéricas, características descritivas, características de algoritmos de ordenação e outras características.

A categoria de características numéricas extrai tudo que pode ser medido como inteiro e possui o seguinte conjunto de características: número de declarações de atribuição; número de linhas de código; complexidade McCabe; total de operadores; total de operandos; número de operadores único; número de operando único; total do número de operadores e

operandos; total do número de operadores e operandos únicos; número de variáveis; número de laços de repetição; número de laços aninhados e número de bloco.

A categoria de características descritivas possui: se um algoritmo é recursivo, se é uma recursão em cauda, papeis (*roles*) de variáveis e *arrays*. Essas características podem ser identificadas como booleano, indicando ausência ou existência das características correspondentes. Enquanto outras características possui informações sobre blocos e laços de repetição, informação do contador do *loop* e informações de dependência.

Após extrair as características, cada algoritmo pode ser representado pelo seu vetor de características. A ferramenta do estudo utiliza a técnica de árvore de decisão para classificar os algoritmos. É por meio dessa abordagem que os autores classificaram os programas enviados por um determinado grupo de alunos.

Para verificar a precisão do Aari, foi realizado uma categorização manual. Inicialmente foi realizado um agrupamento manual dos algoritmos de ordenação, diferenciando-os em duas etapas. A primeira rodada é referente a implementação do algoritmo sem o ensino prévio dos métodos de ordenação descritos anteriormente. Desta forma, foi pedido para que 112 alunos implementassem o método de ordenação que eles sabiam. Na segunda rodada, foi apresentado o funcionamento de cada algoritmo previamente e, após a apresentação, eles poderiam implementar qualquer outro algoritmo como também programar o mesmo da primeira etapa. Somente 80 alunos participaram da segunda rodada. Esses alunos também tinham participado da primeira etapa.

A Figura 2.6 apresenta o gráfico do agrupamento manual dos métodos de ordenação, *bubble sort*, *insertion sort*, *selection sort*, *merge sort* e *quick sort*, selecionados para verificar a precisão do Aari. O eixo x é representado pelos algoritmos de ordenação citados anteriormente, além da *Inneficiente variations*. Essa última representação é consequência de modificações realizadas pelos alunos na estrutura de qualquer algoritmo de ordenação. A categoria *Others* foi criada a partir das implementações de outros métodos de ordenação: *shell sort* e *heapsort*. O eixo y indica a quantidade de implementações reconhecidas. Para cada algoritmo de ordenação há duas colunas: a coluna da esquerda representa as soluções computacionais da primeira etapa, enquanto a coluna da direita demonstra as implementações da segunda rodada. É possível notar, após a apresentação dos algoritmos de ordenação na etapa 2, que: poucas implementações foram classificadas como *Inneficiente variations*; menos estudantes optaram em implementar o *bubble sort*, o *selection sort* e *Others*; e houve mais implementações do *merge sort* e do *quick sort*.

Após o agrupamento manual dos métodos de ordenação implementados pelos alunos, Taherkhani et al. (2012) realizaram o reconhecimento automático das implementações por meio da ferramenta Aari utilizando mais uma categoria de características, características de algoritmos de ordenação. Tal categoria considera as variáveis mais utilizadas, o uso de variáveis temporárias, se o algoritmo necessita de uma memória extra. Caso existam

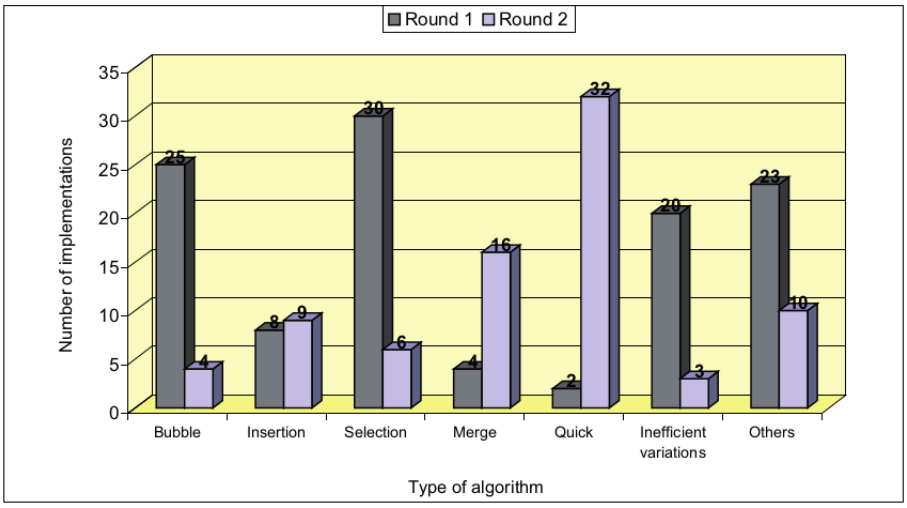

Figura 2.6. Agrupamento manual das implementações dos estudos estudantes na primeira e segunda etapa (TAHERKHANI et al., 2012).

dois *loops* aninhados, pode ocorrer dois tipos de características: o laço externo incrementa e o laço interno decrementa; e quando o laço interno é inicializado com o valor do laço externo. Inicialmente a ferramenta foi treinada para reconhecer os algoritmos de ordenação citados anteriormente. A Figura 2.7 apresenta um gráfico para comparar cada agrupamento manual realizado anteriormente com o reconhecimento automático da ferramenta. Possui as mesmas propriedades da Figura 2.6 com exceção das colunas: a coluna da esquerda referencia o agrupamento manual de cada algoritmo de ordenação da segunda etapa e a coluna da direita apresenta os algoritmos reconhecidos corretamente pelo Aari. Nota-se que todas as implementações dos métodos de ordenação *bubble sort*, *selection sort* e *quicksort* foram classificados corretamente. É possível verificar também que a ferramenta não foi capaz de reconhecer vários algoritmos como *others*, visto que ele não foi treinado para reconhecer tais algoritmos.

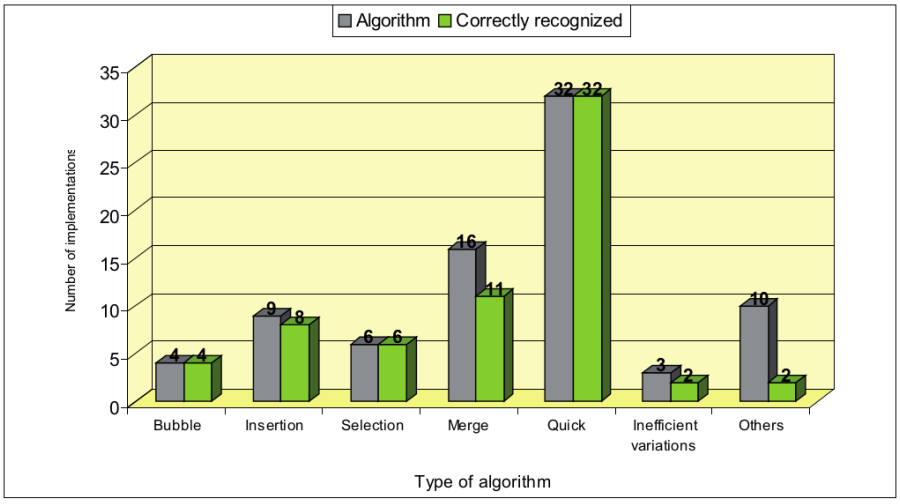

Figura 2.7. Comparação das implementações dos alunos dos *clusters* manual e automático da segunda etapa (TAHERKHANI et al., 2012).

A ferramenta mostrou-se capaz de identificar a maioria dos métodos de ordenação para quais foi treinada previamente. Tal abordagem torna-se interessante quando há um plano de ensino detalhando os problemas a serem resolvidos, possibilitando o treinamento da ferramenta. Entretanto, caso seja utilizado para classificar diversos problemas desconhecidos para o classificador, não há uma categorização prévia adequada. Desta forma, inviabilizaria sua utilização em MOOC se fosse utilizado para classificar os problemas de todos os cursos de programação.

Glassman et al. (2015) apresentam uma ferramenta para visualização de implementações que apresenta os agrupamentos de códigos-fontes formados, fornecendo as principais instruções utilizadas pelas implementações presentes em um determinado conjunto de submissões com a finalidade de auxiliar os professores que realizarão as correções.

Para verificar a similaridade das submissões a fim de realizar o agrupamento, são realizados os seguintes passos: formatar o código-fonte, executar casos de teste, extrair a sequência de variáveis, identificar variáveis em comum, renomear variáveis comuns e únicas para, então, realizar o agrupamento. Formatar o código-fonte consiste na reformatação de cada implementação: na remoção dos espaços entre os *tokens* (mantendo os espaços somente após as palavras reservadas) de comentários e de linhas em branco. Essas modificação no código-fonte, além de deixá-lo legível, também permite cada linha de código ser representada como uma *string* para que seja possível encontrá-la em outras soluções (GLASSMAN et al., 2015).

A segunda etapa do agrupamento consiste na execução do mesmo caso de teste para todas as implementações. A cada passo da execução, os nomes e valores de variáveis locais e globais, bem como o valor de retorno da função são gravados como se fosse um histórico de execução ou *trace*. A partir desse histórico, a ferramenta extrai a sequência de valores de todas as variáveis.

A partir do conhecimento da sequência de valores de cada variável, a ferramenta identifica quais são as variáveis comuns. Tais variáveis são reconhecidas a partir das suas sequências idênticas dado dois ou mais históricos de execução. As variáveis que só ocorrem uma vez no *trace* são as variáveis únicas. Após reconhecer as variáveis comuns e únicas, a ferramenta as renomeia para o nome da variável que ocorreu em mais históricos de execução.

Após todos esses passos, a normalização do código-fonte realizada em todas as implementações garante que o estilo de escrita do aluno não influenciará na verificação de similaridade. O agrupamento é realizado por meio da comparação estática entre conjuntos de linhas de diversas implementações. Desta forma, não utiliza uma técnica de agrupamento e de similaridade específica. Cada agrupamento, representado como conjunto de programas, pode possuir 1 ou mais programas cujo conjunto de linhas é idêntico.

Na Figura 2.8, que exibe a interface da ferramenta proposta, o OverCode, é possível notar a utilização do conjunto de programas para representar os agrupamentos. A primeira

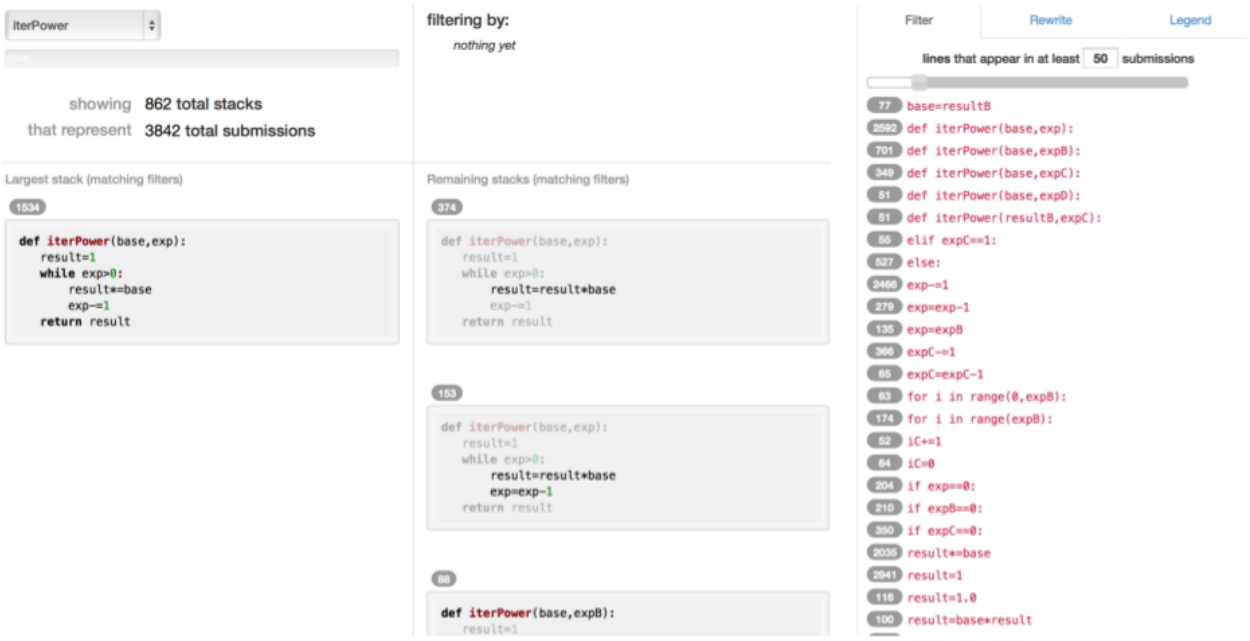

Figura 2.8. Interface da ferramenta OverCode (GLASSMAN et al., 2015).

coluna da esquerda exibe dois painéis. O primeiro painel apresenta o número de agrupamentos, representado pelos conjuntos de programas, e o número total de submissões, enquanto o segundo painel, mostra o maior conjunto de programas. A coluna central apresenta a opção de busca, filtrando por uma determinada palavra no quadro superior, e as conjunto de programas remanescentes no quadro inferior. A terceira coluna apresenta a frequência com que as linhas de códigos estão presentes nas soluções dos agrupamentos.

A comparação dos conjuntos de programas menores com a conjunto de programa maior ocorre entre a primeira e a segunda coluna dando ênfase nas linhas que estão implementadas diferentes. Com isso, é possível verificar como cada conjunto foi montado e a característica daquele agrupamento quando comparada com o conjunto maior. Como utiliza análise dinâmica, extraindo o histórico de execução para agrupar as submissões, é possível verificar o *trace* de uma variável ao longo de sua execução em um caso de teste a fim de auxiliar os usuários a entenderem a execução do algoritmo.

A Figura 2.9 apresenta o agrupamento, representado pela quantidade de conjunto de programas no eixo y e o total de submissões em uma conjunto de programa no eixo x, em relação a três problemas resolvidos em Python: o problema `iterPower` refere-se a implementação de uma função, que possui a base e o expoente como parâmetros, para calcular o exponencial com sucessivas multiplicações; o problema `hangman` possui um vetor de caracteres (*string*) e uma lista de caractere como parâmetros, na qual deve retornas todos os caracteres da *string* que não estão presentes na lista de caracteres; por fim, o problema `compDeriv` calcula a derivada de um polinomial, no qual os coeficientes estão presentes em uma lista. Esses problemas obtiveram 3875, 1118 e 1433 soluções corretas submetidas, respectivamente. Na devida ordem, a maior conjunto de programa de cada problema são

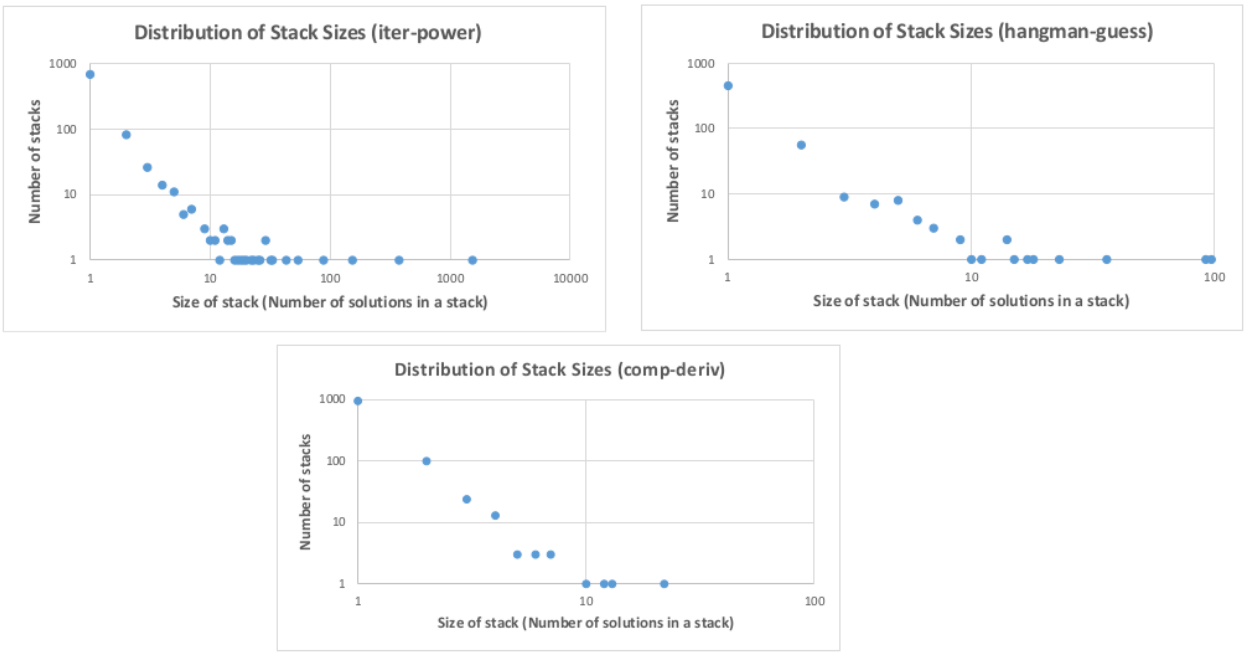

Figura 2.9. Representação do tamanho dos agrupamentos (conjunto de programas) para cada problema (GLASSMAN et al., 2015)

constituídas de 1534, 97 e 22 implementações e 684, 452 e 959 conjuntos de programas com apenas uma solução.

Por fim, os autores concluíram que a interface auxilia os professores a terem uma visão de alto nível das soluções (implementações), podendo compreender os erros e fornecer um *feedback* mais relevante, devido ao agrupamento das implementações. Isso diminui consideravelmente a quantidade de submissões a serem efetivamente corrigidas.

A análise dinâmica e os recursos realizados para que fosse montado a conjunto de programa mostrou-se eficiente, principalmente para o problema `iterPower`, no qual sua maior conjunto de programa obteve quase 40% das implementações corretas. Com isso, torna-se evidente o quanto a ferramenta pode auxiliar os professores a realizar as correções dos códigos-fontes. A etapa de padronização do código realizado durante a análise, no qual verifica-se as variáveis comuns e renomeia-as, contribui com a comparação bloco a bloco, devido a partes do bloco, onde ocorrem a utilização dessas variáveis, serem parecidas. Poderia ter sido verificado todos os problemas implementados em apenas um código-fonte para verificar qual seria o resultado dos agrupamentos e compará-los.

Wei e Wu (2015) apresentam uma ferramenta para auxiliar na correção das submissões do MOOC de forma a agrupar pedaços (*chuncks*) de códigos fontes semelhantes, agrupá-los conforme sua similaridade e alocar cada conjunto de implementações ao estudante com conhecimento suficiente para revisá-los.

O particionamento do código tem por objetivo torná-lo legível e fácil de compreender para a correção. A ferramenta realiza o particionamento por funções. Também foi necessário normalizar cada submissão a fim de encontrar o estilo de escrita, visto que até mesmo o

nome da variável pode alterar o estilo de escrita. A ferramenta desenvolvida por Wei e Wu (2015) normaliza o código por meio de três regras: remoção de espaços, linhas em branco e comentários; exclusão de palavras reservadas da linguagem de programação, identificadores predefinidos e nomes de funções de bibliotecas; e substituição dos identificadores de variáveis do usuário por um símbolo especial.

Com isso foi calculado um valor *hash* para todas as *k-substring* de um código-fonte, sendo k a quantidade de *substrings* da implementação, a fim de verificar a similaridade do estilo de escrita *token* a *token* é utilizado o algoritmo de `winnowing` (SCHLEIMER et al., 2003) para escolher o menor subconjunto do estilo de escrita a fim de realizar as comparações por meio do coeficiente de similaridade de *Jaccard* (JACCARD, 1901).

E com a finalidade de identificar a dificuldade da implementação, foi utilizado a distância Euclidiana para comparar as características extraídas – quantidade de métodos invocados e laços de repetição aninhados – e o K Nearest Neighbor (k-NN) para identificar o nível de dificuldade de revisão do *chunck*.

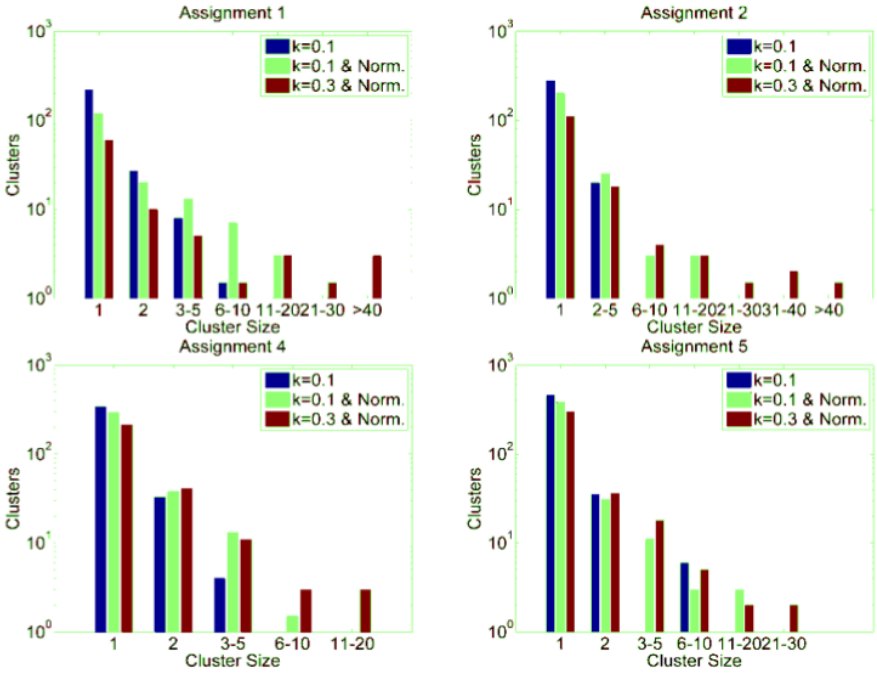

Figura 2.10. Distribuição dos agrupamentos (WEI; WU, 2015).

A Figura 2.10 apresenta a distribuição dos agrupamentos em quatro problemas distintos. Para todos os gráficos, a abcissa é referente ao tamanho dos pedaços, enquanto a ordenada representa o número de agrupamentos. É possível notar três tipos de agrupamento. A barra horizontal azul representa o agrupamento sem normalização. Enquanto as barras verde e vermelho possuem o valor de k diferente, entretanto ambas estão normalizadas. As tarefas 1 e 2 possuem apenas uma função para resolver um pequeno problema e apresentou bons agrupamentos, nos quais possuíam mais de 40 implementações em um mesmo grupo quando k era igual a 0,3. As tarefas 4 e 5 possuem implementações com várias funções, por esse

motivo, os agrupamentos não foram satisfeitos, principalmente quando não normalizados. Por fim, pode-se notar que a distribuição dos *chuncks* sem normalização de código é inadequado, visto que vários pedaços em todas as tarefas estavam sozinhos, e o maior agrupamento possuía entre 6 a 10 funções.

É possível notar que a abordagem dos códigos-fontes compostas de diversas funções (tarefas 4 e 5) não obteve sucesso, pois, independente da tarefa selecionada, nota-se que a maioria dos *clusters* formados possuíam apenas um código-fonte. Entretanto, a submissão de implementações contendo apenas uma função (tarefas 1 e 2) para resolver o problema torna-se interessante, devido à realização de alguns agrupamentos com mais de 40 implementações semelhantes. Em um ambiente controlado, no qual todos os alunos irão submeter diversas soluções com somente uma função, é interessante a utilização da ferramenta. Caso contrário, seu uso não interferirá significativamente no auxílio a correções de códigos-fontes.

A Tabela 2.3 apresenta as principais características de cada trabalho relacionado importantes para o desenvolvimento desse projeto, possibilitando a comparação dos tipos de análises utilizados, quais características foram extraídas a partir dessas análises e qual abordagem obteve melhor desempenho. Para todos os trabalhos relacionados, os dados de entrada são implementações.

Artigos	**Característica extraídas dos dados de entrada**	**Algoritmo para cálculo de distância ou similaridade**	**Algoritmo de clusterização / Classificador**	**Avaliação**	**Conclusões**
Yin et al. (2015), Moghadam et al. (2015), Choudhury et al. (2016)	Árvore de sintaxe abstrata	Distância de Edição de TED normalizada que utiliza estrutura *top-down* e pontuação silhueta	OPTICS	Agrupamento de algoritmos baseado nas soluções de um problema. Em cada *cluster*, identifica as diferenças nas implementações de uma abordagem particular	TED normalizado demonstra conjuntos mais estáveis e menos discrepantes
Glassman et al. (2014)	12 características de alto nível dos elementos da linguagem e instruções relacionados com laços de repetição. 48 características de baixo nível envolvendo operações primitivas.	Informação Mútua Ajustada	K-means	Utilizando a métrica AMI, compara os agrupamento automatizados com os manuais. 0 indica agrupamentos puramente independentes e 1, perfeita concordância entre os agrupamentos.	Para k maior ou igual a 15, encontrou-se alta concordância entre os *cluster* do k-means e do professor
Taherkhani et al. (2012)	Características extraídas com valores inteiros, específicas de métodos de ordenação, do contador de *loops* e dependência de informação e para reconhecer padrões para classificação.	Vetor de características,calculada a partir das características extraídas das implementações	C4.5 (árvore de decisão)	Comparação da ferramenta com a classificação manual dos professores.	Bons resultados para algoritmos de ordenação que foram utilizados na fase de treinamento. E taxa de acerto considerável baseado em um possível erro do professor.
Glassman et al. (2015), Glassman (2016, 2014), Glassman et al. (2013), Glassman (2013)	Rastro do programa e blocos de código.	Diferença entre o tamanho dos conjuntos de linhas	Comparação entre conjuntos de linhas de código	Grande quantidade de conjunto de programas com poucos blocos de código-fonte e poucas conjunto de programas com grande quantidade de blocos implementados	Possibilidade de retornar um *feedback* mais preciso para cada grupo e facilita a observação da solução do problema.
Wei e Wu (2015)	Normaliza o código e considera apenas o estilo de escrita do estudante	Coeficiente de similaridade de Jaccard	Algoritmo *Winnowing*	Classificação de *workload*: distância Euclidiana e k-NN	A agrupamento por pedaços de código aumentou sua eficiência.

Tabela 2.3. Principais informações dos trabalhos relacionados

Ambas as abordagens que utilizaram pedaços das implementações para realizar o agrupamento (GLASSMAN et al., 2015; WEI; WU, 2015) mostraram-se vantajosos pela quantidade de conjuntos de linhas. A renomeação de variáveis comuns, obtidas por meio de casos de teste, também é interessante devido ao fato de igualar as instruções que utilizam tais variáveis a fim de realizar as comparações (GLASSMAN et al., 2015). Apesar da utilização das ASTs somente em um único problema (YIN et al., 2015), tal abordagem obteve agrupamentos significativos a fim de auxiliar os professores para realizar as correções.

Contudo, a utilização de um algoritmo de classificação para identificar implementações semelhantes (TAHERKHANI et al., 2012) torna inviável a utilização nessa pesquisa, devido a necessidade de um grande conjuntos de implementações para realizar o conjunto de treinamento e teste a fim de treinar o classificador. E a abordagem utilizando dois níveis hierárquicos para agrupamento (GLASSMAN et al., 2014) possui um conjunto de características vasto, mas a maioria das características pode ser descartada no momento do agrupamento manual realizado pelos professores. Isso pode ter ocorrido pelo fato dos códigos-fontes utilizados na pesquisa não representarem todas as soluções possíveis de um problema. Não é possível afirmar isso, visto que não possuímos as implementações.

Capítulo 3

Métodos e ferramentas

Neste capítulo é apresentado o método e as ferramentas desenvolvidas e alteradas para gerar nossos resultados. O método consiste das seguintes etapas: definição da linguagem de programação e das características a serem extraídas; códigos-fontes; extração de características; cálculo da matriz de similaridade; e projeção e visualização. Em seguida, são especificadas as configurações dessas etapas para o contexto do presente trabalho. Por fim, são apresentadas as ferramentas utilizadas para extrair as características e visualizar as projeções, bem como as modificações necessárias de cada ferramenta.

3.1. Método

Após a revisão da literatura, verificamos que o nosso estudo pode ser dividido em algumas etapas. Conforme apresentado na Figura 3.1, primeiro é necessário escolher qual linguagem de programação será utilizada. 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. Por exemplo: `C`, `Java` ou `Python`. Deve ser escolhida conforme a demanda da sua utilização, pela disponibilidade de cursos de programação ofertado em MOOCs ou pelo conhecimento de ferramentas que possam extrair características de programas escritos nessa linguagem, como o `cpplint` para a linguagem `C`, `Checkstyle` para `Java` e `pep8` para `Python`.

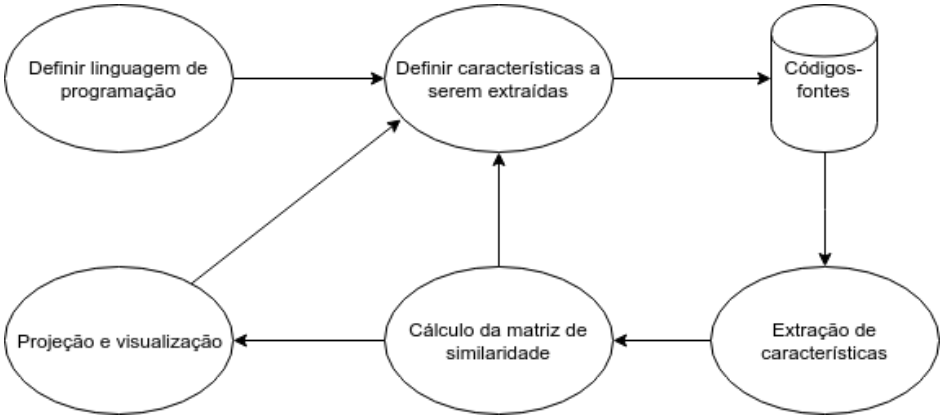

Figura 3.1. Fluxograma das etapas necessárias para a realização do projeto.

A etapa seguinte trata da **definição das características a serem extraídas** e será realizada com base nos trabalhos relacionados. Verificamos que é possível utilizar características originadas de um único tipo de análise (estática, dinâmica ou do estilo de escrita), bem como associar características de análises distintas para obter um melhor resultado.

A etapa **códigos-fontes** refere-se à construção da base de dados que será utilizada no estudo. Tal base tem que possuir uma quantidade considerável de submissões de códigos-fontes, devido a quantidade de usuários que utilizam o MOOC, para que seja possível avaliar o desenvolvimento do projeto com uma quantidade condizente com o número de usuários.

A **extração de características** é a fase na qual escolhe-se uma ferramenta disponibilizada livremente pela Internet que extraia as características selecionadas previamente, visto que não desejamos criar uma ferramenta para esse fim. Tal estágio é responsável pela adaptação das ferramentas selecionadas para que seja possível obter as características e salvá-las em algum tipo de arquivo, como o formato `XML` e o `CSV`, por exemplo.

Na etapa de **cálculo da matriz de similaridade** analisam-se os dados obtidos a fim de verificar a similaridade entre as diferentes instâncias de submissões com relação às características extraídas. Isso permite a utilização de técnicas para verificar a similaridade entre os códigos-fontes, como a similaridade do cosseno.

Por fim, **projeção e visualização** refere-se ao emprego de técnicas de projeção multidimensional para que seja possível realizar um mapeamento de um espaço multidimensional (alta dimensionalidade) para um espaço de baixa dimensionalidade (bidimensional ou tridimensional), buscando a menor perda de informação possível, como a manutenção das relações de similaridade do espaço multidimensional no espaço de baixa dimensionalidade. Neste estudo uma dimensão é referente a uma característica extraída da implementação. Portanto, a quantidade de dimensões nos dados é igual a quantidade de características extraídas. Com isso, é necessário selecionar uma técnica para diminuir um espaço n-dimensional para duas ou três dimensões a fim de realizar a visualização exploratória para identificar padrões nos dados e possíveis agrupamentos formados.

No caso das características extraídas não serem relevantes o suficiente, a visualização não será capaz de produzir modelos gráficos significativos que evidenciem os padrões existentes nos dados, sendo necessário voltar à segunda etapa para definição de outras características a serem extraídas.

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 *ScienceView* para o professor. Por isso, temos duas questões de pesquisa (QP):

- **QP**$_1$: o algoritmo de agrupamento produz grupos de códigos-fontes com boa qualidade?
- **QP**$_2$: a utilização de subsídios de avaliação reduz o tempo de correção de todas as submissões?

O subsídio de avaliação foi uma adaptação da ferramenta `ScienceView` (ALENCAR et al., 2012) que, originalmente, verifica as mudanças nas relações de artigos científicos com o decorrer do tempo, utilizando técnicas de mineração visual embasadas em uma técnica de projeção multidimensional chamada Time-based Least Square Projection (ALENCAR, 2013).

Para validar o tempo gasto para todas as correções, foram criado dois grupos de professores. Um grupo utilizou a técnica proposta nessa pesquisa para realizar a avaliação das implementações, enquanto o outro grupo avaliou as mesmas submissões manualmente. Além disso, avaliamos o *feedback* da ferramenta para os professores por meio de *survey* e questionário para cada professor.

3.2. Considerações para avaliações

Considerando os estágios estabelecidos no método descrito na Seção anterior (3.1), foram definidos requisitos e elementos a serem empregados na avaliação dos códigos-fontes neste trabalho, apresentados nas seções seguintes.

3.2.1. Definição da linguagem de programação

Foi definido 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. Como também o conhecimento prévio de ferramentas que possam auxiliar na verificação e extração de características.

3.2.2. Definição das características a serem extraídas

Com base em todos os trabalhos relacionados, podemos perceber que os três tipos de análise – estática, estilo de escrita e dinâmica – obtiveram bons resultados. Isso nos possibilitou decidir qual tipo de análise utilizar. Desta forma, optamos pelo tipo de análise

por meio da disponibilidade de ferramentas para verificar tais tipos de características e que possuíssem código aberto para realizar adaptações a fim de alcançar nosso objetivo. Desde que posteriormente ocorra uma seleção das características mais relevantes para avaliação, a escolha natural das características, apresentadas a seguir, não afeta a qualidade das projeções.A seleção de características relevantes faz parte dos trabalhos futuros.

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. Não utilizaremos a análise dinâmica pela falta de conhecimento de ferramentas que nos forneça tal característica e pelo pouco tempo hábil para implementação de uma ferramenta para realizar esse tipo de análise e validação dessa ferramenta.

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 `PEP 8` (ROSSUM et al., 2001a).

3.2.3. Códigos-fontes

Essa etapa refere-se a construção da 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 somente implementações desenvolvidas em `Python` de um curso introdutório à Computação e Programação.

3.2.4. Extração de características

Obtivemos as informações necessárias por meio da análise estática e do estilo de escrita com o auxílio das ferramentas `pep8` (ROSSUM et al., 2001b) e `mccabe` (MCCABE, 1976) que funcionam como *plugins* para o `flake8`. Extraímos da análise estática, além do estilo de escrita, a quantidade de linhas do código-fonte e a complexidade ciclomática de cada *plugin*, respectivamente. Contudo, o foco do `pep8` é verificar se o estilo de escrita PEP 8 (ROSSUM et al., 2001a) está sendo praticado corretamente. Por esse motivo, extraímos as características relacionadas a: indentação (Tabela A.1); espaços em branco (Tabela A.2); linhas em branco (Tabela A.3); declaração de importação de bibliotecas (Tabela A.4); tamanho da linha (Tabela A.5); quantidade de instruções por linha e formas de instrução (Tabela A.6);

por fim, verificação de sintaxe e geração de *tokens* (Tabela A.7). Todas essas tabelas estão presentes no Apêndice A.

A ferramenta `mccabe` (CORDASCO, 2013) fornece informações sobre a complexidade ciclomática (MCCABE, 1976) de cada função ou método implementado. Essa complexidade é referente às estruturas de decisões implementadas (MCCABE, 1976), ou seja, seu cálculo é realizado com base na utilização de `if/else` e laços de repetição.

3.2.5. Cálculo da matriz de similaridade

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. Visto que, durante a extração de características obtemos uma representação matricial dos dados. Essa matriz é tipicamente esparsa – possui grande quantidade de 0 (zeros). A ocorrência da grande quantidade de 0 (zeros) ocorre pela busca dos programadores em implementar suas soluções computacionais corretamente. Nesse tipo de situação, a similaridade do cosseno é mais utilizada por não favorecer comparações entre 0 (zeros) (ALENCAR, 2013). Por isso, ela faz com que implementações para problemas distintos não sejam consideradas semelhantes devido a quantidade de 0 (zeros) nas mesmas características. A grande quantidade de 0 (zeros) pode ser observada na Figura 3.3.

3.2.6. Projeção e visualização

Para possibilitar a visualização dos padrões nos dados utilizamos a ferramenta *ScienceView* (ALENCAR et al., 2012), que utiliza a técnica de projeção multidimensional dinâmica (ALENCAR, 2013) e o algoritmo de agrupamento DBSCAN (ESTER et al., 1996) para encontrar possíveis agrupamentos na projeção. A T-LSP (ALENCAR, 2013) produz uma sequência temporal de mapas conforme a mudança nas relações de similaridade dos dados. Com isso, poderemos verificar se o aluno está progredindo no curso, visto que a técnica de projeção permitirá visualizar projeções relativas a implementações ao longo de várias semanas ou meses, por exemplo. Desta forma, podemos obter sequências temporais de mapas baseado na similaridade das implementações de um único aluno.

É possível evidenciar os padrões formados na projeção de duas formas: executa-se o algoritmo de agrupamento para cada uma das projeções na sequência, exibindo as características cujos valores provocaram a formação daquele grupo na projeção; ou selecionando um conjunto de instâncias próximas numa dada projeção na sequência, e exibindo características cujos valores foram similares dados as instâncias presentes na seleção.

Salienta-se que, para entender os agrupamentos, é necessário saber o que cada erro significa a fim de verificar se todas as implementações contidas no grupo possuem esses erros e, então, avaliá-las.

Após gerar a visualização, é possível identificar os grupos formados ao longo da projeção ativando o que a ferramenta chama de extração de tópicos. Com isso, é possível verificar as características semelhantes de cada grupo, apresentado visualmente por uma região cinzenta com uma anotação sobre as característica semelhantes.

3.3. Ferramentas

Definido o método da pesquisa e as configurações adotadas neste trabalho, nesta seção apresentamos todas as ferramentas e modificações necessárias para que fosse possível realizá-la. A Seção 3.3.1 apresenta as ferramentas e suas modificações para que seja possível criar o arquivo de entrada da *ScienceView*. A Seção 3.3.2 descreve a ferramenta utilizada para visualizar as projeções e agrupamentos, as modificações necessárias para leitura do arquivo `CSV` e um tutorial sobre como utilizar a ferramenta a fim de aprimorar a correção das submissões.

3.3.1. ScienceView-Python

A *ScienceView-Python* é o conjunto de ferramentas e modificações necessárias para criarmos um arquivo com as características extraídas de cada código-fonte a fim de ser utilizada na *ScienceView* (Seção 3.3.2). Essa ferramenta analisa a pasta contendo os códigos-fontes passada por parâmetro recursivamente, analisando todos os arquivos com extensão `py`.

Essa ferramenta é composta pelo `flake8` (CORDASCO, 2010) e os *plugins* `pep8` (ROSSUM et al., 2001b) e `mccabe` (CORDASCO, 2013) pois, além de atender ao tipo de características a serem extraídas, todas eles possuem código aberto. Contudo, antes de extrair as características das implementações, foi necessário definir qual o formato de arquivo seria adequado para utilizar como saída da *ScienceView-Python* e como arquivo de entrada da *ScienceView*.

É importante selecionar um formato de arquivo para padronizar a escrita e leitura do arquivo, além de possibilitar a adaptação da *ScienceView* para exibir a similidade de implementações desenvolvidas em outras linguagens. O formato CSV (valores separados por vírgulas ou algum outro símbolo distinto) é genérico o suficiente para este propósito e também facilita sua edição em softwares de planilhas, caso queria remover uma coluna, por exemplo.

Para extrair as características das implementações, foi necessário a modificação das ferramentas `pep8` e `flake8`. No `pep8`, alteramos o método `check_files()` que é chamado apenas uma vez e é responsável por verificar cada código-fonte especificado no momento da execução por linha de comando. Tal modificação consiste na criação do arquivo no formato

`CSV` com o cabeçalho: nome do arquivo, total de linhas do código-fonte, quantidade de erros do estilo de escrita e cada erro do `PEP 8`.

A Figura 3.2 apresenta o formato da impressão de cada característica do `flake8`. Notamos que era possível obter as características no momento em que fosse realizado cada `print` dos dados, visto que, todas as linhas possuem o tipo do erro, E302 ou a complexidade ciclomática C901, por exemplo. Desta forma, alteramos o método `get_file_result()` da classe `reporter` do `flake8`. Criamos um dicionário contendo todos os erros do `PEP 8` e do `mccabe` como chave. A cada impressão verificamos qual é o tipo do erro e incrementamos o valor dessa chave.

```
base-de-dados/tr1/3309055/corrigido-v1.py:36:1: C901 'fibonacci_poo' is too complex (2)
base-de-dados/tr1/3309055/corrigido-v1.py:36:1: E302 expected 2 blank lines, found 1
base-de-dados/tr1/3309055/corrigido-v1.py:37:1: E101 indentation contains mixed spaces and tabs
base-de-dados/tr1/3309055/corrigido-v1.py:42:1: C901 'fibonacci' is too complex (2)
base-de-dados/tr1/3309055/corrigido-v1.py:42:1: E302 expected 2 blank lines, found 1
base-de-dados/tr1/3309055/corrigido-v1.py:48:1: C901 'letras' is too complex (2)
base-de-dados/tr1/3309055/corrigido-v1.py:48:1: E302 expected 2 blank lines, found 1
base-de-dados/tr1/3309055/corrigido-v1.py:54:1: E302 expected 2 blank lines, found 1
base-de-dados/tr1/3309055/corrigido-v1.py:54:15: E231 missing whitespace after ','
base-de-dados/tr1/3309055/corrigido-v1.py:63:1: C901 'contar' is too complex (3)
base-de-dados/tr1/3309055/corrigido-v1.py:63:1: E302 expected 2 blank lines, found 1
```

Figura 3.2. Representação das características analisadas pelo `flake8`.

A Figura 3.3 apresenta o arquivo gerado pelas ferramentas após a execução das análises de cada código-fonte presente no repositório analisado. As três primeiras colunas são: o nome do arquivo, a quantidade de linhas implementadas e o total de erros do estilo de escrita PEP 8 (ROSSUM et al., 2001a), respectivamente. As demais colunas são referentes a cada erro do PEP 8, com a última coluna representando a complexidade ciclomática calculada pela ferramenta `mccabe` (CORDASCO, 2013).

	filename	lines	errorE	E101	E111	E112	E113	E114	E115	E116	E121	E122	E123	E124	E125	E126	E127	E128	E129	E131	E133	E201	E202	E203	E211	E221	E222	E223
2	base-de-dados/tr4/3309138/corrigido-v4.py	250	113	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20	0	0	0
3	base-de-dados/tr4/3344524/corrigido-v4.py	189	65	2	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	23	0	0	0
4	base-de-dados/tr4/3560495/corrigido-v4.py	215	84	57	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
5	base-de-dados/tr4/3479469/corrigido-v4.py	195	20	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	2	0	0	0
6	base-de-dados/tr4/3455855/corrigido-v4.py	145	81	11	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
7	base-de-dados/tr4/3530035/corrigido-v4.py	150	19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	base-de-dados/tr4/3458121/corrigido-v4.py	45	16	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	base-de-dados/tr4/3529892/corrigido-v4.py	210	124	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	17	2	1	0	0	0
10	base-de-dados/tr4/3167972/corrigido-v4.py	147	79	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0
11	base-de-dados/tr4/3456310/corrigido-v4.py	205	113	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	1	0	0	0	0	0
12	base-de-dados/tr4/3457749/corrigido-v4.py	208	66	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	25	0	0	0
13	base-de-dados/tr4/3460142/corrigido-v4.py	206	72	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	26	0
14	base-de-dados/tr4/3458604/corrigido-v4.py	207	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
15	base-de-dados/tr4/3529760/corrigido-v4.py	156	42	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
16	base-de-dados/tr4/3530185/corrigido-v4.py	92	27	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

Figura 3.3. Arquivo no formato `CSV` gerado após as adaptações inseridas nas ferramentas `flake8` e `pep8`.

3.3.2. ScienceView

A ferramenta *ScienceView* (ALENCAR, 2013) foi desenvolvida para criar mapas de documentos temporais que exibam as mudanças nas as relações entre artigos científicos, utilizando a projeção multidimensional dinâmica T-LSP a partir de artigos nos formatos *ISI*, *Endnote Export Format* ou *BibTeX*.

A primeira adaptação na ferramenta condiz com a inserção do novo formato `CSV`. Foi necessário criar um filtro para a extensão `.csv` a fim que só aparecesse os arquivos desse formato no momento de selecionar o arquivo. Em seguida, foi preciso realizar a leitura desses dados contidos no arquivo. Assim, foi necessário encontrar uma biblioteca disponível no Maven para realizar a leitura do arquivo para obtermos as características presentes no cabeçalho, nome do arquivo e a quantidade de cada erro desse arquivo para pode inserir no banco de dados da ferramenta. Inserimos novos comandos `SQL` para guardar todas as informações da leitura no banco de dados. Toda matriz esparsa obtida na leitura é salva no banco de dados para que seja utilizada como base para o cálculo da matriz de similaridade.

Originalmente, a ferramenta suportava apenas números inteiros em sua representação interna. No entanto, alguns dos valores importados do formato `CSV` podem ser números reais. Desta forma, foram alteradas as formas de representação e cálculos para utilizar números em ponto flutuante de precisão dupla.

Modificamos também a ferramenta para que, ao clicar duas vezes em um marcador visual na projeção (que representa um código fonte), seja mostrado o código-fonte referente ao marcador (programa sob avaliação) e seu vetor de características (com e sem normalização).

A fim de criar uma nova projeção multidimensional dinâmica, a ferramenta guia o usuário para: inserir um novo conjunto de dados ou utilizar uma coleção presente no banco de dados; definir parâmetros sobre o pré-processamentos dos dados; decidir qual cálculo de similaridade será utilizado; e determinar os parâmetros referente a T-LSP.

A Figura 3.4 apresenta os painéis para selecionar uma coleção existente e inserir uma nova coleção na base de dados. Para acrescer uma nova coleção na base, clique no botão *Select file*, altere o filtro do tipo do arquivo, selecionando o arquivo `CSV (*.csv)`. Depois clique no botão *Select path* e selecione o caminho que está as implementações. Esse caminho deve conter a primeira pasta do arquivo descrito no arquivo `CSV`. Clique em *Load* para inserir a coleção na base de dados e selecione a coleção que acabou de ser inserida, clicando em *Select* no painel *Selected collection*.

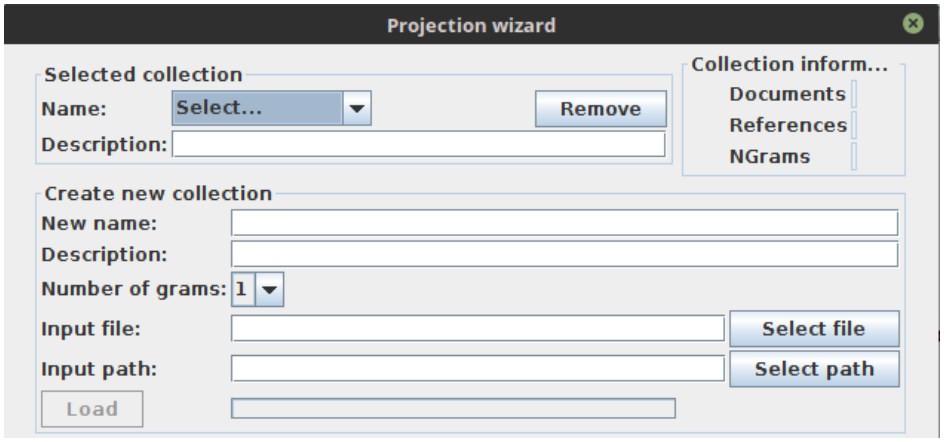

Figura 3.4. Interface para selecionar coleção e inserir nova coleção na base de dados

A Figura 3.5 apresenta o painel para seleção do cálculo de normalização descrito como *Normalization*. É possível selecionar o cálculo que deseja utilizar ao clicar em `NONE`. Será apresentado todos os cálculos de normalização disponíveis.

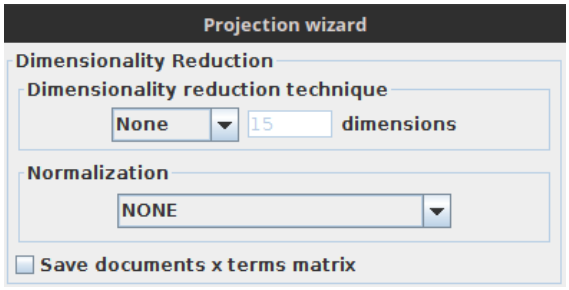

Figura 3.5. Interface para selecionar o cálculo de normalização.

A Figura 3.6 apresenta o painel para escolha do cálculo de similaridade descrito como *Choose the Distance Type*. É possível selecionar o cálculo de similaridade que deseja utilizar, clicando em *Cosine-based*. Então, sera apresentado todas os cálculos de similaridade disponíveis na ferramenta.

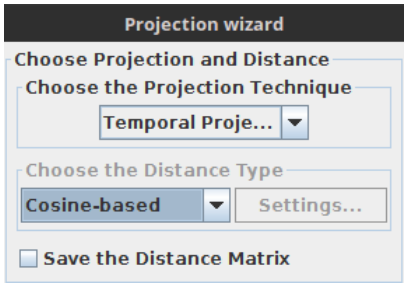

Figura 3.6. Interface para selecionar o cálculo de similaridade.

A Figura 3.7 apresenta a interface inicial para verificar a visualização da projeção. É possível visualizar o desenvolvimento da projeção de duas formas: clicando no botão com o ícone *Play*; ou visualizar cada projeção na sequência de projeções interagindo com a barra de progressão da visualização, localizada na parte inferior da interface ao lado dos botões, clicando no botão com o ícone *Forward*. Ao final, dado que a sequência de projeções é cumulativa, será apresentado a projeção completa, conforme a Figura 4.3, na qual cada ponto corresponde a uma implementação e, ao clicar duas vezes, é apresentado seu código-fonte, bem como a quantidade de erros e seus valores normalizados.

Também é possível ativar a extração de tópicos ao decorrer das projeções. Desta forma é possível visualizar os agrupamentos encontrados pela *ScienceView*, bem como os tópicos ou características de cada grupo. A Figura 3.8 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 relevantes àquele agrupamento. A ferramenta também realiza o rastreamento dos grupos simultaneamente. Consequentemente,

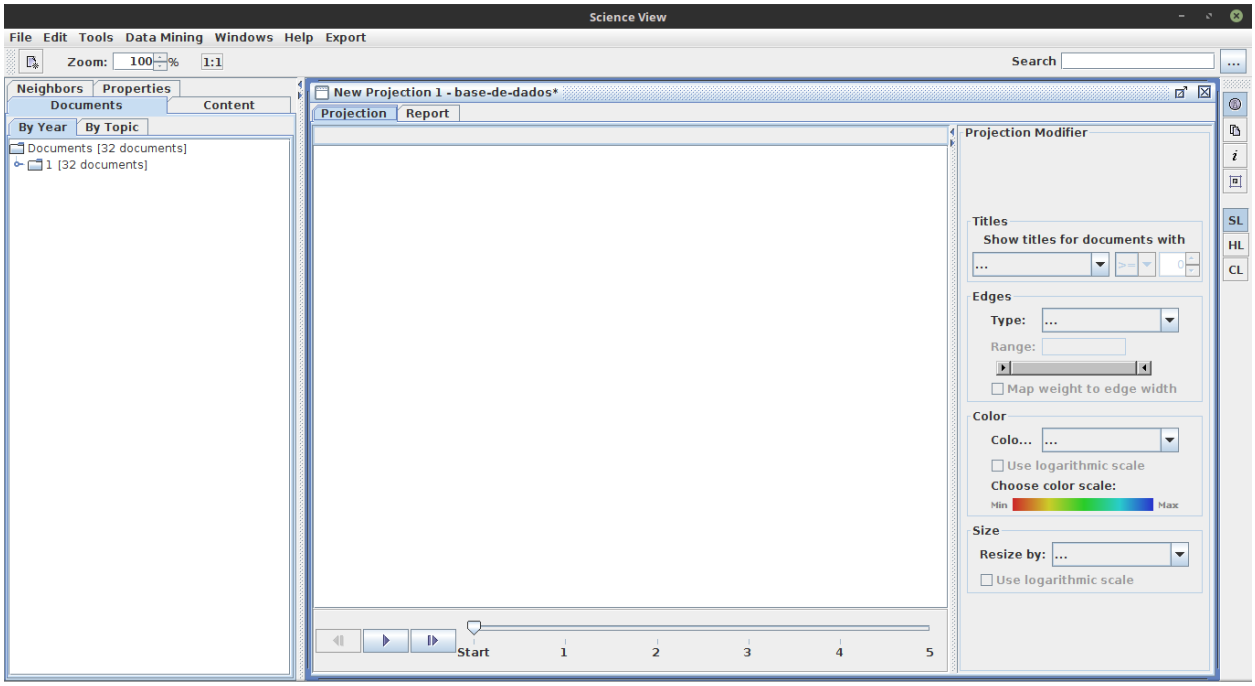

Figura 3.7. Interface inicial da visualização.

além de gerar os grupos para cada projeção na sequência, a *ScienceView* também rastreia esses grupos ao longo do tempo. A extração de tópicos e o rastreamento podem ser ativados pelo menu superior, clicando em: *Data Mining*, *Topic Extraction and Tracking* e *Run*.

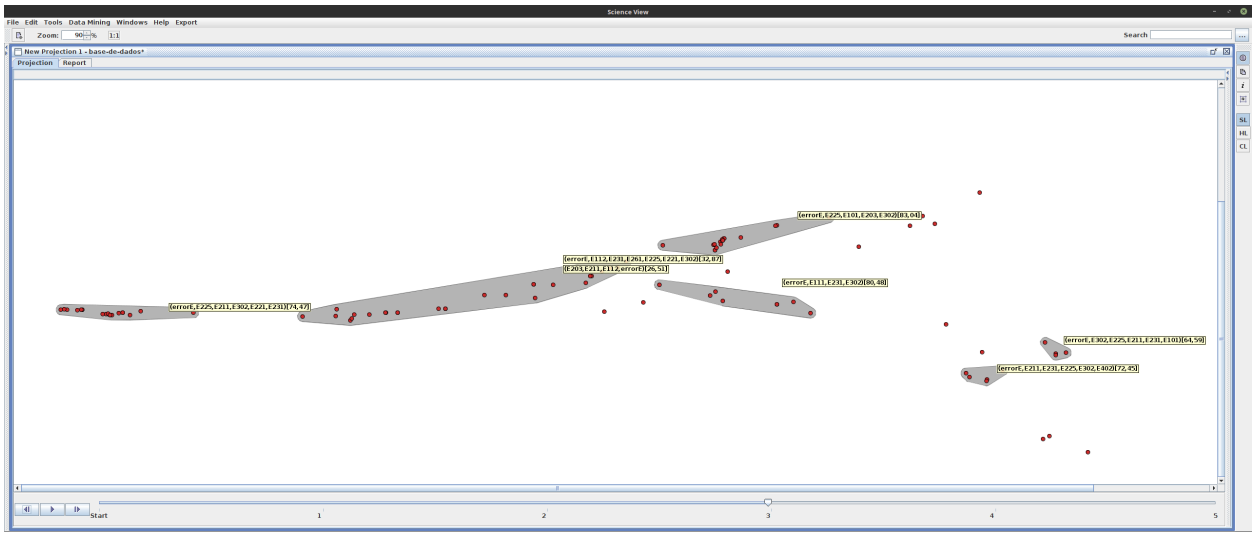

Figura 3.8. Final da exibição das projeções com a extração de tópicos ativada.

A Figura 3.9 exibe a possibilidade de abrir mais de uma implementação em uma mesma janela, utilizando abas. Para habilitar esse recurso, é necessário clicar no segundo botão no painel vertical a direita. Ao deixa o cursor sobre esse botão, aparecerá o seguinte texto: *View Content*. Em seguida, selecione as implementações que deseja abrir, mantendo o botão esquerdo do *mouse* pressionado e enquadrando na área verde, as implementações que deseja abrir na mesma janela.

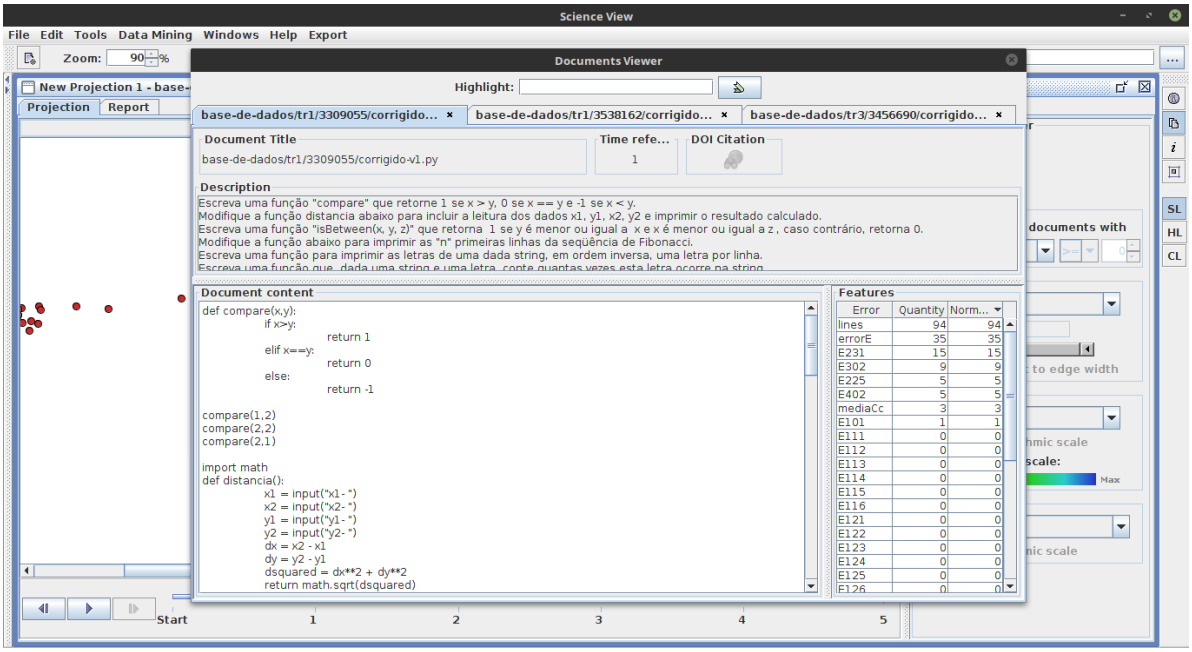

Figura 3.9. Abrir várias implementações em uma única janela.

A Figura 3.10 apresenta a possibilidade de abrir uma implementação em cada janela para que seja possível comparar a tabela de erros, por exemplo. Para abrir cada implementação em uma janela diferente, clique duas vezes com o botão esquerdo do *mouse* sobre o ponto da implementação. O visualizador da implementações permite redimensionar sua janela, bem como cada painel.

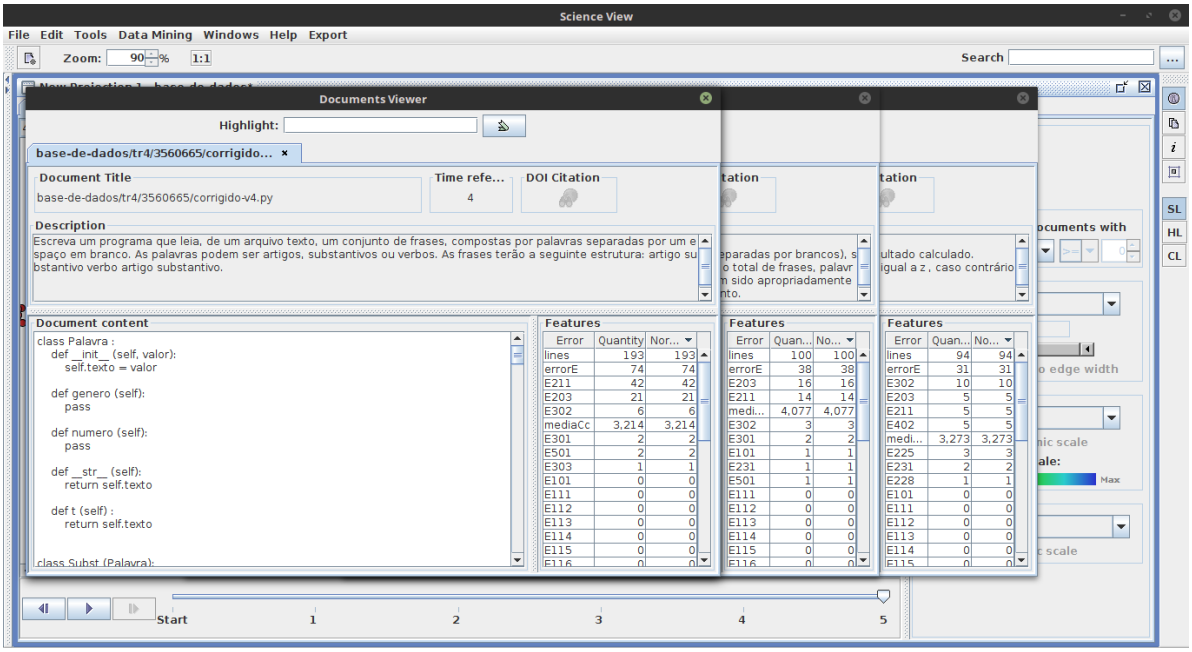

Figura 3.10. Varias implementações abertas em janelas diferentes.

3.4. Considerações finais

Utilizando o método e as ferramentas descritas neste capítulo, o próximo capítulo apresenta um estudo sobre avaliação de programas em Python, descrevendo a base de dados, a forma que foi realizado seu experimento sua avaliação e os resultados.

Capítulo 4

Resultados

Neste capítulo, são apresentadas as avaliações da projeção e da visualização como instrumento para avaliação de programas, ambos considerando um conjunto de programas escritos em Python para uma disciplina introdutória à Computação. A Seção 4.2 e a Seção 4.3 respondem a primeira e segunda questão de pesquisa, respectivamente.

4.1. Descrição da base

Essa base é constituída de 152 submissões em Python de 5 problemas distintos. Todas elas são oriundas de uma disciplina de programação orientada a objetos com a duração de um semestre. As submissões foram criadas por 49 alunos e cada aluno é representado por uma sequência de números, visto que eles foram anonimizados.

Os problemas foram aplicados por meio de exercício e tratam dos seguintes tópicos. O primeiro exercício consiste na revisão de conceitos básicos como: estruturas de condição e laço de repetição. O segundo exercício define a criação de classes com mais de um construtor, e manipulação de arquivos e cadeia de caracteres. O terceiro exercício consiste na criação das classes `Palavra` e `Texto`, no qual a primeira classe deve ser utilizada na segunda classe, e uma classe para gerar orações gramaticais. O quarto exercício refere-se a utilização de herança e polimorfismo, identificando cada palavra contida em um arquivo. O quinto exercício consiste na alteração de uma classe, implementando sobreposição de operadores.

4.2. Avaliação da projeção com preservação de vizinhança

A *ScienceView* utiliza a preservação da vizinhança (Seção 2.2.2) para verificar a qualidade das projeções. A Figura 4.1 apresenta a qualidade da projeção para essa base

de dados. Essa técnica visa avaliar se houve uma preservação da vizinhança dos objetos no espaço de alta dimensionalidade na projeção. A *Neighborhood Preservation* é calculada tomando os k vizinhos mais próximos de uma instância no espaço de alta dimensionalidade e os k vizinhos mais próximos na projeção, e verificando-se que proporção da vizinhança é preservada na projeção. A precisão final para um dado valor de k é a média das precisões para cada instância. Esse cálculo é feito para vários valores de k (tipicamente $k = [1, ..., 30]$) de forma a obtermos uma curva. Quanto mais altos os valores da curva para cada valor de k, maior a qualidade da projeção. Considerando que a técnica Time-based Least Square Projection (ALENCAR, 2013) gera uma sequência de projeções, temos uma curva para cada projeção da sequência.

Por exemplo, na Figura 4.1, na qual cada curva representa um dos cinco problemas citados anteriormente, é possível observar que as projeções no início da sequência apresentam maior qualidade quando comparada as demais, como é possível observar nas curvas dos problemas 3, 4 e 5. Isso é esperado, dado que, além de serem considerados mais submissões, as submissões referentes aos problemas 3, 4 e 5 são mais complexos do que aqueles referentes aos problemas 1 e 2, com características mais diversificadas e difícil representação com precisão no espaço 2D da projeção.

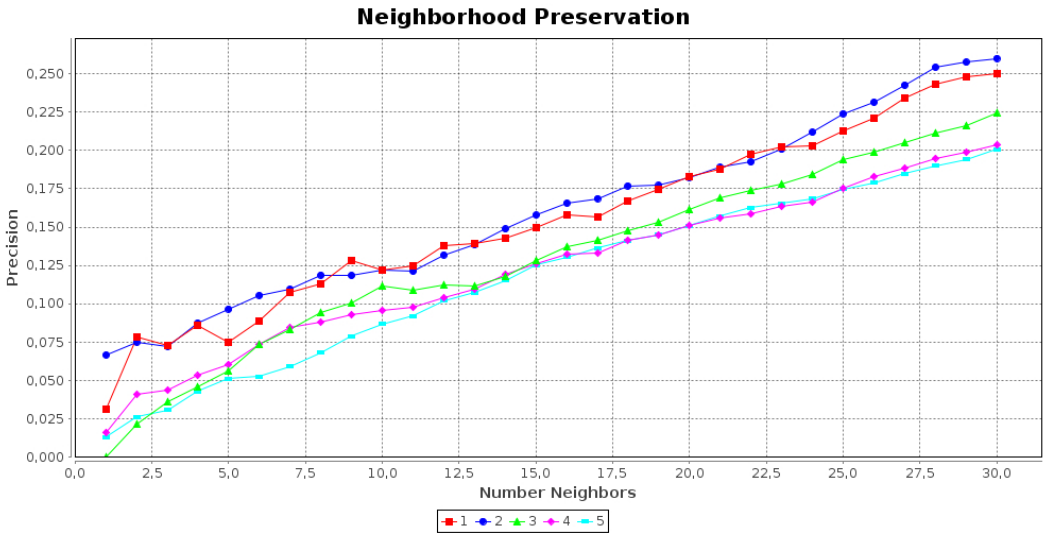

Figura 4.1. Qualidade da projeção, utilizando a preservação de vizinhança (*neighborhood preservation*).

Finalmente, observa-se a precisão de 20% para a linha 5, obtida com 30 vizinhos para a projeção considerando todos os problemas (e, consequentemente, todas as submissões). Considerando outros trabalhos que utilizam esta medida para avaliação de qualidade de projeções (PAULOVICH, 2008), a qualidade da projeção é adequada e compatível com outras projeções feitas com a técnica LSP e similares, visto que obteve resultado semelhante aos obtidos por Paulovich (2008).

4.3. Avaliação qualitativa da visualização

A Figura 4.2 apresenta os estágios realizados no estudo para obter os resultados qualitativos dessa pesquisa. O estudo inicia-se com o treinamento, seguido de duas etapas de correções das submissões utilizando a ferramenta *ScienceView* e apenas a tabela de medidas (Apêndice A), finalizando com o questionário (Apêndice B). A coluna a direita mostra o tempo de duração em minutos de cada etapa, com exceção do questionário que os voluntários responderam sem qualquer tipo de contagem de tempo.

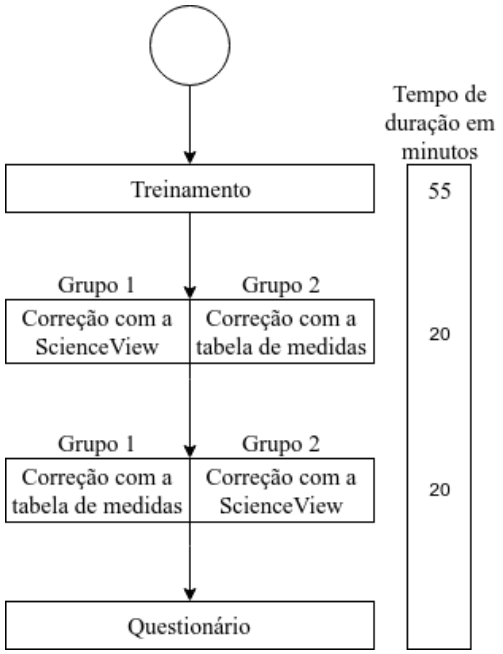

Figura 4.2. Fluxograma do estudo da avaliação qualitativa da visualização

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 da descrita na Seção 4.1 para não enviesar o estudo.

Encerrado o treinamento, procedeu-se para correção das submissões com a base descrita na Seção 4.1 com duração de 20 minutos para cada uma das duas etapas da correção. Para iniciarmos as correções, foi pedido para que os participantes encerrassem o funcionamento da *ScienceView* e a executassem com a criação da base de dados. O estudo

teve participação de 4 pessoas, devido vossa disponibilidade, e ambos os grupos utilizaram os dados obtidos pela *ScienceView-Python* (Seção 3.3.1). Todos os contribuintes realizaram a correção manual, o qual consistiu na correção somente do código-fonte, utilizando a tabela de medidas (Apêndice A), e somente 3 pessoas utilizaram a ferramenta *ScienceView* para auxiliar na correção. Essa etapa de correção consistiu em duas fases, todas envolvendo a correção das submissões, considerando os tipos de erros apontados na extração de características e a criação de um relato com os programas avaliados e comentários de correção.

Na primeira etapa da correção, 2 voluntários realizaram a correção manual, utilizando a tabela de medidas identificados pela ScienceView-Python, enquanto os outros 2 utilizaram a ferramenta ScienceView para auxiliar na correção. Na segunda etapa da correção, um dos contribuintes que realizou a correção manual passou a utilizar a *ScienceView*[1], enquanto os 2 que utilizaram a ferramenta na etapa anterior, realizaram a correção manual. Os participantes utilizaram a ferramenta sem ajuda no que já tinha sido apresentado e o avaliaram por meio de um questionário.

A Figura 4.3 apresenta a visualização dos agrupamentos das 152 submissões contidas na base de dados. Isso foi possível após a adaptação da ferramenta para leitura de arquivos no formato `CSV`. Cada ponto da visualização é referente a uma submissão. Ao clicar em um dos pontos, uma interface exibe sua implementação e as características extraídas pela *ScienceView-Python* para aquela submissão. Também é possível ordenar as colunas de características *Quantity* e *Normalized* em ordem crescente ou decrescente para visualizar as

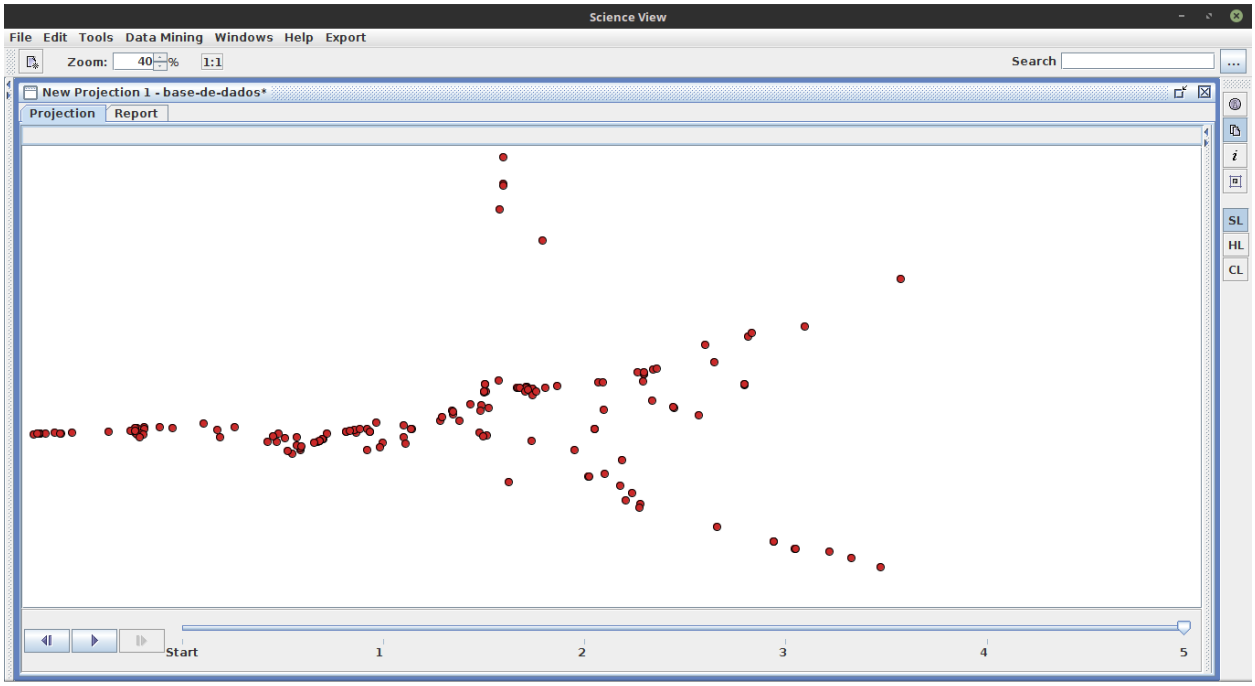

Figura 4.3. Visualização dos agrupamentos da base de dados gerado pela `ScienceView`.

[1] O outro participante não conseguiu executar a ferramenta ScienceView e não participou desta parte do estudo.

características que mais ocorreram. Por padrão, a tabela é apresentado ordenada de forma decrescente pela coluna *Normalized*.

A Tabela 4.1 apresenta a quantidade de implementações que cada participante corrigiu e de avaliações realizadas pelo voluntário a fim de auxiliar o aluno na aprendizagem. 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 e a quantidade de avaliações conforme o tipo de correção.

	Voluntário 1	Voluntário 2	Voluntário 3	Voluntário 4
Correção tradicional	3 códigos-fontes e 3 avaliações	4 códigos-fontes e 20 avaliações	3 códigos-fontes e 8 avaliações	5 códigos-fontes e 11 avaliações
ScienceView	Não utilizou a ferramenta	4 códigos-fontes e 4 avaliações	3 códigos-fontes e observou que a ferramenta auxiliou no encontro dos erros de forma mais rápida.	4 códigos-fontes e 8 avaliações

Tabela 4.1. Quantidade de correções e avaliações realizados com e sem a utilização da ferramenta

Apesar dos 3 voluntários afirmarem que a *ScienceView* colaborou para a correção das implementações, 2 deles corrigiram a mesma quantidade de implementações utilizando a ferramenta e manualmente. A exceção ocorreu com apenas 1 participante que extraiu os tópicos semelhantes manualmente. Na primeira extração, selecionou apenas 2 códigos-fontes e observou que os erros apresentados na tabela de erros eram semelhantes, visto que observou a importação de bibliotecas entre funções ao invés de realizá-las no início da implementação. Na segunda extração, selecionou 9 códigos-fontes e constatou que os erros ocorreram nas implementações e foi compreendido corretamente pela ferramenta.

4.4. Respostas do questionário

Para finalizar o estudo, os voluntários responderam um questionário anonimamente. O questionário, apresentado no (Apêndice B), contém questões objetivas e dissertativas para sabermos algumas informações profissionais 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.

Acreditávamos que o tempo de experiência como professor poderia interferir na quantidade de correções realizadas com a ferramenta, por isso foi questionado esse tempo de experiência. Dentre eles, 2 voluntários possuem 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.

4.5. Ameaças à validade

O treinamento realizado não foi suficiente para avaliar as implementações por meio da utilização da *ScienceView*, visto que um dos participantes relatou que a falta de conhecimento sobre a ferramenta fez com que correção manual acontecesse de forma mais rápida. Corrobora com isto a opinião de outro participante, que observou que a ferramenta agilizará o processo de correção conforme aumentar a frequência de uso da ferramenta. Principalmente, pudemos observar que os participantes do estudo não consideraram a avaliação de grupos de trabalho, optando por avaliar os trabalhos individualmente. Dadas essas observações, podemos concluir que o treinamento não foi suficiente para capacitar todos os participantes sobre a utilização da ferramenta.

Durante o treinamento, não foi exposto explicitamente que, ao utilizar a *ScienceView*, a avaliação é quanto aos códigos-fontes. Com isso, além de corrigirem as implementações, alguns participantes buscaram verificar se as características semelhantes apontadas pela

extração de tópicos realmente ocorriam nas implementações. Isso pode ter impactado na correção, devido ao tempo gasto para fazer essas verificações.

Outra ameaça é quanto ao tamanho da base de dados. Esta era formada por 152 códigos-fontes que solucionam 5 problemas distintos, algo considerado pequeno para um MOOC visto que uma das características do MOOC é a grande quantidade de usuários (*massive*). Desta forma, essa base de dados não reflete ao montante de submissões que podem ocorrer em um MOOC.

Outra limitação é quanto a quantidade de participantes. A aplicação do estudo contou com somente 4 pessoas, e ainda, somente 3 pessoas utilizarem a *ScienceView*, não nos retorna nada concreto estatisticamente. Para viabilizar mais participantes, é necessário, além de tempo, a disponibilidade de professores para participar do treinamento e utilizar a ferramenta, focando que o desenvolvimento do projeto pode contribuir para avaliação de trabalhos, além de possibilitar a verificação dos erros mais frequentes a fim de melhorar o aprendizado.

4.6. Considerações finais

Por meio dos resultados obtidos, o próximo capítulo apresenta as conclusões deste trabalho, considerando tanto a análise qualitativa quanto a quantitativa.

Capítulo 5

Conclusão

Este capítulo apresenta nossa conclusão com base nos resultados obtidos a fim de responder nossas questões de pesquisa e os trabalhos futuros para uma possível continuidade do projeto.

Por meio dos resultados obtidos, podemos responder nossas questões de pesquisa firmadas na Seção 3.1:

- $\mathbf{QP}_1$: o algoritmo de agrupamento produz grupos de códigos-fontes com boa qualidade?
- $\mathbf{QP}_2$: a utilização de subsídios de avaliação reduz o tempo de correção de todas as submissões?

A fim de responder a $\mathbf{QP}_1$, foi estudado e utilizado a preservação da vizinhança (PAULOVICH; MINGHIM, 2008), que quantifica a qualidade de uma projeção. Sua utilização ocorreu por meio da *ScienceView*. Tomaremos, como base para a nossa resposta, a Figura 4.1. Considerando outros trabalhos que utilizam esta medida para avaliação de qualidade de projeções (PAULOVICH, 2008), a qualidade da projeção é adequada e compatível com outras projeções feitas com a técnica LSP e similares, visto que obteve resultado semelhante aos obtidos por Paulovich (2008).

Para responder a $\mathbf{QP}_2$ foi necessário aplicar o estudo para saber quantas implementações os participantes analisariam em um determinado período de tempo, realizando as correções manualmente e por meio da utilização dos subsídios proposto. A utilização de subsídios não reduziu o tempo de correção das implementações em um período de 20 minutos, visto que alguns participantes corrigiram a mesma quantidade de implementações nas duas formas de análise, enquanto um participante realizou mais correções manualmente, quando comparado a utilização da ferramenta.

Contudo, quando utilizaram a *ScienceView*, podemos observar que os participantes realizaram poucas correções de implementações que estão no mesmo agrupamento. O

Voluntário 2 (Tabela 4.1) analisou uma implementação de um agrupamento com 9 códigos-fontes, duas implementações de um grupo com 12 códigos-fontes e uma implementação de um agrupamento contendo 5 códigos-fontes distintos. O Voluntário 3, apesar de corrigir uma implementação que estava isolada na projeção, analisou uma implementação de um agrupamento contendo 12 códigos-fontes e uma implementação de um agrupamento com 8 códigos-fontes. Finalmente, o Voluntário 4 corrigiu uma implementação que estava isolada na projeção, uma implementação de um agrupamento contendo 5 códigos-fontes, e duas implementação de agrupamentos distintos contendo 2 códigos-fontes distintos. Apesar de não podermos comprovar a segunda questão de pesquisa, o total de códigos-fontes que podem ser considerados corrigidos por meio da correção de uma implementação do agrupamento é alta. Isso nos dá indícios que, se melhorarmos o treinamento, poderemos alcançar nosso objetivo.

Como trabalhos futuros, é importante sanar as ameaças a validade (Seção 4.5) desse projeto. Focando, inicialmente, na construção de uma base de dados de implementações maior, a fim de tornar-se próximo a quantidade de submissões possíveis em um MOOC. Em seguida, melhorar nosso treinamento para que todos os participantes utilizem a ferramenta para sua finalidade, e encontrar mais professores dispostos a participar de todo o experimento.

Para uma base de dados maior, seria interessante avaliar o quanto as características extraídas neste estudo são relevantes e, caso necessário, realimentar o sistema da mineração de dados, como descrito na Seção 2.2. Por exemplo, poderia-se considerar características dinâmicas das submissões, executando casos de teste e, encontrando as variáveis comuns e renomeando-as para o nome mais utilizado (GLASSMAN et al., 2015). A combinação da análise dinâmica e estática pode gerar resultados melhores e particularmente úteis para avaliação de programas.

Referências

ALARIO-HOYOS, Carlos; PéREZ-SANAGUSTíN, Mar; DELGADO-KLOOS, Carlos; MUñOZ-ORGANERO, Mario; HERAS, Antonio Rodríguez-de-las et al. Analysing the impact of built-in and external social tools in a mooc on educational technologies. In: SPRINGER. *European Conference on Technology Enhanced Learning*. Paphos, Cyprus, 2013. p. 5–18.

ALENCAR, Aretha Barbosa. *Visualização da evolução temporal de coleções de artigos científicos*. Tese (Doutorado) — University of São Paulo, São Carlos, SP, fev. 2013. Disponível em: <http://www.teses.usp.br/teses/disponiveis/55/55134/tde-11042013-155653>.

ALENCAR, Aretha B.; BöRNER, Katy; PAULOVICH, Fernando V.; OLIVEIRA, Maria Cristina Ferreira de. Time-aware visualization of document collections. In: *27th Annual ACM Symposium on Applied Computing*. New York, NY: ACM, 2012. p. 997–1004. ISBN 978-1-4503-0857-1.

ANKERST, Mihael; BREUNIG, Markus M.; KRIEGEL, Hans-Peter; SANDER, Jörg. Optics: Ordering points to identify the clustering structure. *SIGMOD Rec.*, ACM, New York, NY, USA, v. 28, n. 2, p. 49–60, jun. 1999. ISSN 0163-5808. Disponível em: <http://doi.acm.org/10.1145/304181.304187>.

BAUER, H-U; PAWELZIK, Klaus R. Quantifying the neighborhood preservation of self-organizing feature maps. *IEEE Transactions on neural networks*, IEEE, v. 3, n. 4, p. 570–579, 1992.

BECK, Kent. *Test-driven development: by example*. Boston, MA: Addison-Wesley Professional, 2003. ISBN 9788577807246.

BERRY, Michael J.; LINOFF, Gordon. *Data Mining Techniques: For Marketing, Sales, and Customer Support*. New York, NY, USA: John Wiley & Sons, Inc., 1997. ISBN 0471179809.

CAMARA, Bruno Henrique Pachulski. *Uma investigação sobre o uso de critérios de teste no desenvolvimento baseado em testes para o ensino de programação*. Dissertação (Mestrado) — Universidade Tecnológica Federal do Paraná, Cornélio Procópio, 2016.

CHAKRABARTI, Soumen; ESTER, Martin; FAYYAD, Usama; GEHRKE, Johannes; HAN, Jiawei; MORISHITA, Shinichi; PIATETSKY-SHAPIRO, Gregory; WANG, Wei. Data mining curriculum: A proposal (version 1.0). *Intensive Working Group of ACM SIGKDD Curriculum Committee*, p. 140, 2006.

CHOUDHURY, Rohan Roy; YIN, HeZheng; MOGHADAM, Joseph; FOX, Armando. Autostyle: Toward coding style feedback at scale. In: ACM. *Proceedings of the 19th ACM Conference on Computer Supported Cooperative Work and Social Computing Companion*. San Francisco, CA, USA, 2016. (CSCW '16), p. 21–24. Disponível em: <http://dx.doi.org/10.1145/2818052.2874315>.

CORDASCO, Ian. *Flake8: Your Tool For Style Guide Enforcement*. 2010. 2.5.1. Disponível em: <https://flake8.pycqa.org/en/latest/>.

CORDASCO, Ian. *McCabe checker, plugin for flake8*. 2013. 0.3.1. Disponível em: <https://pypi.python.org/pypi/mccabe>.

DELLAROCAS, Chrysanthos; ALSTYNE, Marshall Van. Money models for moocs. *Communications of the ACM*, ACM, v. 56, n. 8, p. 25–28, 2013.

EDWARDS, Stephen H. Rethinking computer science education from a test-first perspective. In: *Companion of the 18th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*. New York, NY, USA: ACM, 2003. (OOPSLA '03), p. 148–155. ISBN 1-58113-751-6. Disponível em: <http://doi.acm.org/10.1145/949344.949390>.

EDWARDS, Stephen H. Using software testing to move students from trial-and-error to reflection-in-action. In: *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education*. New York, NY, USA: ACM, 2004. p. 26–30. ISBN 1-58113-798-2.

ESTER, Martin; KRIEGEL, Hans P; SANDER, Jorg; XU, Xiaowei. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. *Second International Conference on Knowledge Discovery and Data Mining*, p. 226–231, 1996. ISSN 09758887. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.20.2930>.

FASSBINDER, Aracele; DELAMARO, Márcio Eduardo; BARBOSA, Ellen Francine. Construção e uso de moocs: Uma revisão sistemática. In: *Anais do Simpósio Brasileiro de Informática na Educação*. Dourados, MS: SBIE, CBIE, 2014. v. 25, n. 1, p. 332.

FAYYAD, Usama; PIATETSKY-SHAPIRO, Gregory; SMYTH, Padhraic. From data mining to knowledge discovery in databases. *AI magazine*, v. 17, n. 3, p. 37, 1996.

FITZPATRICK, Jerry. Applying the abc metric to c, c++, and java. *More C++ gems*, p. 245–264, 1997.

FRIEDMAN, Jerome H; TUKEY, John W. A projection pursuit algorithm for exploratory data analysis. September, 1974.

GLASSMAN, Elena. Visualizing and classifying multiple solutions to engineering design problems. In: ACM. *Proceedings of the ninth annual international ACM conference on International computing education research*. New York, NY, USA, 2013. p. 175–176.

GLASSMAN, Elena L. Interacting with massive numbers of student solutions. In: ACM. *Proceedings of the adjunct publication of the 27th annual ACM symposium on User interface software and technology*. Honolulu, HI, USA, 2014. p. 17–20.

GLASSMAN, Elena L. *Clustering and Visualizing Solution Variation in Massive Programming Classes*. Tese (Doutorado) — Massachusetts Institute of Technology, 2016.

GLASSMAN, Elena L; GULLEY, Ned; MILLER, Robert C. Toward facilitating assistance to students attempting engineering design problems. In: ACM. *Proceedings of the ninth annual international ACM conference on International computing education research*. New York, NY, USA, 2013. p. 41–46.

GLASSMAN, Elena L; LIN, Aaron; CAI, Carrie J; MILLER, Robert C. Learnersourcing personalized hints. In: ACM. *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing*. [S.l.], 2016. p. 1626–1636.

GLASSMAN, Elena L.; SCOTT, Jeremy; SINGH, Rishabh; GUO, Philip J.; MILLER, Robert C. Overcode: Visualizing variation in student solutions to programming problems at scale. *ACM Trans. Comput.-Hum. Interact.*, ACM, New York, NY, USA, v. 22, n. 2, p. 7:1–7:35, mar. 2015. ISSN 1073-0516. Disponível em: <http://doi.acm.org/10.1145/2699751>.

GLASSMAN, Elena L.; SINGH, Rishabh; MILLER, Robert C. Feature engineering for clustering student solutions. In: *Proceedings of the First ACM Conference on Learning @ Scale Conference*. New York, NY, USA: ACM, 2014. (L@S '14), p. 171–172. ISBN 978-1-4503-2669-8. Disponível em: <http://doi.acm.org/10.1145/2556325.2567865>.

HALKIDI, Maria; BATISTAKIS, Yannis; VAZIRGIANNIS, Michalis. On clustering validation techniques. *Journal of Intelligent Information Systems*, v. 17, n. 2, p. 107–145, 2001. ISSN 1573-7675. Disponível em: <http://dx.doi.org/10.1023/A:1012801612483>.

JACCARD, Paul. *Etude comparative de la distribution florale dans une portion des Alpes et du Jura*. Société Vaudoise des Sciences Naturelles: Impr. Corbaz, 1901.

KAPP, Karl M. *The gamification of learning and instruction: game-based methods and strategies for training and education*. [S.l.]: John Wiley & Sons, 2012.

KEIM, Daniel A. Information visualization and visual data mining. *IEEE transactions on Visualization and Computer Graphics*, IEEE, v. 8, n. 1, p. 1–8, 2002.

KLOBAS, Mackintosh; MURPHY. The anatomy of moocs. In: KIM, Paul (Ed.). *Massive open online courses: the MOOC revolution*. New York, NY; Abingdon, Oxon: Routledge, 2014. cap. 1, p. 1–22. ISBN 041573309X.

LOUDEN, Kenneth C. Gramáticas livres de contexto e análise sintática. In: *Compiladores-Princípios e Práticas*. São Paulo, SP: Cengage Learning, 2004. cap. 3, p. 95–142.

MAATEN, Laurens van der; HINTON, Geoffrey. Visualizing data using t-sne. *Journal of Machine Learning Research*, v. 9, n. Nov, p. 2579–2605, 2008.

MACQUEEN, James et al. Some methods for classification and analysis of multivariate observations. In: OAKLAND, CA, USA. *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*. Los Angeles, California, USA, 1967. v. 1, n. 14, p. 281–297.

MCCABE, Thomas J. A complexity measure. *IEEE Transactions on software Engineering*, IEEE, n. 4, p. 308–320, 1976.

MEHLENBACHER, Brad. Massive open online courses (moocs): Educational innovation or threat to higher education? In: *Proceedings of the Workshop on Open Source and Design of Communication*. New York, NY, USA: ACM, 2012. (OSDOC '12), p. 99–99. ISBN 978-1-4503-1525-8. Disponível em: <http://doi.acm.org/10.1145/2316936.2316953>.

MOGHADAM, Joseph Bahman; CHOUDHURY, Rohan Roy; YIN, HeZheng; FOX, Armando. Autostyle: Toward coding style feedback at scale. In: *Proceedings of the Second (2015) ACM Conference on Learning @ Scale*. New York, NY, USA: ACM, 2015. (L@S '15), p. 261–266. ISBN 978-1-4503-3411-2. Disponível em: <http://doi.acm.org/10.1145/2724660.2728672>.

OLIVEIRA, MC Ferreira de; LEVKOWITZ, Haim. From visual data exploration to visual data mining: a survey. *IEEE Transactions on Visualization and Computer Graphics*, IEEE, v. 9, n. 3, p. 378–394, 2003.

PAULOVICH, Fernando Vieira. *Mapeamento de dados multi-dimensionais - integrando mineração e visualização*. Tese (Doutorado) — University of São Paulo, São Carlos, SP, 2008. Disponível em: <http://www.teses.usp.br/teses/disponiveis/55/55134/tde-04032009-145018/pt-br.php>.

PAULOVICH, Fernando V; MINGHIM, Rosane. Hipp: A novel hierarchical point placement strategy and its application to the exploration of document collections. *IEEE Transactions on Visualization and Computer Graphics*, IEEE, v. 14, n. 6, p. 1229–1236, 2008.

PAULOVICH, Fernando V; NONATO, Luis G; MINGHIM, Rosane; LEVKOWITZ, Haim. Least square projection: A fast high-precision multidimensional projection technique and its application to document mapping. *IEEE Transactions on Visualization and Computer Graphics*, IEEE, v. 14, n. 3, p. 564–575, 2008.

REZENDE, S.O.; PUGLIESI, J.B.; MELANDA, E.A..; PAULA, M.F. de. Mineração de dados. In: *Sistemas inteligentes: fundamentos e aplicações*. São Paulo, SP: Editora Manole, 2003. cap. 12, p. 307–335. ISBN 9788520416839.

RODRIGUEZ, C Osvaldo. 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*, v. 15, n. 2, 2012.

ROSSUM, Guido van; WARSAW, Barry; COGHLAN, Nick. Pep 8–style guide for python code. *Available at ht tp://www. python. org/dev/peps/pep-0008*, 2001.

ROSSUM, Guido van; WARSAW, Barry; COGHLAN, Nick. *PEP8: Python style guide checker*. 2001. 1.5.7. Disponível em: <https://www.python.org/dev/peps/pep-0008/>.

ROUSSEEUW, Peter J. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, Elsevier, v. 20, p. 53–65, 1987.

SCHLEIMER, Saul; WILKERSON, Daniel S.; AIKEN, Alex. Winnowing: Local algorithms for document fingerprinting. In: *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*. New York, NY, USA: ACM, 2003. (SIGMOD '03), p. 76–85. ISBN 1-58113-634-X. Disponível em: <http://doi.acm.org/10.1145/872757.872770>.

SCHMIDT, Douglas C.; MCCORMICK, Zach. Producing and delivering a coursera mooc on pattern-oriented software architecture for concurrent and networked software. In: *Proceedings of the 2013 Companion Publication for Conference on Systems, Programming, & Applications: Software for Humanity*. New York, NY, USA: ACM, 2013. (SPLASH '13), p. 167–176. ISBN 978-1-4503-1995-9. Disponível em: <http://doi.acm.org/10.1145/2508075.2508465>.

SIEMENS, George. Massive open online courses: Innovation in education. *Open educational resources: Innovation, research and practice*, Vancouver, BC Canada: Commonwealth of Learning and Athabasca University, v. 5, 2013.

SIVAMUNI, Kalaimagal; BHATTACHARYA, Sujoy. Assembling pieces of the moocs jigsaw puzzle. In: IEEE. *MOOC Innovation and Technology in Education (MITE), 2013 IEEE International Conference in*. Jaipur, India, 2013. p. 393–398.

SUBBIAN, Vignesh. Role of moocs in integrated stem education: A learning perspective. In: IEEE. *Integrated STEM Education Conference (ISEC), 2013 IEEE*. Friend Center Olden and William Streets Princeton, NJ, USA, 2013. p. 1–4.

TAHERKHANI, Ahmad; KORHONEN, Ari; MALMI, Lauri. Automatic recognition of students' sorting algorithm implementations in a data structures and algorithms course. In: *Proceedings of the 12th Koli Calling International Conference on Computing Education Research*. New York, NY, USA: ACM, 2012. (Koli Calling '12), p. 83–92. ISBN 978-1-4503-1795-5. Disponível em: <http://doi.acm.org/10.1145/2401796.2401806>.

TAN, Pang-Ning; STEINBACH, Michael; KUMAR, Vipin. Classification: Basic concepts, decision trees, and model evaluation. In: . Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2005. cap. 4, p. 145–205. ISBN 0321321367.

TAN, Pang-Ning; STEINBACH, Michael; KUMAR, Vipin. Cluster analysis: Basic concepts and algorithms. In: . Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2005. cap. 8, p. 487–568. ISBN 0321321367.

TAN, Pang-Ning; STEINBACH, Michael; KUMAR, Vipin. *Introduction to Data Mining, (First Edition)*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2005. ISBN 0321321367.

WEI, Zhaodong; WU, Wenjun. A peer grading tool for MOOCs on programming. *IFIP Advances in Information and Communication Technology*, v. 503, p. 378–385, 2015. ISSN 18684238. Disponível em: <http://www.scopus.com/inward/record.url?eid=2-s2.0-84927750230{\&}partnerID=tZOtx>.

YIN, Hezheng; MOGHADAM, Joseph; FOX, Armando. Clustering student programming assignments to multiply instructor leverage. In: *Proceedings of the Second (2015) ACM Conference on Learning @ Scale*. New York, NY, USA: ACM, 2015. (L@S '15), p. 367–372. ISBN 978-1-4503-3411-2. Disponível em: <http://doi.acm.org/10.1145/2724660.2728695>.

ZHANG, Kaizhong; SHASHA, Dennis. Simple fast algorithms for the editing distance between trees and related problems. *SIAM journal on computing*, SIAM, v. 18, n. 6, p. 1245–1262, 1989.

Apêndices

Apêndice

A

Erros do PEP8

Erro	Descrição	Exemplo do erro
		Correto: if a == 0:\n a = 1\n b = 1
E101	Indentação contém espaços e tabulações misturados	if a == 0:\n a = 1\n\tb = 1
		Correto: a = 1 ou if a == 0:\n a = 1
E111	Nível sem indentação e foi encontrado espaços em branco	
E112	Nível com uma indentação, mas não foi encontrado uma indentação	
E113	Nível sem indentação e foi encontrado indentação	
E114	Nível sem indentação e foi encontrado espaços em branco e comentário de uma linha	
E115	Nível com uma indentação, mas não foi encontrado uma indentação, seguido de comentário	
E116	Nível sem indentação e foi encontrado indentação seguido de comentário	
		Correto: a = (\n) ou a = (\n 42)
E121	Há espaços, porém é menos que uma tabulação (4 espaços)	a = (\n 42)
E122	Sem indentação	a = (\n42)
E123	Não encontra ")"ou indentation of opening bracket's line	a = (\n) ou a = (\n 42\n)
E124	Não encontra ")"visualmente indentado	a = (24,\n 42\n)
E125	Falta indentação na continuação da declaração	if (\n b):\n pass
E126	Excesso de indentação para ")"	a = (\n 42)
E127	Excesso de indentação afim de indentar visualmente	a = (24,\n 42)
E128	Continuação da linha com limitação de indentação para indentar visualmente	a = (24,\n 42)
E129	Linha indentada visualmente	if (a or\n b):\n pass
E131	unaligned for hanging indent	a = (\n 42\n 24)
E133	Fecha ")"e falta indentação	

Tabela A.1. Erros extraídos do estilo de escrita PEP8 referente a indentação

Erro	Descrição	Exemplo do erro
		Correto: spam(ham[1], {eggs: 2})
E201	Espaço em branco após (, [ou {	spam(ham[1], {eggs: 2})
E202	Espaço em branco antes de),] ou	spam(ham[1], {eggs: 2})
E203	Espaço em branco antes de ":", ";", ","	if x == 4: print x, y; x, y = y , x
		Correto: spam(1) ou dict['key'] = list[index]
E211	Espaço em branco antes de "("ou "["	dict ['key'] = list[index] ou dict['key'] = list [index]
		Correto: a = 12 + 3
E221	Múltiplos espaços antes do operador	a = 4 + 5
E222	Múltiplos espaços depois do operador	a = 4 + 5
E223	Tabulação antes do operador	a = 4\t+ 5
E224	Tabulação depois do operador	a = 4 +\t5
		Correto: foo(bar, key='word', *args, **kwargs)
E225	Falta espaço em branco em volta do operador	i=i+1
E226	Aritmética	c = (a+b) * (a-b) ou hypot2 = x*x + y*y
E227	Bit a bit ou shift	c = a|b
E228	Modulo	msg = fmt%(errno, errmsg)
		Correto: [a, b] ou (3,) ou a[1:4] ou a[1:4:2]
E231	Não tem espaço em branco após ",", ";"e ":"	['a','b']
		Correto: a = (1, 2)
E241	Múltiplos espaços após ","	a = (1, 2)
E242	Tabulação após ","	a = (1,\t2)
		Correto: def complex(real, imag=0.0): ou boolean(a >= b)
E251	Espaços inesperados ao redor de "="(atribuição)	def complex(real, imag = 0.0):
		Correto: x = x + 1 # Increment x ou x = x + 1 # Increment x
E261	Menos de dois espaços antes do comentário	x = x + 1 # Increment x
E262	Comentário deve iniciar com # e um espaço	x = x + 1 #Increment x ou x = x + 1 # Increment x
E265	Comentário em bloco deve inicar com # e um espaço	#Block comment
E266	Muitos # para o bloco de comentário	### Block comment
		Correto: True and False
E271	Múltiplos espaços antes de uma palavra reservada	True and False
E272	Múltiplos espaços depois de uma palavra reservada	True and False
E273	Tabulação após a palavra reservada	True and\tFalse
E274	Tabulação antes da palavra reservada	True\tand False

Tabela A.2. Erros extraídos do estilo de escrita PEP8 referente a espaços em branco

Erro	Descrição	Exemplo do erro
		Correto: def a():\n pass\n\n\ndef b():\n pass
		Correto: def a():\n pass\n\n\n# Foo\n# Bar\n\ndef b():\n pass
E301	Espera uma linha em branco e encontrou nenhuma	class Foo:\n b = 0\n def bar():\n pass
E302	Espera duas linhas em branco e encontrou uma quantidade diferente de dois	def a():\n pass\n\ndef b(n):\n pass
E303	Muitas linhas em branco	def a():\n pass\n\n\n\ndef b(n):\n pass
E304	Blank lines found after function decorator - Encontrou linhas em branco após a função principal	@decorator\n\ndef a():\n pass

Tabela A.3. Erros extraídos do estilo de escrita PEP8 referente a linhas em branco

Erro	Descrição	Exemplo do erro
		Correto: import os\nimport sys
E401	Múltiplos import em uma linha	import sys, os
E402	import após qualquer outro tipo de instrução	a = 1\nimport os

Tabela A.4. Erros extraídos do estilo de escrita PEP8 referente a importação de bibliotecas

Erro	Descrição	Exemplo do erro
E501	Linha com 80 caractéres ou mais	Correto: aaa = [123,\n 123]
		Correto: aaa = ("bbb "\n "ccc")
		Correto: aaa = "bbb "\\n "ccc"
		Correto: aaa = 123 # \\
E502	Contrabarra é redundante entre parênteses	aaa = [123, \\n 123] ou aaa = ("bbb "\\n "ccc")

Tabela A.5. Erros extraídos do estilo de escrita PEP8 referente a tamanho da instrução em caracteres

Erro	Descrição	Exemplo do erro
		Correto: if foo == 'blah':\n do_blah_thing()
		Correto: do_one() ou do_two() ou do_three()
E701	Várias instruções em uma linha com ":"	if foo == 'blah': do_blah_thing()
E702	Múltiplas instruções em uma linha separados com ";"	do_one(); do_two(); do_three()
E703	Declaração termina com um ";"	do_four(); # useless semicolon
		Correto: if arg is not None:
E711	Comparar singleton com com operador lógico diferente	if arg != None:
E712	Comparar singleton com com operador lógico igual	if arg == True:
		Correto: if x not in y:\n pass
		Correto: assert (X in Y or X is Z)
		Correto: if not (X in Y):\n pass
		Correto: zz = x is not y
E713	Deve-se utilizar not in para verificar se variável está contido em outra	Z = not X in Y ou if not X.B in Y:\n pass
E714	Deve-se utilizar is not para comparar objeto	if not X is Y:\n pass ou Z = not X.B is Y
		Correto: if isinstance(obj, int):
E721	Não comparar tipos, usar isinstance()	if type(obj) is type(1):
E731	Utilizou expressão lamba, sendo que é pra usar def	f = lambda x: 2*x

Tabela A.6. Erros extraídos do estilo de escrita PEP8 referente a quantidade de instrução por linha e formas de instrução

Erro	Descrição	Exemplo do erro
E901	Checa se a sintaxe é válida	
E902	Tokenize the file, run physical line checks and yield tokens	

Tabela A.7. Erros extraídos do estilo de escrita PEP8 referente a verificação de sintaxe e geração de *tokens*

Apêndice B

Questionário

As questões que possuem o símbolo * (asterisco) no final são obrigatórias. As questões 1, 2, 4, 5, 6, 7, 8 e 9 são objetivas e as questões 3, 10 e 11 são dissertativas.

1. Qual seu nível de conhecimento em Python? *
   ( ) Básico
   ( ) Intermediário
   ( ) Avançado
2. Qual seu tempo de experiência como professor? *
   ( ) Menos que 1 ano
   ( ) De 1 a 2 anos
   ( ) De 2 a 5 anos
   ( ) De 5 a 10 anos
   ( ) Acima de 10 anos
3. O tutorial ajudou a utilizar a ferramenta? *
   ( ) Sim
   ( ) Não
4. Caso a resposta anterior for negativa, quais informações podem ser apresentadas no tutorial?
5. A descrição do exercício estava bem detalhada para solucionar o problema? *
   ( ) Sim
   ( ) Não
6. A descrição do exercício ocasionou dúvidas em relação ao desenvolvimento da solução? *
   ( ) Sim
   ( ) Não

7. Faltou alguma informação a ser apresentada na janela que consta o código-fonte? *
   ( ) Sim
   ( ) Não
8. Em relação a pergunta anterior. Se respondeu “Sim”, quais informações podem ser apresentadas?
9. A visualização da projeção na Science View, colaborou para a correção das implementações? *
   ( ) Sim
   ( ) Não
10. Disserte sobre o *feedback* recebido da Science View por meio da visualização dos agrupamentos. *
11. Disserte sobre a sua experiência em relação ao uso da Science View. *