Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Coordenação de Pós-Graduação em Ciência da Computação

Feedback através da Comparação de Códigos no Apoio ao Processo de Ensino-Aprendizagem de Introdução à Programação

Matheus Gaudencio do Rêgo

Tese submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Campina Grande - Campus I como parte dos requisitos necessários para obtenção do grau de Doutor em Ciência da Computação.

Área de Concentração: Ciência da Computação
Linha de Pesquisa: Engenharia de Software

Dalton D. Serey Guerrero (Orientador)
Ayla Débora Dantas S. Rebouças (Orientador)

Campina Grande, Paraíba, Brasil

Resumo

A atividade de codificar programas é uma das mais utilizadas no ensino de programação. Uma das dificuldades na realização desta atividade está em ser capaz de obter informação sobre os códigos produzidos. A atividade manual de análise de códigos é lenta, especialmente quando se pensa em observar um conjunto de programas como um todo. Ao observar o conjunto de códigos, professores e alunos podem refletir sobre as estratégias tentadas para uma determinada questão ou identificar relações entre as diferentes questões de uma lista de exercícios.

Este trabalho de doutorado utilizou estratégias automáticas para a comparação de códigos de uma população de programas. Tais técnicas foram avaliadas e, a partir delas, foram criadas visualizações que possibilitam explorar tal informação. A metodologia inicial para fazer essa investigação consistiu em um experimento avaliando como professores comparam códigos e como esta comparação se assemelha a técnicas automáticas de comparação de código.

Posteriormente, os algoritmos de comparação de código foram usados para a criação de uma visualização em grafo de tais comparações. Esta visualização teve seu uso testado pelos alunos em uma atividade de avaliação de códigos entre-pares. Por fim, foi criada uma visualização através de um mapa-de-calor para comparar diferentes questões de uma lista de exercícios, bem como a indentificação dos tópicos que cada questão explorava.

Como resultados, foi possível identificar que os professores podem não ter apresentado alta concordância entre si (mínimo de 62%) em como códigos se relacionam, mas ainda assim foi possível capturar a noção de similaridade entre a forma em que alguns professores e estratégias automáticas comparam códigos (mínimo de 75%). No entanto, para atingir tal concordância, é preciso considerar um parâmetro que define quando uma estratégia deve considerar duas soluções como muito próximas.

Além disso, observou-se que a avaliação de código pelos próprios alunos guiada pela comparação entre artefatos não alterou a qualidade da avaliação de soluções de um problema, mas melhorou a percepção que o estudante tem sobre o processo de avaliação de códigos entre-pares, bem como a percepção da sua própria capacidade de revisar o que foi produzido pelos colegas. Os estudantes também consideraram que a visualização era simples de ser utilizada.

O útimo estudo apontou que é possível construir uma visualização para comparar diferentes questões, bem como identificar tópicos explorados numa lista de exercícios. Em um estudo qualitativo, foi possível refletir sobre a produção de exercícios durante a disciplina e sobre a intensidade com que determinados tópicos foram praticados ao longo do semestre.

Abstract

In introductory programming courses it is common to ask students to solve exercises that require code as solution. When evaluating a code, a teacher or student often compare how that solution relates to other codes. This comparison gives insight about different strategies to solve a problem or information about what students are doing for a given problem. It is a costly task to evaluate codes and understand how they are related.

We propose a novel approach to use automatic code comparison strategies to compare a pool of programs. Using the result from this approach we created visualizations for this given data. First, we present a study on how automatic code comparison strategies are related to the way that teachers compare codes. Then we use the code similarity information to create a graph visualization that was later used by students during a peer assessment task. Finally we create a heatmap that show how different questions are related and which topics each of those questions explore. Then we do a qualitative study on how that information can be used and is useful for a teacher.

Our first experiment shows that teachers may not have a high agreement among themselves while comparing codes, but is possible to have a personalized fined tuned algorithm that has a great similarity score with some teachers. Code comparison doesn't seem to improve the technical quality of feedback created by the students during the peer assessment task, but that visualization improves the student's confidence in their peer assessment capability. The last study shows that we can have information about how different exercises are related and how some topics are being used in such exercises.

Agradecimentos

Ao professor Camilo de Lelis Gondim Medeiros, meu primeiro professor de programação na Universidade Federal de Campina Grande. Que o sorriso dele, a alegria para ensinar e a preocupação com os alunos nunca sejam esquecidos.

Aos professores Francisco Brasileiro, Dalton Guerrero e Ayla Débora. O primeiro, por ter me colocado em pesquisa e me orientado no mestrado. E a Dalton e Ayla pela orientação que foi me dada no Doutorado.

À minha irmã, que foi indiretamente responsável por toda essa pesquisa. Tudo começou quando fui buscá-la em uma reunião sobre similaridade genética. Acabei chegando mais cedo, aprendendo sobre o assunto e aplicando os mesmos princípios ao que seria o começo do meu doutorado.

Agradeço também a toda minha família, especialmente Gabriel, Rogéria e Rômulo; irmão, mãe e pai, que não me perguntaram constantemente sobre o meu doutorado e sempre me ofereceram ajuda. Aos meus amigos Felipe Gesteira (e à esposa Ana e ao filho Jorge) e a Lorena Maia.

A todos os colegas e amigos do SPLab, minha nova residência no doutorado. Especialmente a professora Eliane, Catharine, João Arthur, Katyusco, Kláudio, Saulo, Fabrício, Luis, Ana, Arthur, Bruno, Lilian, Paloma. Aos professores Jorge e Franklin. Agradecimentos às funcionárias da COPIN, Rebeka e Liana.

E um grande obrigado a todos os alunos de programação com quem tive contato. Foram tantos acompanhando a disciplina desde 2008, mas fica uma dedicatória especial a todos aqueles com quem tive oportunidade de estar mais próximo: Carla Amaral, Carlos Rafael, Jessica Sousa, João Pedro, Letícia Wanderley, Gustavo Henrique, Martha Michelly, Julio Leitão, Filipe Wesley, Jessika Renally, Julie Pessoa, Walter Alves, Gilles Medeiros, Victor Hugo e Tainah Emmanuele.

Conteúdo

1 Introdução **1**

1.1 Contextualização 1

1.2 O problema 4

1.3 Objetivos 5

1.4 Sumário da tese 6

1.4.1 Implicações 7

1.5 Outline da tese 7

1.6 Publicações 8

2 Fundamentação Teórica **11**

2.1 Ensino de Programação 11

2.2 Feedback e Diagnóstico 12

2.3 Comparação de Códigos 14

2.3.1 Medidas de Similaridade 16

2.3.2 Visualização de Questões Similares 21

2.3.3 Agrupamento Semântico de Termos 23

3 Comparação Automática e Manual de Programas **25**

3.1 Contextualização 25

3.2 Design Experimental 26

3.2.1 Questões de Pesquisa 26

3.2.2 Amostra do Experimento 27

3.2.3 Coleta de Dados 30

3.2.4 Processamento dos Dados 32

3.3 Resultados Obtidos 32
3.3.1 Q1. Há concordância entre os professores na avaliação de similaridade questões? 32
3.3.2 Q2. Há concordância entre os professores e estratégias automáticas na avaliação de similaridade? 34
3.4 Discussão 35
3.5 Ameaças à Validade 36
3.6 Trabalhos Relacionados 37
3.7 Sumário 37

4 Visualização e Uso da Comparação de Programas na Análise de Soluções 39
4.1 Visualização da Comparação de Programas 39
4.2 Design do Experimento 41
4.2.1 Questões de Pesquisa 41
4.2.2 Metodologia 41
4.2.3 Dados 46
4.2.4 Coleta de Dados 53
4.2.5 Processamento dos Dados 55
4.3 Resultados Obtidos 56
4.3.1 Q2. Ao mostrar uma visualização da comparação de códigos durante a revisão de códigos, os alunos melhoram sua capacidade imediata de produzir uma solução correta para uma questão semelhante à avaliada? 57
4.3.2 Q3. A visualização da comparação de códigos estimula os alunos a olharem mais códigos? 58
4.3.3 Q4. Os alunos gostam da ferramenta de avaliação de código? 58
4.4 Discussão 59
4.5 Ameaças à Validade 60
4.6 Trabalhos Relacionados 61
4.7 Sumário 61

5 Análise de Questões de Listas de Exercícios **63**
5.1 Visualização das Questões de uma Lista 63
5.1.1 Identificação da Solução Relevante 63
5.1.2 Visualização da Comparação por Similaridade 64
5.2 Agrupamento Semântico 65
5.3 Estudo Qualitativo 66
5.3.1 Amostra do Estudo 67
5.3.2 Coleta de Dados 68
5.4 Resultados 68
5.4.1 Identificação da Solução Relevante 68
5.4.2 Identificação de Tópicos 68
5.4.3 Comparação de Similaridade entre Questões 71
5.5 Discussão 73
5.6 Limitações do Estudo 74
5.7 Trabalhos Relacionados 74
5.8 Sumário 75

6 Conclusões **76**
6.1 Contribuições 77
6.2 Trabalhos Futuros 77
6.3 Outras Implicações da Tese 78

A Webshell **80**

B Hábitos de Estudo e Desempenho do Aluno **82**

C TSTView **83**

D Dados do Experimento de Comparação de Códigos **85**

E Dados do Experimento da Avaliação de Códigos entre-pares **87**

Lista de Figuras

2.1 Exemplo de visualização da comparação de códigos utilizando a distância de Jaccard e $\Delta = 0,05$. 22

3.1 As três soluções de base para o cálculo do número da série de Fibonacci. (a) sem função, (b) função iterativa, (c) função recursiva 28
3.2 Tela da aplicação web utilizada para que o professor comparasse três códigos. 31
3.3 Número de vezes em que os códigos C_2 C_1 foram escolhidos como igualmente similares/diferentes de R para cada professor. 34

4.1 Ferramenta de comparação de códigos com a visualização proposta 40
4.2 Primeira etapa da ferramenta de comparação de códigos 43
4.3 Segunda etapa da ferramenta de comparação de códigos para o grupo de controle 44

5.1 Exemplo de Mapa de Calor 65
5.2 Questões e os tópicos detectados 70
5.3 Similaridade entre questões e os tópicos associados 72
5.4 Dois grupos de questões detectados 73

A.1 Exemplo da análise do processo de codificação do aluno capturado pelo Webshell. 81

C.1 Tela do TSTView para acompanhamento dos alunos na visão do professor. . 83
C.2 Tela do TSTView para acompanhamento individual dos alunos. 84

Lista de Tabelas

2.1 Código de um programa e sua representação em *tokens* e árvore. 17
2.2 Exemplo 1 de um código, tokens relacionados e representação em árvore. . 18
2.3 Exemplo 2 de um código, tokens relacionados e representação em árvore. . 19
2.4 Exemplo 3 de um código e árvore relacionada. 20
2.5 Exemplo 4 de um código e árvore relacionada. 20

3.1 Atividade de geração dos números da série de Fibonacci. 27
3.2 Cinco variações de códigos descartadas. 29
3.3 Variações de códigos utilizadas. 29
3.4 Concordância entre professores. 33
3.5 Concordância e valores de Δ entre estratégias automáticas e professores. . . 35

4.1 Questão Média dos Pares. 47
4.2 Questão Média dos Extremos. 48
4.3 Afirmações relacionadas a própria percepção do aluno como estudante de programação. 54
4.4 Afirmações relacionadas ao processo de avaliação de código entre pares. . . 56
4.5 Média de comentários de diferentes categorias para diferentes códigos. . . . 57
4.6 Média de acerto de uma questão (0 a 1). 58
4.7 Número médio de códigos observados pelos alunos divididos em duas categorias de tempo gasto durante a observação. 58
4.8 Afirmações relacionadas à utilização da ferramenta. 59

5.1 Tópico, quantidades de questões associadas e termos relevantes para a definição do tópico 69

D.1 Votos dos professores para cada tripla de códigos 86

E.1 Dados dos alunos no experimento de avaliação de códigos entre pares . . . 88

Capítulo 1

Introdução

1.1 Contextualização

Programação é uma atividade que requer o uso de diferentes capacidades cognitivas fundamentais como a de abstração e raciocínio matemático, a habilidade de processar diferentes informações ao mesmo tempo, de construir processos por analogia, processos com condicionais, e do pensamento em etapas [Pea and Kurland 1984]. Não só tais capacidades são requeridas como é possível desenvolver, através da prática de programação, as habilidades de pensamento analítico inerentes ao pensamento computacional [Wing 2006]. É com este novo arcabouço que o estudante de programação é capaz de construir códigos e de resolver alguns dos problemas computacionais.

Assim, o ensino de programação faz parte essencial dos currículos mais básicos de cursos de Ciência da Computação e de áreas correlatas [Roberts et al. 1999] [Shackelford et al. 2006] desde as propostas curriculares de 1991 [Marshall 2012]. A última proposta curricular vigente da ACM [Sahami et al. 2013] cita que uma disciplina introdutória de programação deve focar em fundamentos da área. Tal base é definida por conceitos que permeiam a atividade de programar. São exemplos destes fundamentos: o conhecimento de conceitos básicos de sintaxe e semântica, estruturas de dados, resolução de problemas, construção e execução de algoritmos.

Com o currículo em mãos, o professor de programação precisa iniciar um diálogo entre a matéria proposta e o aluno no processo de ensino-aprendizagem [Borges, M. A. F. 2000]. Neste processo, o professor atua como mediador, procurando incentivar o

estudante a adquirir novas habilidades e diagnosticando o que cada aluno compreende de programação. O mediador deve também dialogar com o aluno para entender as dificuldades de compreensão e sugerindo caminhos para vencer tais dificuldades [Tirronen and Isomöttönen 2011].

Entretanto, realizar um diagnóstico do que o aluno compreende de programação representa uma das maiores dificuldades do professor em sala de aula [Chamillard and Braun 2000; Ben-Ari 1998]. É difícil identificar quando o aluno pode ter uma idéia consistente de como funciona a base de programação, mas não compreendeu determinado conteúdo, ou ainda, é difícil diagnosticar quando o aluno tem uma compreenção equivocada sobre determinados assuntos. Existem diferentes alternativas para realizar tal diagnóstico, como o acompanhamento em projetos, questões de múltipla-escolha, avaliação entre-pares de estudantes [Lister and Leaney 2003]. Além disso, a avaliação dos artefatos produzidos por estudantes representa outra grande oportunidade de diagnóstico [Clancy et al. 2003].

Em computação, é comum o uso de produção de códigos como principal tarefa direcionada aos alunos [Daly and Waldron 2004]. De posse destes artefatos o professor e alunos avaliam e discutem sobre os códigos produzidos. Ao apontar as soluções mais comuns, erros existentes nos códigos ou sugestões de melhoria o professor está gerando *feedback*, ou seja, "informação comunicada ao aprendiz com o objetivo de modificar seu pensamento ou comportamento para promover a aprendizagem" [Cardoso 2011; Shute 2008]. Ao mesmo tempo, o professor realiza uma avaliação do que foi produzido de forma a gerar um diagnóstico do que o aluno compreende de programação.

Idealmente, o feedback fornecido pelo professor deve ser rápido, de forma que o aluno não repita erros cometidos anteriormente ou que fique sem avançar na resolução de exercícios por muito tempo. Ao mesmo tempo, o feedback precisa ser útil, ou seja, precisa dar informação suficiente para que o aluno possa vencer as dificuldades encontradas. Entretanto, o feedback manual e dialógico dado pelo professor tem um alto custo devido a grande quantidade de códigos que podem ser produzidos pelos alunos. Assim, o professor pode acabar demorando para dar feedback ou dar um feedback rápido mas de baixa qualidade.

Existem alternativas para a produção de feedback rápido com o uso de ferramentas que avaliam automaticamente os artefatos produzidos pelo aluno. Como exemplo, testadores automáticos de código são capazes de produzir informação quase instantânea sobre a

aceitação de um código como possível solução correta para um problema de programação proposto pelo professor [Douce et al. 2005]. Outras propostas automáticas existem, como a descrita por Benford et al. [Benford et al. 1993] onde o feedback indica a qualidade do programa produzido é baseado nas métricas de código como o número de problemas na formatação, nível de complexidade ou presença de más-práticas de programação. O mediador, ao avaliar informações geradas por estas ferramentas, pode fazer uso desta informação para o diagnóstico rápido da situação de um aluno.

Além dos exemplos apresentados anteriormente, existe outra classe de ferramentas que faz uso não apenas da análise de um único código em específico, mas de uma população de códigos para gerar informação relevante. Como exemplo, existe uma abordagem para gerar uma nota automática para um determinado código que se baseia na semelhança deste código com alguma solução correta válida para uma determinada questão. Esta nota serve de feedback para o aluno indicando o quão próxima a solução dele é de uma solução considerada ideal. Ainda, existem ferramentas que avaliam os códigos dos alunos buscando identificar plágios [Robins et al. 2003; Pears et al. 2007] o que auxilia o professor no diagnóstico dos estudantes neste tipo de ação.

A abordagem de utilizar outros códigos para feedback e diagnóstico existe não apenas em abordagens automáticas, mas também em atividades diversas em sala, como na avaliação entre-pares. Neste modelo de avaliação os próprios alunos avaliam mutualmente seus códigos [Sitthiworachart and Joy 2004] dando sugestões de como o código pode ser melhorado para que o mesmo satisfaça a especificação da questão ou que melhore em aspectos qualitativos, como a estruturação do código e nomeação de variáveis.

Porém, tal estratégia demanda um certo grau de maturidade em codificação do aluno, e pode não ser muito eficaz em cursos introdutórios de programação, especialmente quando o aluno não tem informação relevante sobre as diferentes estratégias de soluções. Este processo também não é tão ágil quanto uma estratégia automática, mas pode produzir informação mais rica por permitir que uma pessoa possa acrescentar informação além daquelas obtidas automaticamente.

A técnica de comparação de diferentes soluções de uma única questão, pode também ser utilizada pelo professor para confrontar diferentes soluções de diferentes questões. Nessa situação, busca-se detectar o quão parecidas são as soluções dos

diferentes exercícios propostos de maneira que o professor possa atuar no processo de ensino-aprendizagem [Darling-Hammond 2008] identificando questões que são muito exploradas na lista, pois apresentam soluções parecidas, ou questões que apresentem características únicas quando comparada às demais questões.

Estas duas últimas atividades, a avaliação de códigos entre alunos e a avaliação das diferentes questões, utilizam a capacidade de alunos e professores em comparar códigos e é feita atualmente de forma manual e com pouco amparo ferramental, o que é custoso ao processo de ensino de programação. Entretanto, já existem técnicas automáticas de detecção automática de similaridade de código que são aplicadas em outros contextos, como na detecção de plágio e na geração de feedback para, por exemplo, sugestão de proximidade com uma solução de referência.

1.2 O problema

É comum o uso de técnicas de comparação de código como mecanismo automático para auxiliar no feedback ou no diagnóstico em programação. Entretanto, este uso é limitado a detecção de plágio ou geração de um feedback que indica a proximidade do código com alguma solução de referência. Estes cenários só avaliam: *i*) se determinado par de soluções estão muito próximas, ou; *ii*) qual a distância do código para poucas soluções de referência. Tais trabalhos não avaliam a comparação entre todos os pares de elementos existentes nem se tal comparação é válida quando usada em todo um conjunto de programas.

Considerando a comparação par-a-par de uma população de códigos, tal informação precisa ainda ser apresentada de forma amigável ao professor e ao estudante de forma que os mesmos possam fazer uso desta informação, tanto como forma de feedback como para diagnostico da turma. Assim, não há uma visualização ou estratégia amigável de processamento destas comparações para que os alunos possam obter feedback do processo de ensino de programação ou para que o professor possa realizar um diagnóstico da turma ou de uma lista de exercícios.

1.3 Objetivos

A comparação de programas de forma manual faz parte da prática do professor e aluno durante o aprendizado em programação. O professor intuitivamente busca entender as estratégias adotadas pela turma ou faz uso desse diagnóstico para atuar na turma, focando em determinado conteúdo ou em determinada solução para uma questão. Já o aluno pode fazer uso de tal feedback para identificar diferenças entre a sua maneira de pensar e como os colegas trabalharam para resolver a questão.

É objetivo geral deste trabalho, fazer uso e avaliar estratégias automáticas de comparação de códigos para uma população de programas, indo além do foco em detecção de plágio ou da geração de feedback automático considerando uma solução de referência. Ao fazer uso desta informação, deseja-se construir visualizações de tais comparações e que as mesmas possam ser usadas para enriquecer os processos de aprendizagem. Para atingir tal objetivo, três objetivos específicos foram definidos.

O primeiro objetivo específico deste trabalho é o de avaliar estratégias de comparação de códigos considerando uma população de programas e identificando se, e em que condições e grau, tais estratégias conseguem capturar a noção de comparação de códigos de professores de programação.

O segundo objetivo específico é gerar uma visualização que represente uma comparação de programas de uma mesma questão. Esta visualização deve representar a informação extraída das estratégias automáticas de comparação de códigos e ser fácil de ser utilizada. É objetivo trazer um exemplo de uso desta visualização, especificamente como meio de feedback ao aluno durante um processo de avaliação entre-pares de programas.

O último objetivo é de gerar uma visualização e informação útil da análise das soluções para diferentes questões em termos dos tópicos relativos aos conteúdos que tais exercícios buscam explorar. Também são objetivos, contextualizar o uso dessa visualização no cenário de avaliação de uma lista de exercícios e gerar informação sobre grupos de questões semelhantes através de técnicas de agrupamento de documentos.

1.4 Sumário da tese

É tese deste trabalho que técnicas de comparação de códigos podem ser utilizadas para a comparação geral de uma população de códigos, bem como de que é possível representar tais comparações através de visualizações a serem utilizadas, como exemplo, na geração de feedback ao aluno e de informação que auxilia o diagnóstico do professor sobre os exercícios de programação produzidos.

Assim, esta tese permite novos usos da comparação de código no ensino de programação. Como consequência, são resultados desta tese: *i*) o uso e avaliação de diferentes estratégias automáticas de comparação de código para capturar a estratégia de comparação que professores utilizam; *ii*) o uso dessas estratégias como base para um mecanismo para visualização rápida de diferentes códigos para análise e comparação destes artefatos; e *iii*) o uso dessas mesmas estratégias para avaliar como se comparam e se relacionam as questões de uma lista de exercício. Estes resultados foram obtidos a partir de três estudos.

O primeiro estudo realizado, o de avaliação de estratégias de comparação de códigos, buscou verificar inicialmente se há concordância entre como os próprios professores avaliam a similaridade de códigos. Para isto, utilizou-se de uma base sintética de códigos que tenta considerar diferentes variantes exploradas na literatura. Em seguida, comparam-se diferentes estratégias automáticas e como estas se aproximam das escolhas de similaridade realizadas por professores. Foi possível observar neste estudo que: *i*) os professores apresentaram entre 62% e 95% de concordância entre si; *ii*) as estratégias automáticas apresentaram uma concordância com os professores de pelo menos 75% e até 97%.

O segundo estudo propôs uma visualização de diferentes códigos para um determinado exercício em forma de grafo. Neste grafo, cada nó representa uma solução e cada aresta representa a relação de proximidade entre as soluções. Essa visualização foi testada em uma ativdade de avaliação entre-pares. Com ela, os alunos se sentiram mais aptos à tarefa de avaliação entre pares e consideraram simples o uso de tal visualização.

O último estudo aplica o algoritmo de comparação de códigos a uma lista de exercícios, mostrando resultados que podem ser obtidos a partir da visualização dos dados de similaridade entre diferentes questões. Além da comparação direta entre questões, é possível agrupá-las usando estratégias automáticas de identificações de tópicos, onde cada tópico

representa um conjunto de termos que, em conjunto, representam a idéia da questão.

Assim, a tese aqui apresentada é demonstrada através desses três estudos. Primeiro, torna válido o uso de estratégias automáticas de comparação de códigos para um conjunto de programas, e, em seguida, provê uma visualização das soluções de uma determinada questão. Por fim, as estratégias automáticas de comparação de códigos são utilizadas para identificar tópicos e gerar uma visualização da semelhança entre questões.

1.4.1 Implicações

A avaliação das estratégias automáticas de comparação de código permite que novos trabalhos possam discutir e utilizar tais estratégias considerando aquelas que são mais semelhantes à estratégia do professor para comparar programas. Além disto, este trabalho levanta a necessidade de discutir as diferenças nas estratégias de comparação que os professores podem adotar durante atividades que requerem a comparação de programas.

Em seguida, é proposta uma visualização para exibir a informação das comparações de código. Nesta visualização em forma de grafo é possível identificar códigos muito próximos e soluções distoantes. Esta visualização foi usada na avaliação entre-pares como exemplo de aplicação, mas é possível usar em outros contextos. O professor pode, por exemplo, através desta visualização, identificar a solução com maior número de soluções próximas para levar a discussão da turma sobre esta e outras alternativas de códigos. O professor pode ainda utilizar tal informação para agilizar sua forma de corrigir questões e deixar mais justa a avaliação no sentido de avaliar com nota semelhante soluções semelhantes, já que visualizará mais facilmente os tipos de solução produzidas e suas variações.

Por fim, o último resultado apresentado, o da comparação de questões, inicia uma discussão sobre a própria produção de exercícios que são pedidos ao aluno. O acompanhamento da relação entre questões e dos agrupamentos identificados por elementos sintáticos permitem que o professor possa repensar e refletir sobre a qualidade de sua lista de exercícios.

1.5 Outline da tese

O restante desta tese esta organizado da seguinte forma:

Capitulo 2 Apresenta o arcabouço teórico necessário para o acompanhamento desta tese;

Capitulo 3 Este capítulo explora a avaliação de como professores comparam códigos, primeiro contextualizando o leitor com uma descrição geral das estratégias de similaridade adotadas neste trabalho, apresentando o design do experimento adotado para mostrar então os resultados obtidos e toda a discussão existente. Todo processo de design experimental é descrito através das etapas de descrição das questões de pesquisa, da seleção de amostra do experimento, da etapa de coleta de dados, processamento de dados e descrição para a replicação do experimento;

Capítulo 4 No Capítulo 4, uma visualização para comparações de programas através de um grafo é sugerida, e seu uso é avaliado em uma atividade de avaliação de códigos realizada entre alunos, utilizando uma ferramenta Web para permitir tal visualização. Este uso é avaliado através de um experimento que segue as mesmas etapas bem definidas utilizadas no capítulo anterior;

Capítulo 5 Este capítulo apresenta uma proposta de identificação de tópicos e de visualização da similaridade entre diferentes questões se baseando nas soluções dos alunos através de estratégias de comparação de códigos. Apresenta também um estudo qualitativo sobre o uso dessa visualização e identificação numa lista de exercícios.

Capítulo 6 Por fim, apresenta-se uma discussão geral sobre o trabalho, bem como as conclusões e trabalhos futuros desta tese.

1.6 Publicações

Este trabalho de tese resultou diretamente em duas publicações em conferências de renome, sendo o primeiro trabalho referente ao estudo sobre como professores comparam códigos e o segundo trabalho sobre a identificação de tópicos e a comparação de questões de uma lista de exercícios:

- Matheus Gaudencio, Ayla Dantas, and Dalton D.S. Guerrero. 2014. Can computers compare student code solutions as well as teachers?. *Proceedings of the 45th ACM technical symposium on Computer science education* (SIGCSE

'14). ACM, New York, NY, USA, 21-26. DOI=10.1145/2538862.2538973 http://doi.acm.org/10.1145/2538862.2538973

- Gaudencio, M.; Dantas, A.; Guerrero, D. D. S. 2013. Análise Automática de Exercícios de Programação como Forma de Avaliar a Cobertura de Tópicos da Disciplina. *XXIV Simpósio Brasileiro de Informática na Educação, 2013. v. 1.* (SBIE 2013). Maceió, AL.

Além destas publicações, são resultados do trabalho de doutorado publicações em outras três vertentes de pesquisa no ensino de programação. Na primeira, o *Webshell*, busca-se prover ao aluno um ambiente de programação web em que o professor possa extrair informação útil com o passo-a-passo do processo de codificação do estudante. Na segunda linha de pesquisa, sobre o hábito de estudo dos alunos, avaliam-se características da frequência de produção de códigos numa lista de exercício e como podem ser caracterizados os alunos por este aspecto. O último estudo avalia o uso de uma ferramenta para acompanhamento das atividades do aluno seja no momento atual de aula ou avaliando o histórico do mesmo. São estes trabalhos, em ordem:

- Rodrigues, J. R. R.; Gaudencio, M.; Guerrero, D. D. S. 2013. Compreendendo o Processo de Codificação de um Exercício de Programação em Python. *WEI - XXI Workshop sobre Educação em Computação*. (CSBC 2013). Maceió, AL.

- De Araújo, E. C.; Gaudencio, M.; Menezes, A; Ferreira, I.; Ribeiro, I. Fagner, A.; Ponciano, L.; Morais, F.; Figueiredo, J. C. A.; Guerrero, D. D. S. 2013. O Papel do Hábito de Estudo no Desempenho do Aluno de Programação. *WEI - XXI Workshop sobre Educação em Computação*. (CSBC 2013). Maceió, AL.

- Gaudencio, M.; Wanderley, L. F.; Lemos, F. W.; De Araújo, E. C.; Figueiredo, J. C. A.; Guerrero, D. D. S. 2013. Eu Sei o que Vocês Fizeram (Agora e) na Aula Passada: o TSTView no Acompanhamento de Exercícios de Programação. In *XXIV Simpósio Brasileiro de Informática na Educação, 2013. v. 1.* (SBIE 2013). Maceió, AL.

Em comum com este trabalho de tese, essas pesquisas exploram o ensino de programação. O Webshell e TSTView também têm como características em comum o uso

de estratégias automáticas para provêr informação útil a professores e alunos. Estas três pesquisas são descritas em maiores detalhes nos Anexos A, B e C.

Capítulo 2

Fundamentação Teórica

Dentro do grande contexto do ensino de programação, este trabalho oferece uma proposta teórica e de ferramentas para apoiar o ensino-aprendizagem de programação por meio da comparação automática de soluções de códigos de alunos. Esta informação pode ser usada no feedback e diagnóstico durante o processo de ensino de programação.

Para a realização deste trabalho, faz-se necessária a fundamentação teórica em três áreas distintas. A primeira área é a grande área de ensino de programação. A segunda área é a da geração de feedback sobre o código, especialmente para o aluno . A última área é a de comparação de códigos e como a mesma é utilizada na literatura.

2.1 Ensino de Programação

Por ser disciplina fundamental, o ensino de programação é discutido amplamente na literatura. Lemos [Lemos 1979] escreveu o primeiro levantamento (*survey*) da área apresentando dez diferentes técnicas de ensino de programação utilizadas em 306 cursos de gerência de negócios e em computação da época. Em 1987, outro grande levantamento foi escrito por du Boulay e Sothcott [du Boulay and Sothcott 1987] desta vez com foco em sistemas automáticos de tutores e ferramentas de teste para o auxílio no ensino de programação. Em 2003, Robins et al. [Robins et al. 2003] apresentou uma revisão da área no tocante a: diferenças do ensino de programação para programadores iniciantes e veteranos, estratégias de programação, criação e compreensão de programas, e nas diferenças entre programação orientada a objetos e programação procedural.

Entretanto, apenas em 2007 há registro na literatura de um survey buscando realizar o mapeamento e compreensão geral da grande área de ensino de programação. Este artigo publicado em 2007, por Pears et al. [Pears et al. 2007] apresenta uma discussão sobre o que há de mais relevante na área até a época de sua publicação. Em especial, há a identificação de 4 sub-áreas de pesquisa no ensino de programação. São estas: *curricula*, pedagogia, escolha de linguagem de programação, e ferramentas.

O foco deste trabalho de doutorado está na construção do arcabouço teórico e de ferramentas para comparação de programas. Considerando as ferramentas produzidas por este trabalho, é importante considerar o uso destes utilitários para oferecer feedback aos alunos e diagnóstico ao professor sobre os programas produzidos pelos alunos. A próxima seção discute os aspectos de feedback e diagnóstico a serem considerados como implicações do resultado deste trabalho. Em seguida, apresenta-se o referêncial teórico sobre comparação de código que serve de base para a construção teórica deste trabalho.

2.2 Feedback e Diagnóstico

O trabalho de Pears et al. [Pears et al. 2007] identifica a presença massiva de ferramentas no auxílio do processo de ensino-aprendizagem. Tais ferramentas buscam auxiliar o professor e gerar feedback a ser utilizado pelo aluno. Neste contexto, Cardoso [Cardoso 2011] discute sobre as diferentes visões de feedback possíveis, especificamente de ferramentas on-line. Dentre as definições trabalhadas, considera-se para esta tese uma definição próxima a de Shute [Shute 2008].

De acordo com Cardoso, Shute define feedback como "informação comunicada ao aprendiz com o objetivo de modificar seu pensamento ou comportamento para promover a aprendizagem". É possível ocorrer tal comunicação em diferentes etapas de uma aula. Considerando o ensino de programação, Chamillard e Braun [Chamillard and Braun 2000] apontam que a realização de exercícios que necessitam a construção de um código é uma prática comum no ensino de programação, especialmente através do uso de laboratórios práticos e de listas de exercícios. Durante estas atividades, o aluno desenvolve programas a serem avaliados gerando feedback ao aluno e informação de diagnóstico ao professor.

Este processo de feedback pode se dar de forma automática, como exemplo, por meio

da avaliação automática da solução provida por meio de testes automáticos. Na proposta de Petit et al. [Petit et al. 2012], o professor e aluno pode receber a informação a respeito de um programa submetido ter sido bem sucedido ou não nos casos de teste para uma determinada questão. Caso os testes executados não apresentem falhas, o aluno pode garantir que a sua solução faz o que se pede. Entretanto, não necessariamente o programa apresenta a melhor estrutura, clareza ou legibilidade para a resolução da questão. Em caso de falha, o aluno pode apenas ter cometido um erro de digitação e pensar que toda a idéia por trás da sua solução é falha.

Para oferecer mais informações além teste sobre programas produzidos, há propostas para auxiliar na criação manual de feedback. Bancroft e Roe [Bancroft and Roe 2006] apresentam um sistema integrado de submissões e anotações (comentários) de exercícios que permite professores e monitores de disciplina comentarem sobre o código submetido e, assim, darem feedback ao aluno. Entretanto, esta anotação não é imediata e depende da capacidade do avaliador produzir comentários de qualidade sobre o código dos alunos.

Para agilizar esse processo de anotação e prover uma avaliação manual rápida, Cummins et al. [Cummins et al. 2010], construiu uma plataforma onde cada avaliador pode inserir marcações (tags) no código do aluno. Tais marcações podem representar erros comuns, como MAU_AREJAMENTO ou CÓDIGO_MORTO. Pelas marcações serem curtas e simples, este tipo de anotação é mais rápida de ser produzida do que a inserção de comentários. Entretanto, ainda é necessário que haja a dedicação do avaliador para preencher tais anotações. Além disso, há o retrabalho do professor/monitor de prover as mesmas anotações para vários códigos muito semelhantes.

Neste trabalho, o ferramental produzido procura prover informação útil sobre como se relacionam as diferentes soluções de uma questão e as diferentes questões de uma lista. Esta informação pode ser usada pelo professor para gerar diagnóstico sobre a turma ou entrar como forma de feedback ao aluno. Este feedback é automático e instantâneo. Para isto, formas automáticas de comparação de programas foram avaliadas, adaptadas e consideradas para gerar dados e visualizações úteis ao professor e ao aluno. Na próxima seção, são apresentadas as diferentes estratégias de comparação de programas, visualização de diferenças de similaridade e agrupamento de programas através de análise semântica de texto.

2.3 Comparação de Códigos

Focando especificamente na detecção de código-fonte duplicado, Roy e Cordy [Roy and Cordy. 2007] apresentaram em 2007 um survey sobre o tema e identificaram seis maneiras distintas para a classificação de algoritmos de detecção de similaridade de código:

Caracteres Detecção baseada na similaridade textual (caracteres) de códigos;

Tokens Neste modelo de detecção, os tokens (elementos léxicos e estruturais de um programa) são extraídos dos códigos e estes são utilizados para a detecção da similaridade;

Árvores Sintáticas O programa é interpretado como uma árvore abstrata e a detecção é realizada através de uma análise de semelhança de árvores;

Grafo de dependência (Program Dependency Graph) O programa é visto como um grafo que representa seu fluxo de execução e a detecção é feita através da comparação dos grafos gerados;

Métricas Códigos são comparados a partir das similaridades e diferenças entre métricas extraídas do código, como complexidade ciclomática, linhas de código, número de variáveis, entre outras;

Híbrida Na abordagem híbrida, dois ou mais modelos de detecção são utilizados. Como exemplo, é possível realizar uma comparação por semelhança de tokens e pela métrica do número de linhas de código dando pesos diferentes para cada um destes dois aspectos.

É importante observar que embora a comparação de código através de grafos de dependência e métricas sejam ricas pela realização da avaliação funcional do código, estas técnicas não são diretamente aplicáveis para a comparação de códigos de um curso introdutório de programação. Isto ocorre porque tais abordagens requerem um código fortemente estruturado em funções e nas chamadas a tais funções para ser capaz de diferenciar códigos [Ohmann 2013]. Entretanto, este tipo de caracteristica, não é comumente encontrado em questões introdutórias de programação.

Estas técnicas já foram bastantes exploradas na literatura dentro do contexto de detecção de plágio. O primeiro trabalho na área, escrito por Donaldson [Donaldson et al. 1981] cita o uso de métricas de software para a comparação e detecção de plágio. Outros trabalhos continuaram a explorar o processo de detecção de plágio como apresentam os surveys de Robins [Robins et al. 2003] e Pears [Pears et al. 2007] além da pesquisa realizada por Hage et al. [Hage and P. Vugt 2010] para a comparação de ferramentas de detecção de plágio.

Entretanto, toda pesquisa em detecção de plágio apresenta uma limitação quanto à avaliação do uso das técnicas de comparação de códigos. Enquanto as submissões para uma mesma questão são confrontadas par-a-par para a detecção de similaridade, estes trabalhos avaliam apenas a similaridade de pares de códigos que estão no limiar para serem considerados plágios.

Por exemplo, dadas 3 soluções, A, B e C, a pesquisa na área de plágio busca identificar se os pares AB, AC ou BC representam códigos plagiados. Enquanto métricas de similaridade podem ser usadas para identificar quais destes pares são plágios, tais trabalhos não avaliam qual a relação de similaridade entre esses códigos quando os mesmos não são considerados plágios. Numa proposta de construção e avaliação de uma ferramenta colaborativa para análise de códigos, o CoMoTo [Meyer et al. 2011], há menção da necessidade de realização de um estudo dessa natureza, bem como seu potencial para auxiliar o professor em sala de aula.

A pesquisa em avaliação automática é uma área que também faz uso da detecção de similaridade para a comparação de distâncias do código do aluno para uma ou mais soluções de referência [Ihantola et al. 2010; Saikkonen et al. 2001; Naudé 2007]. Nesta área, é comum que uma solução proposta pelo aluno seja avaliada pela sua proximidade com uma solução de referência e que esta proximidade defina uma nota como forma de feedback dada ao estudante.

No entanto, essa nota calculada de forma automática pode não representar bem a qualidade de um código produzido, especialmente quando tais soluções estão distantes das soluções de referência [Rahman et al. 2008]. Apesar de que ainda existe correlação entre o que é produzido automaticamente e manualmente [Sherman et al. 2013].

Assim, a primeira limitação existente dos trabalhos discutidos está no não uso e não avaliação da informaçao da similaridade entre códigos para todos os códigos existentes

de uma população. A segunda limitação está na comparação de similaridade apenas com determinadas soluções, consideradas de referência. Este trabalho de tese busca avaliar a comparação de código considerando toda uma população de soluções, além de oferecer um novo ferramental de uso dessa informação para diferentes cenários, como a visualização de soluções de uma questão e da relação entre diferentes questões de uma lista de exercício. Para tanto, a próxima subseção explica estratégias atuais de comparação de similaridade de códigos.

2.3.1 Medidas de Similaridade

A estratégia mais simples de comparação de dois códigos é feita a partir da escolha de uma característica que é comum a ambos e da contagem do número de vezes que tal característica está presente nestes dois elementos. Por exemplo, a contagem de palavras em comum entre dois códigos oferece uma estratégia bastante simples de detecção de similaridade. Considerando duas cadeias, "A B C D" e "C D E F", a simples contagem de palavras em comum retorna uma semelhança de 2 elementos ("C" e "D").

Entretanto, a contagem absoluta de características semelhantes pode não ser uma medida significativa quando os códigos comparados diferem em tamanho. Por exemplo, sejam dois códigos "A B C D . . . X W Y Z" e "A B C D". Eles apresentam 4 características semelhantes. Este mesmo valor de semelhança existe entre os códigos "A B C D" e "A B C D". Assim, essa contagem absoluta de características, não é um bom representante da similaridade entre códigos por não incorporar a medida de diferença de tamanho entre os códigos. Como alternativa, o uso da similaridade relativa representa um valor mais fiel para a representação de similaridade. Esta similaridade é o valor de características semelhantes entre dois códigos sobre o total de características existentes. Nos exemplos citados anteriormente, o código "A B C D" teria uma similaridade de aproximadamente 15% $(4/26)$ em relação a um código formado pelo alfabeto, mas uma similaridade de 100% consigo mesmo.

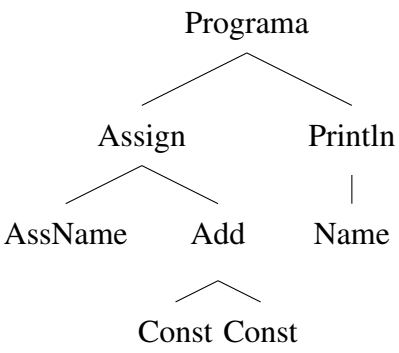
Assim, o uso da similaridade relativa ajuda a considerar a comparação de códigos dentro de uma população que apresenta tamanhos diferentes . Entretanto, determinar qual a característica a ser utilizada é um dos aspectos mais importantes para a comparação de similaridade de códigos. Existem diferentes estratégias de escolhas de características possíveis, entre elas, características sintáticas, semânticas, etc.

São características sintáticas aquelas que representam construções básicas para a criação de um código, como símbolos, regras de gramática e palavras-chave. Construções semânticas consideram o significado de um elemento sintático no código. Como exemplo, uma variável X pode assumir determinado tipo de valor e comportamento para um trecho de código, mas assumir outro tipo de valor e comportamento em outro trecho de código. Ou seja, estas características inerentes a execução do código e seu contexto são características semânticas.

Das alternativas apresentadas, a detecção de similaridade sintática é a mais simples e rápida sendo útil na avaliação de códigos pequenos, devido a pouca natureza de elementos extra sintáticos que sejam diferenciais. Assim, ao utilizar algoritmos de detecção de similaridade que usam características sintáticas estes trabalham com os *tokens* da linguagem, bem como a árvore sintática abstrata dos códigos. Um token representa um conjunto de caracteres que têm significado quando interpretados em conjunto pela linguagem e uma árvore sintática abstrata representa a estrutura sintática do código e a relação entre estes tokens. Na Tabela 2.1 é possível ver um programa e sua representação em forma de tokens existentes e no formato de uma árvore sintática abstrata.

Tabela 2.1: Código de um programa e sua representação em *tokens* e árvore.

Código	Linha do código e Token presente	Árvore

`a = 1 + 2`
`print a`
	`1 Assign`
	`1 AssName`
	`1 Add`
	`1 Const`
	`1 Const`
	`3 Println`
	`3 Name`
		

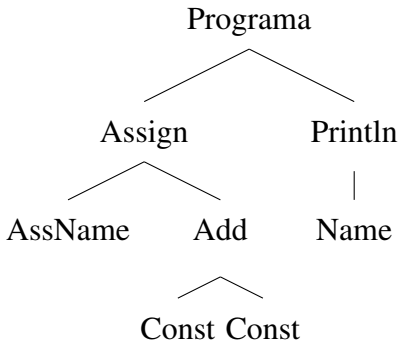
Estratégias de comparação sintática de códigos

Escolhida a característica relevante, existem diferentes estratégias que podem ser adotadas para a comparação de códigos. Sete algoritmos de detecção de similaridade foram usados nesta pesquisa. A primeira estratégia é a distância de Jaccard [Tan et al. 2005]. Para o cálculo da distância de similaridade, é preciso extrair de cada código, o multiconjunto dos tokens do código. Cada multiconjunto é definido pelo agrupamento, sem preocupação com a ordem, de cada token existente no código, incluindo as repetições do mesmo. Dados dois códigos X e Y, a distância de similaridade, ou simplesmente $Score(x, y)$, é calculada pela razão entre a interseção e a união dos multiconjuntos de tokens dos códigos X e Y. Como exemplo, os código da Tabela 2.2 e da Tabela 2.3, quando comparados pela distância de Jaccard, apresentam um $Score$ de $\frac{\|\{\text{Const,Const,Println}\}\|}{\|\{\text{Add,Assign,AssName,Const,Const,Println,Name,Mul}\}\|} = \frac{3}{8}$ ou seja, $0,375$.

Tabela 2.2: Exemplo 1 de um código, tokens relacionados e representação em árvore.

```
a = 1 + 2
print a
```

```
1 Assign
1 AssName
1 Add
1 Const
1 Const
3 Println
3 Name
```



MultiConjunto =

{Add, Assign, AssName, Const, Const, Println, Name}

Tabela 2.3: Exemplo 2 de um código, tokens relacionados e representação em árvore.

`print 1 * 2`	`1 Println`
	`1 Mul`
	`1 Const`
	`1 Const`
		Programa
		|
		Println
		|
		Mul
		Const Const

MultiConjunto = $\{\text{Const}, \text{Const}, \text{Mul}, \text{Println}\}$		

A segunda estratégia avaliada faz uso da edição de texto (distância de Levenshtein) [Manning et al. 2008a] para o cálculo do $Score$. A distância de Levensthein é calculada como a quantidade mínima de operações (o_p) de inserção, remoção ou alteração de tokens de dois textos para que estes fiquem iguais. Quando dois códigos são completamente distintos, o número necessário de operações para que o dois textos x e y fiquem identicos é de $\max(|x|, |y|)$, ou seja o número de tokens do maior texto. Para normalizar o valor da distância considerando os diferentes tamanhos de código possíveis, calcula-se esta distância normalizada como $\frac{o_p}{\max(|x|,|y|)}$. Para gerar uma medida de similaridade, o $Score$, basta calcular o complemento para 1 da distância normalizada, ou seja, $Score$ é calculado por $1 - \frac{o_p}{\max(|x|,|y|)}$. Para o exemplo anterior, o valor de similaridade destes códigos seria de $1 - \frac{5}{7}$, ou seja, $0,28$.

No cálculo da similaridade por tokens, a ordem destes não importava no cálculo de similaridade. Na similaridade baseada na edição de texto, a ordem dos tokens tem impacto no valor calculado. Essas duas estratégias no entanto não consideram que determinados tokens estão agrupados sob uma determinada hierarquia em uma árvore abstrata que representa a sintáxe do programa. É possível comparar calcular o quão distantes duas árvores estão entre-si de forma parecida com a edição de texto, considerando uma operação a mais, a de troca de nós na árvore. Cada troca conta como 1 operação, mas cada nó leva também seus respectivos filhos durante a troca. Esta estratégia descreve o cálculo da distância de edição

de árvores de acordo com Zhang-Shasha [Zhang and Shasha 1989].

A Tabela 2.4 apresenta um código que se comparado com o código da Tabela 2.5, apresenta valor 1 pela distância de Jaccard. Já pela distância da edição de texto apresentam similaridade de $2/6$ e na distância de árvores apresenta um valor de score de $5/6$ (1 operação de troca de nós da árvore sintática geraria códigos idênticos). Considerando estes três resultados, é possível observar que a similaridade calculada pela distância de Jaccard ignora por completo a ordem dos tokens. Já a distância de texto penaliza em maior grau a ordem dos tokens, enquanto a similaridade baseada em distância de árvores considera a ordem dos tokens, mas é flexível no cenário em que uma troca dos nós (com seus filhos) tornam as árvores idênticas.

Tabela 2.4: Exemplo 3 de um código e árvore relacionada.

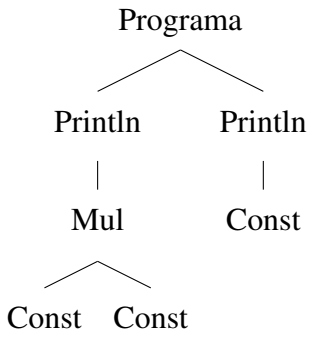
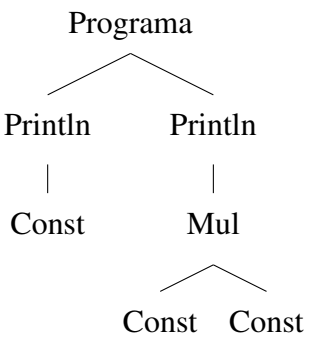
`print 1 * 2` `print 1`	
MultiConjunto $= \{\text{Const}, \text{Const}, \text{Const}, \text{Mul}\}$	

Tabela 2.5: Exemplo 4 de um código e árvore relacionada.

`print 1` `print 1 * 2`	
MultiConjunto $= \{\text{Const}, \text{Const}, \text{Const}, \text{Mul}\}$	

As quatro estratégias restantes se baseiam em variações dos algoritmos de similaridade baseados na frequência de termos. Em cada uma dessas variações, há uma contagem da frequência dos termos (tokens) que aparecem nos códigos de forma a gerar um vetor (TF) que é comparado, para cada código, através da distância dos cossenos destes vetores. O vetor de frequência é uma tupla de N dimensões, uma para cada termo existente, com o valor de frequência daquele termo para tal documento. Assim, comparar a similaridade de dois códigos é tão simples quanto calcular a distância de cossenos entre os dois vetores.

Na literatura, estas estratégias são conhecida como TFxIDF ou simplesmente, TFIDF. Na abordagem tradicional de uso deste algoritmo, cada TF é multiplicado pela frequência inversa de cada termo (IDF) nos documentos, de forma que os termos de menor frequência (raros) em todos os documentos tenham maior peso no cálculo final do vetor de TFIDF. Assim, um termo muito frequente num documento quase não tem peso na TFIDF se ele for igualmente frequente nos demais documentos.

Existem diferentes formas de calcular tanto a TF como a IDF para o cálculo da similaridade. Para denotar as diferentes estratégias existentes, utiliza-se a notação SMART [Manning et al. 2008b], onde cada medida de similaridade (TFIDF-ABC) é definida por 3 letras (ABC): a primeira representa a estratégia de cálculo da TF, a segunda da IDF e a terceira a estratégia do cálculo de similaridade.

Quatro diferentes estratégias são relevantes para comparação de código. São estas: 1) TFIDF-NNC: contagem normal dos tokens; 2) TFIDF-ANC: contagem normalizada pelo token mais frequente de todos os documentos; 3) TFIDF-BNC: contagem binária dos tokens, considerando assim que a frequencia de um termo que aparece uma ou mais vezes nos códigos é 1; e, 4) TFIDF-LNC: onde o vetor é gerado a partir do logaritmo da frequência dos termos. Em todos os 4 cenários, a parte IDF do algoritmo, isto é, a parte que pondera a importância do termo considerando sua presença ao longo dos documentos, é ignorada na medida que a comparação só importa na avaliação par-a-par.

2.3.2 Visualização de Questões Similares

Dada uma estratégia de identificação de similaridade de códigos, é possível identificar o quanto dois códigos se aproximam. Entretanto, esta informação não é facilmente processada por uma pessoa que se depara com uma população grande de códigos a serem analisados.

Uma das maneiras de visualizar a proximidade de diferentes entidades dada uma relação de semelhança entre elas, é através da árvore filogenética. Esta é uma árvore binária em que as entidades são mais semelhantes na medida que apresentam um ancestral comum mais próximo. Tal técnica é utlizada na identificação de componentes principais de software [Sanches et al. 2011] e na visualização de dados educacionais [Moro et al. 2013].

O algoritmo desenvolvido por Saitou e Nei [Saitou and Nei 1987] é uma estratégia utilizada para a construção dessas árvores. Para tanto, dada uma população de entidades, constrói-se uma matriz de similaridade $N \times N$, onde cada linha representa um código, e cada coluna é o espelhamento dos códigos das linhas. Cada elemento da matriz é preenchido com o valor de similaridade entre as entidades representadas em cada linha e coluna. Usando esta matriz, é gerada uma nova matriz Q que identifica, para cada par de entidades, o quão distante estes dois elementos estão entre si, considerando a distância de cada entidade para os demais elementos. O par de menor distância é agregado na matriz Q, de forma que estes nós passam a ter um ancestral em comum. Depois calcula-se o novo elemento de menor distância, considerando apenas o nó ancestral gerado.

A Figura 2.1 mostra um exemplo desta estratégia de visualização em uso. Nesta visualização, os códigos com uma diferença menor do que um determinado limite de similaridade foram agrupados no mesmo ramo. Com este tipo de visualização é possível identificar quais as estratégias mais comuns e como tais estratégias gerais diferem entre si. O aluno pode tomar conhecimento de em qual grupo sua solução se encaixa e quais as estratégias mais aplicadas na resolução do problema.

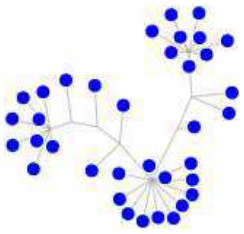


Figura 2.1: Exemplo de visualização da comparação de códigos utilizando a distância de Jaccard e $\Delta = 0,05$.

2.3.3 Agrupamento Semântico de Termos

Fazer a análise de questões através da similaridade de diferentes questões é simples e direta, mas não necessariamente é bastante para dar ao professor uma visão maior sobre o que está sendo avaliado em uma lista de exercícios. Para isso, uma segunda estratégia pode ser adotada, explorando também comparação de códigos: a de identificação de tópicos. Um tópico é um tema que identifica um determinado conteúdo. Por exemplo, uma questão poderia ter "*while*" ou "estruturas de dados avançadas" como tópicos.

A identificação de tópicos é um tema explorado em diferentes comunidades científicas, sendo usado, por exemplo, na extração de tópicos a partir de documentos de referência para a construção de mapas conceituais [Dalmolin et al. 2009]. Em computação, o primeiro uso dessa técnica ocorreu com o objetivo de identificar, dentro de um programa, quais os diferentes tópicos que estavam associados a um conjunto de classes [Maletic and Marcus 2000]. Nesse trabalho, os autores fizeram uso de uma técnica conhecida como Idexação Semântica Latente (LSI, originalmente chamada de Análise Semântica Latente – LSA). Nessa técnica, os termos são avaliados de forma a identificar quais são aqueles que aparecem frequentemente nos mesmos documentos. Ao identificar os termos comumente associados a determinadas classes, os autores identificaram determinada funcionalidade existente dessas entidades, como exemplo, classes que tratam da GUI por usarem expressões comuns de interface gráfica.

Uma das fragilidades da LSI é estar propensa à ambiguidade dos termos. Um termo como "semáforo" pode acabar agregando documentos tanto de computação como de engenharia de trânsito. Mais recentemente, Blei [Blei et al. 2003] propôs uma técnica, a Alocação Latente de Dirichlet (LDA), que refina a LSI ao fazer com que cada termo de um tópico tenha seu comportamento definido por uma distribuição de Dirichlet.

Na LDA, são definidas de antemão a quantidade de tópicos a serem extraídos. Cada tópico, é inicializado como uma *simplex*, ou seja, uma tupla de n valores de soma 1. Cada elemento desta tupla está associado a um termo do vocabulário dos documentos avaliados e, a cada iteração da LDA, os pesos dos valores das tuplas dos tópicos são alterados de forma a identificar a distribuição mais adequada de termos em cada tópico.

Como resultado, a LDA permite identificar tópicos com expressões utilizadas em diferentes contextos. Por exemplo, na identificação de tópicos de exercícios de Python

utilizando tokens da linguagem, um termo como SUBSCRIPT (acesso a item de uma estrutura) pode aparecer em conjunto com o operador de concatenação de strings para determinado tópico ou com operadores que atuam sobre listas em outro tópico.

Capítulo 3

Comparação Automática e Manual de Programas

Este capítulo propõe uma estratégia de uso de algoritmos automáticos de comparação de código e apresenta um experimento que avalia o quão próximas estão tais estratégias umas das outras e da maneira como professores comparam códigos.

3.1 Contextualização

No Capítulo 2, diferentes estratégias automáticas de comparação de códigos foram apresentadas. Para gerar a informação de similaridade, este trabalho faz uso de elementos sintáticos pois estes são facilmente extraídos, ignoram a apresentação textual do código como a forma de espaçamento ou nome de variáveis, e são simples de serem compreendidos e bem usados na literatura.

Para gerar a medida de similaridade, escolhe-se um valor relativo de similaridade, entre 0 e 1, representando dois códigos completamente diferentes a dois códigos com características sintáticas idênticas. Esta medida relativa representa melhor o nível de similaridade de dois códigos, independente do número total de características existentes em cada código.

Sete estratégias, todas apresentadas no Capítulo 2, são utilizadas aqui: 1) similaridade baseada na distância de Jaccard. 2) similaridade baseada na edição de texto; 3) similaridade baseada na edição de árvore; 4) TFIDF-NNC; 5) TFIDF-ANC; 6) TFIDF-BNC; 7) TFIDF-LNC. Tais algoritmos são bem descritos e utilizados na literatura de comparação

de códigos.

Para avaliar como os professores comparam códigos, construiu-se um experimento com uma base bem definida de códigos sinteticamente criados. Os professores receberam, por cada rodada, um código de referência e um par de códigos de teste e escolheram qual destes dois códigos de teste mais se assemelhavam ao código de referência, ou diziam que os dois códigos eram igualmente distintos/semelhantes a esse código.

De posse das escolhas dos professores, buscou-se entender como as estratégias automáticas fariam tais escolhas. Para tanto, este trabalho definiu um Δ, que representa um limiar na qual as estratégias automáticas passam a considerar dois códigos como igualmente distantes. O uso desse fator para a avaliação de similaridade de códigos é uma das contribuições deste trabalho e permite que as estratégias automáticas se aproximem da maneira que professores comparam programas.

A próxima seção discute o design deste experimento em detalhes, bem como resultados obtidos e conclusões parciais.

3.2 Design Experimental

O primeiro experimento deste trabalho de tese busca atender o objetivo de avaliar estratégias de comparação de códigos considerando uma população de programas e identificando se, e em que condições e grau, tais estratégias conseguem capturar a noção de comparação de códigos de professores de programação.

Quatro subseções bem definidas descrevem este design experimental: 1) Questões de pesquisa, contendo as questões a serem respondidas para atingir o objetivo em questão; 2) Amostra do experimento, indicando como foi feita a seleção amostral do experimento; 3) Coleta de dados, indicando os procedimentos para coleta dos dados, e; 4) Processamento de dados, descrevendo como os dados coletados são processados para serem avaliados.

3.2.1 Questões de Pesquisa

Duas questões de pesquisa foram levantadas para este experimento. Estas questões buscam analisar a viabilidade do uso de estratégias automáticas para a comparação de código.

Q1 Há concordância entre os professores na avaliação de similaridade questões?

Q2 Há concordância entre os professores e estratégias automáticas na avaliação de similaridade?

3.2.2 Amostra do Experimento

Professores

Essas questões de pesquisa foram testadas considerando um conjunto de 11 professores, sendo estes da Universidade Federal de Campina Grande, Universidade Federal da Paraíba e do Instituto Federal do Amazonas. A base de códigos avaliada fez uso da linguagem de programação Python. Nove professores selecionados já haviam ministrado uma disciplina de ensino de programação usando Python e dois professores ensinaram disciplinas de programação em outras linguagens. Todos os professores tinham algum domínio sobre a linguagem Python.

Códigos Sintéticos

A primeira parte da avaliação buscou identificar se há consenso na forma como os professores comparavam códigos dos alunos. Esta comparação busca identificar quais soluções seguiram estratégias semelhantes ou distintas entre si. Para tanto, construiu-se uma base de códigos baseando-se em uma atividade clássica de programação: a geração dos números da série de Fibonacci, apresentada na Tabela 3.1.

Tabela 3.1: Atividade de geração dos números da série de Fibonacci.

Dada a série de Fibonacci (0,1,1,2,3,5,8, ...), onde o número de cada posição maior ou igual a 2 é definido pela soma dos dois números anteriores, faça um programa que receba duas posições de entrada, a e b, onde a >= 2 e b > a e que imprima todos os números de Fibonacci nas posições de a até b (incluindo a e b). Exemplo:

Entrada	Saída esperada
2	1
3	2

Para tanto, três estratégias distintas para a resolução deste exercício foram adotadas. Em comum, todas as estratégias fazem o uso de uma função `main` que é responsável pela impressão da saída esperada. Na estratégia **A**, o valor do número de Fibonacci é calculado diretamente nessa função. Na estratégia **B** uma nova função, `fib`, é adicionada ao código e era responsável pelo cálculo iterativo do número de Fibonacci. Na estratégia **C**, a função `fib` também existe para realizar este cálculo, mas o faz de modo recursivo. A solução básica destas três estratégias é exibida na Figura 3.1 abaixo.

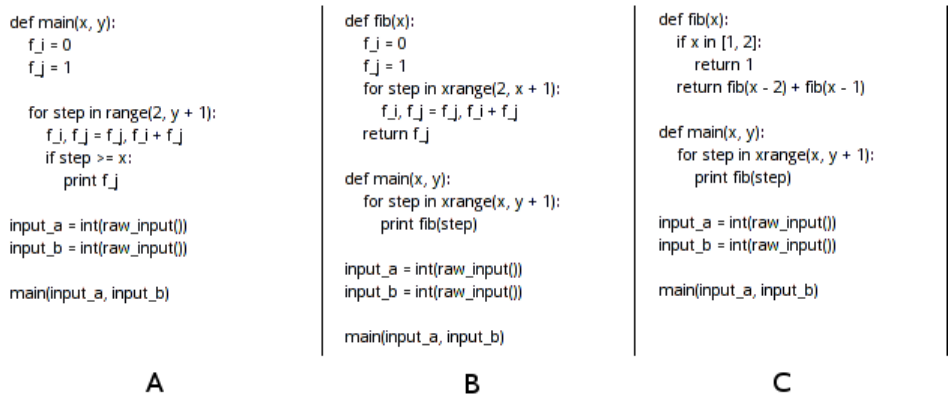


Figura 3.1: As três soluções de base para o cálculo do número da série de Fibonacci. (a) sem função, (b) função iterativa, (c) função recursiva

Enquanto estas três estratégias são particularmente distintas entre si, os alunos não necessariamente as codificam da mesma maneira. Diferenças no processo de codificação podem gerar códigos que, para os professores, representam estratégias únicas. Para permitir uma melhor comparação, foram criados mutantes dessas soluções baseando-se na pesquisa de Roy [Roy 2009] sobre cenários de avaliação de ferramentas de detecção de código duplicado. Foram descartadas as variações descritas na Tabela 3.2 por não serem válidas em Python (linguagem dinamicamente tipada) ou válidas na comparação de códigos por elementos sintáticos. Foram mantidas as variações descritas na Tabela 3.3.

Tabela 3.2: Cinco variações de códigos descartadas.

var	**descrição**
s1b	Adição de um comentário no código
s1c	Mudança da posição do símbolo de começo de bloco
s2c	Mudança no tipo declarado das variáveis
s4a	Troca da posição de comentários do código
s4c	Troca de duas linhas, sendo uma delas uma invocação a uma função com efeito colateral

Tabela 3.3: Variações de códigos utilizadas.

var	**descrição**
s0a	Solução base
s1a	Uso de identação com o dobro de espaços da solução base
s2a	Nome de variável alterada
s2b	Alteração no nome da variável e na ordem dos parâmetros da função
s2d	Adição de uma operação aritimética sem efeito no código
s3a	Novo parâmetro adicionado a uma função
s3b	Um parâmetro removido da função (o parâmetro passa a ser uma tupla)
s3c	Adicionado um comando condicional que sempre será avaliado como verdade
s3d	Duas linhas combinadas em uma
s3e	Adicionado um comando condicional
s4b	Troca de duas linhas de código sem efeito colateral
s4d	Uso de um “while” no lugar de um “for”

As doze variações da Tabela 3.3, incluindo o código base, resultam em 36 códigos para avaliação pelos professores. Todos estes códigos satisfazem a especificação da questão apresentada.

Algoritmos

Sete estratégias automáticas de comparação foram adotadas, e estão descritas na contextualização deste trabalho. São estas: a comparação pela distância de Jaccard, a edição de texto, distância de árvore, TFIDF-NNC, TFIDF-ANC, TFIDF-BNC, TFIDF-LNC.

3.2.3 Coleta de Dados

Comparação de Programas pelos Professores

Onze professores foram convidados para comparar triplas de códigos C_1, C_2 e C_R escolhidas aleatoriamente de uma base de 36 soluções corretas para um mesmo problema. Eles deveriam identificar qual das soluções, C_1 ou C_2 era mais próxima de C_R (solução de referência) ou se ambos os códigos C_1 e C_2 eram igualmente semelhantes ou diferentes de C_R.

Para capturar a comparação de códigos, o experimento foi modelado de forma que cada professor precisasse decidir dentre dois códigos apresentados, C_1 e C_2, qual apresentava uma estratégia mais semelhante a um terceiro código, chamado de código de referência R. Para diminuir o esforço a ser realizado pelo professor, definiu-se que o mesmo código de referência seria utilizado a cada 5 comparações. Assim, ao longo do experimento, 8 códigos distintos foram utilizados e para cada código de referência, 5 pares (C_1, C_2) foram aleatoriamente escolhidos gerando um total de 40 escolhas a serem realizadas pelos professores.

Depois da escolha da base de dados, e para a realização concreta do experimento, cada professor foi então apresentado a uma aplicação online (acessível em: `http://codecompareweb.appspot.com/`) onde três códigos eram apresentados juntamente com o enunciado da questão e a instrução para a escolha da estratégia mais similar. A Figura 3.2 apresenta a tela principal desta aplicação. Nela, o código de referência R é colocado no meio da tela juntamente com os dois códigos C_1 e C_2 a serem avaliados. Abaixo de cada código C_1, C_2 e R há um botão representando a escolha do professor, respectivamente, pelo código C_1 como o que apresenta a estratégia mais parecida com R, pela opção dos códigos C_1 e C_2 serem igualmente semelhantes/diferentes de R e pelo código C_2 apresentar a estratégia mais próxima de R. Esta escolha poderia ser feita também

por teclas de atalho, para facilitar a votação e cada professor poderia escolher o tempo e local mais agradável para a realização desta tarefa e são livres para definir como realizar esta escolha, seja por determinado uso da função fibonacci, pelo tamanho, pela presença de palavras-chaves ou qualquer outra escolha que ele queira. Os dados obtidos a partir das votações dos professores encontram-se no Anexo D.

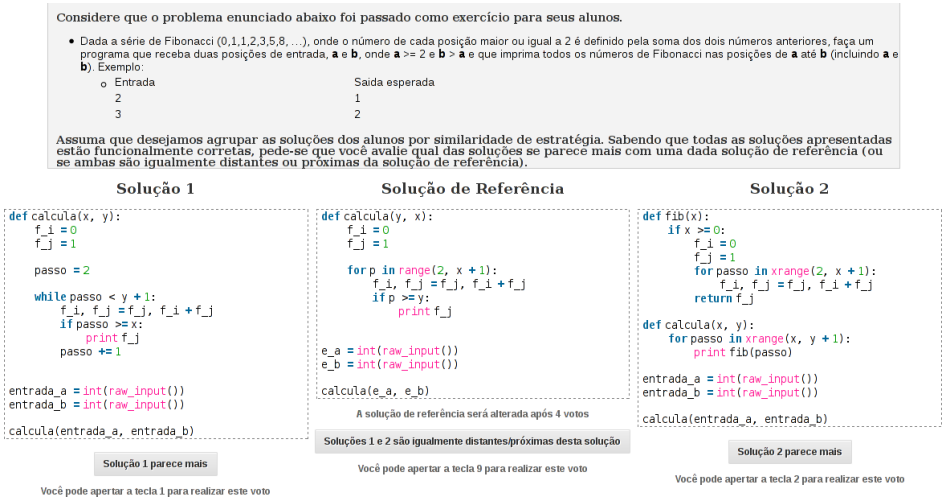


Figura 3.2: Tela da aplicação web utilizada para que o professor comparasse três códigos.

Comparação de Programas pelos Algoritmos

Também era importante definir qual seria o resultado das comparações considerando as estratégias automáticas de comparação de código. Para poder ser determinada a escolha que seria feita pelos algoritmos automáticos, primeiramente definiu-se uma função SIMILAR que, dada uma tripla de códigos (C_1, R, C_2), deveria produzir como saída: 1) C_1 , caso C_1 seja o código com estratégia mais semelhante a R; 2) C_2 caso este seja o código mais próximo a estratégia de R, e; 3) R caso ambos C_1 e C_2 sejam estratégias igualmente semelhantes/diferentes de R.

Essa função $SIMILAR$ pode ser obtida a partir do uso da função $Score$ dos algoritmos. Se $Score(C_1, R) - Score(C_2, R) > 0$, espera-se que C_1 seja um código mais próximo de R do que C_2. No entanto, fazer com que $Score(C_1, R) - Score(C_2, R) = 0$ é estrito na medida que os algoritmos de similaridade são extremamente estritos no cálculo de sua função $Score$.

Para permitir que a função $SIMILAR$ aceite pequenas diferenças que não representam uma mudança de estratégia, incorpora-se um Δ que representa pequenas diferenças de $Score$

que não geram estratégias significativamente distintas. Assim, define-se o valor da função $SIMILAR$ como:

$$\begin{cases} C_1 & \text{if } \text{Score}(C_1, R) - \text{Score}(C_2, R) > \Delta \\ C_2 & \text{if } \text{Score}(C_1, R) - \text{Score}(C_2, R) < -\Delta \\ R & \text{if } |\,\text{Score}(C_1, R) - \text{Score}(C_2, R)| \leq \Delta \end{cases}$$

3.2.4 Processamento dos Dados

Para avaliar a escolha feita entre professores, definiu-se a métrica p_0 de "concordância entre professores". Esta métrica representa o total de escolhas em acordo entre dois professores sobre o total de escolhas realizadas. O intervalo de confiança de cada valor é dado por $p_o \pm 2,021\sqrt{p_o\frac{1-p_o}{40}}$ com confiança de 95%.

Da mesma forma que é feita a concordância entre professores, é possível avaliar a concordância entre estratégias automáticas e professores. No entanto, não há um único valor de Δ que seja representativo para todos os professores e que possa ser escolhido previamente. Se um Δ pequeno for escolhido, a estratégia automática irá diminuir as escolhas por R. No entanto, um Δ grande faz com que $SIMILAR(C_1, R, C_2)$ tenda a R para quase todo C_1, C_2 e R. Assim, para cada análise de concordância entre o modo de comparação dos professores e o modo de comparação de uma estratégia automática, o Δ foi variado até encontrar o valor que gerasse a melhor concordância entre as escolhas das estratégias automáticas e os professores.

3.3 Resultados Obtidos

3.3.1 Q1. Há concordância entre os professores na avaliação de similaridade questões?

É possível observar que, apesar de existir uma concordância de 95% entre dois professores, existe também uma baixa concordância entre-pares, chegando a 62% o menor valor obtido. Estes dados estão exibidos na Tabela 3.4.

Estes dados apontam que há divergências entre alguns dos professores. Avaliando a natureza dessa discordância, ao observar as escolhas discordantes, descobriu-se que isto

Tabela 3.4: Concordância entre professores.

	A	B	C	D	E	F	G	H	I	J	K
A	-	0.85	0.85	0.85	0.82	0.80	0.70	0.72	0.95	0.90	0.80
B	0.85	-	0.90	0.78	0.90	0.80	0.68	0.88	0.85	0.88	0.72
C	0.85	0.90	-	0.82	0.90	0.75	0.70	0.88	0.80	0.82	0.75
D	0.85	0.78	0.82	-	0.78	0.78	0.70	0.80	0.80	0.78	0.75
E	0.82	0.90	0.90	0.78	-	0.80	0.72	0.88	0.82	0.85	0.75
F	0.80	0.80	0.75	0.78	0.80	-	0.68	0.75	0.85	0.85	0.72
G	0.70	0.68	0.70	0.70	0.72	0.68	-	0.62	0.65	0.68	0.62
H	0.72	0.88	0.88	0.80	0.88	0.75	0.62	-	0.72	0.75	0.68
I	0.95	0.85	0.82	0.80	0.82	0.85	0.65	0.72	-	0.95	0.80
J	0.90	0.88	0.85	0.78	0.85	0.85	0.68	0.75	0.95	-	0.80
K	0.80	0.72	0.75	0.75	0.75	0.72	0.62	0.68	0.80	0.80	-

é resultado principalmente de como os professores caracterizam códigos com estratégias muito semelhantes ou não. Na Figura 3.3 é possível visualizar a quantidade de vezes em que cada professor escolheu ambos os códigos como igualmente similares/distintos da solução de referência. Tanto os professores marcados como A, I, J, K e H escolheram esta opção mais de 18 vezes. Por outro lado, os professores B, C, D, E, G e H fizeram essa escolha menos do que 14 vezes.

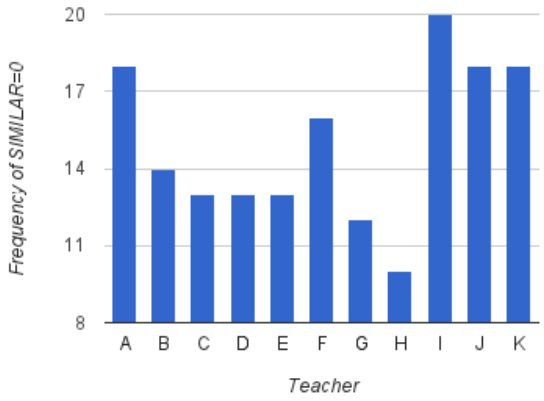


Figura 3.3: Número de vezes em que os códigos C_2 C_1 foram escolhidos como igualmente similares/diferentes de R para cada professor.

No entanto, não é apenas essa decisão de comparar os dois códigos como igualmente distintos/semelhantes que impacta na escolha do professor. O especialista G, apesar de ter feito essa escolha 12 vezes, ainda assim não atingiu uma concordância elevada com outro professor que votou 14 vezes nessa opção. Já o especialista K, mesmo depois de ter feito essa escolha 18 vezes, não conseguiu alcançar uma alta concordância com os professores A e J, que fizeram a mesma escolha também 18 vezes.

3.3.2 Q2. Há concordância entre os professores e estratégias automáticas na avaliação de similaridade?

Os resultados desta comparação e dos valores de Δ utilizados aparecem na Tabela 3.5. É possível ver que o nível de concordância obtido com o melhor das estratégias para cada professor é superior a 0,77. Ainda, para alguns pares de estratégia e professores, um valor maior de Δ é capaz de capturar uma melhor concordância, como no caso de A, F, I, J com as estratégias de edição de texto e árvore, enquanto outros professores atingiram uma melhor concordância, para essas estratégias, com um menor valor de Δ. Ainda, três algoritmos apresentaram um melhor resultado geral: Jaccard, distância de edição e distância de árvore. Enquanto não foi possível capturar a concordância num valor muito elevado para os especialistas G e H, alguns algoritmos para os demais especialistas tiveram a concordância

de até 0,97.

Tabela 3.5: Concordância e valores de Δ entre estratégias automáticas e professores.

			Professores										
			A	B	C	D	E	F	G	H	I	J	K
Estratégias	jaccard	p_0	0,97	0,92	0,92	0,87	0,92	0,87	0,75	0,85	0,97	0,92	0,80
		Δ	0,08	0,04	0,04	0,03	0,05	0,17	0,05	0,02	0,17	0,17	0,04
	árvore	p_0	0,95	0,95	0,90	0,82	0,95	0,82	0,75	0,87	0,95	0,90	0,75
		Δ	0,20	0,06	0,04	0,04	0,06	0,12	0,04	0,07	0,20	0,20	0,04
	texto	p_0	0,95	0,92	0,90	0,82	0,92	0,82	0,75	0,87	0,95	0,90	0,75
		Δ	0,20	0,06	0,07	0,07	0,06	0,11	0,15	0,04	0,20	0,20	0,04
	NNC	p_0	0,87	0,87	0,90	0,82	0,87	0,82	0,77	0,80	0,87	0,85	0,72
		Δ	0,01	0,01	0,01	0,01	0,01	0,01	0,02	0,01	0,04	0,16	0,01
	ANC	p_0	0,92	0,85	0,85	0,85	0,87	0,85	0,77	0,80	0,90	0,85	0,80
		Δ	0,08	0,06	0,08	0,07	0,08	0,08	0,07	0,01	0,08	0,08	0,10
	BNC	p_0	0,92	0,85	0,85	0,85	0,85	0,85	0,77	0,80	0,90	0,85	0,82
		Δ	0,06	0,06	0,06	0,06	0,06	0,09	0,01	0,06	0,09	0,09	0,09
	LNC	p_0	0,92	0,87	0,85	0,82	0,85	0,82	0,77	0,80	0,92	0,87	0,80
		Δ	0,04	0,04	0,10	0,05	0,04	0,08	0,04	0,05	0,09	0,10	0,08

Com o experimento foi possível ver que nem sempre os professores concordaram na comparação de códigos (62% a 95%). Ao mesmo tempo, as três primeiras estratégias automáticas conseguiram obter um valor de concordância sempre acima de 75% com todos os professores.

3.4 Discussão

Cada professor de programação pode ter uma estratégia única de comparação de códigos. Enquanto a estratégia de similaridade baseada na distância de Jaccard apresenta bons resultados para todos os professores, alguns podem concordar mais com outras estratégias automáticas. No entanto, mesmo que dois professores possam concordar com uma mesma estratégia automática, os mesmos podem divergir no valor de Δ que é adequado para

concordar com determinada estratégia. Ou seja, professores podem divergir no quão aceitável é colocar dois códigos como igualmente próximos ou igualmente distantes.

Nenhuma estratégia de TFIDF no entanto é considerada superior para nenhum dos professores avaliados. A variação TFIDF-BNC conta apenas a presença de um determinado termo para colocar dois códigos similares. No entanto, em programação, são poucas as diferentes expressões sintáticas que pode ser usadas de forma única no código. Já a TFIDF-ANC e TFIDF-LNC precisam que um determinado termo apareça com muita frequencia quando comparado aos demais para poder ser destacado no vetor de frequência de termos. Por fim TFIDF-NNC por mais semelhante que seja da estratégia da distância de Jaccard, difere no cálculo da similaridade por ser feito baseada na medida do cosseno, onde a similaridade é reduzida a um meio termo dos dois vetores.

No entanto, as estratégias da TFIDF conseguem uma concordância com um baixo valor de Δ. Isto indica que tais estratégias apresentam uma melhor concordância com os professores nas votações em que um dos códigos C_1 ou C_2 é mais próximo do que o código de referência, ou seja, que eles não eram igualmente próximos de R.

Há espaço para investigar novas estratégias ou de combinar as estratégias existentes. Por exemplo, pode-se utilizar uma abordagem mista que se baseie no consenso de múltiplas abordagens para definir a escolha de qual código é mais semelhante. É possível também utilizar medidas de métricas extraídas do código como critério para seleção de similaridade. Independente da nova estratégia proposta, o design experimental aqui apresentado permite comparar facilmente qualquer nova proposta de estratégia a ser avaliada.

3.5 Ameaças à Validade

As duas principais ameaças à validade deste trabalho que foram identificadas são as seguintes:

A pequena amostra de professores avaliados. Esta ameaça é atenuada na medida que não se busca validar a hipótese para todos os professores de programação. Os resultados indicam que algumas estratégias podem ser mais adequadas para alguns professores. E este resultado é ponto de partida para a discussão apresentada.

O código sintético pode não ser uma amostra típica para avaliação de similaridade.

Enquanto os aspectos criados pelo código sintético podem não capturar todos os aspectos explorados em questões reais ou focar em aspectos que poderiam ser considerados irrelevantes, ele permite avaliar variações comuns de código e serve de ponto de partida para um experimento mais amplo.

3.6 Trabalhos Relacionados

Todos os trabalhos de comparação de código são referências válidas para este trabalho. Ao contrário dos trabalhos existentes que fazem uso de comparação de código, este trabalho buscou avaliar a comparação de código para todos os códigos de uma população, sendo este o maior diferencial em relação ao material existente sobre comparação de código.

Depois da publicação do artigo no SIGCSE, três artigos foram publicados que referenciam este trabalho: dois de Glassman et. al [Glassman et al. 2014; Glassman et al. 2015] e o trabalho de Yin et. al [Yin et al. 2015]. Ambos os trabalhos buscam agrupar soluções próximas, mas não trabalham em si com a medida de similaridade entre questões.

Um trabalho próximo ao que foi realizado neste capítulo é o de Maciel et. al [Maciel et al. 2013] que buscou fazer avaliação do impacto da normalização de códigos na similaridade em técnicas de similaridade de códigos. Entretanto, tal avaliação não comparou a forma com que tais técnicas comparavam em relação a como os professores fazem tal comparação.

3.7 Sumário

Este capítulo buscou avaliar a forma como os professores comparam códigos e como estratégias automáticas podem capturar tal relação. Inicialmente foi feito um estudo considerando uma base de dados sintética onde professores decidiam, dada uma tripla de códigos, contendo dois códigos e um código de referência, qual dos dois códigos era mais próximo do código de referência. Este cenário explorava 3 estratégias completamente diferentes de soluções e 12 variações muito próximas de códigos para cada uma das soluções.

Neste cenário sintético, a primeira descoberta foi a de que professores podem não apresentar um alto índice de concordância (chegando a 62% o menor valor entre si) e de

que as estratégias automáticas, quando considerandas as diferentes variações de Δ, ou seja, do fator que pode considerar dois códigos muito similares ou não, são capazes de capturar o que o professor considera como similar em um nível de pelo menos 75% de concordância no pior caso.

Capítulo 4

Visualização e Uso da Comparação de Programas na Análise de Soluções

No capítulo anterior, foram analisadas diferentes técnicas de similaridade de código. Entre as técnicas avaliadas, a simples comparação entre os tokens sintáticos da linguagem apresentou bons resultados como algoritmo geral, especialmente quando se adapta o limite de tolerância de similaridade considerando cada professor. Neste capítulo, é feita a criação e uso de uma estratégia de visualização para a comparação dos códigos dos alunos e é ilustrado o uso de uma ferramenta para dar suporte a essa visualização. Com esta informação gerada, os alunos avaliam então os códigos de seus pares.

4.1 Visualização da Comparação de Programas

Considerando que as estratégias automáticas foram capazes de comparar códigos de forma semelhante a um professor, construiu-se uma visualização das comparações geradas automaticamente. Para este fim, utilizou-se a estratégia de comparação de programas baseando-se na distância de Jaccard por ser rápida e ter tido uma boa concordância com a maioria dos professores. Isto também foi feito para simplificar a análise experimental desse estudo.

Esta visualização é gerada a partir da estratégia descrita no Capítulo 2, da construção de uma árvore filogenética proposto por Saitou e Nei [Saitou and Nei 1987]. Mihaescu et al. [Mihaescu et al. 2006] demonstra que este algoritmo é uma estratégia rápida

e com tendência a produzir representações fidedígnas as comparações representadas na visualização.

Uma ferramenta foi construída para a exibição dessa visualização e foi usada em sala de aula para averiguar seu potencial de utilização. O uso da ferramenta permitiu uma visualização rápida das seguintes soluções: 1) o maior conjunto de soluções semelhantes, isto é, as soluções próximas mais utilizadas pelos alunos; 2) soluções corretas, mas distantes das soluções gerais do aluno; 3) soluções próximas de uma solução correta mas que não eram bem sucedidas num caso de teste. Atualmente é possível acessar a ferramenta no endereço: http://relatedcode.appspot.com.

Para expandir a visualização de forma tornar possível o seu uso por alunos para a avaliação de códigos, construiu-se uma nova ferramenta com mais recursos. É possível ver uma tela desta nova ferramenta na Figura 4.1. Há um espaço para colocar comentários sobre códigos a serem avaliados e há também o grafo com as relações entre os códigos. O aluno pode clicar num elemento do grafo para tornar sua exibição permanente no canto inferior esquerdo ou passar por cima de um nó para exibir o código que este nó representa. Os nós são coloridos neste grafo indicando sucesso (verde) ou falha (vermelho) da solução. Por fim, ao considerar que sua avaliação está concluída, o aluno clica em submeter.

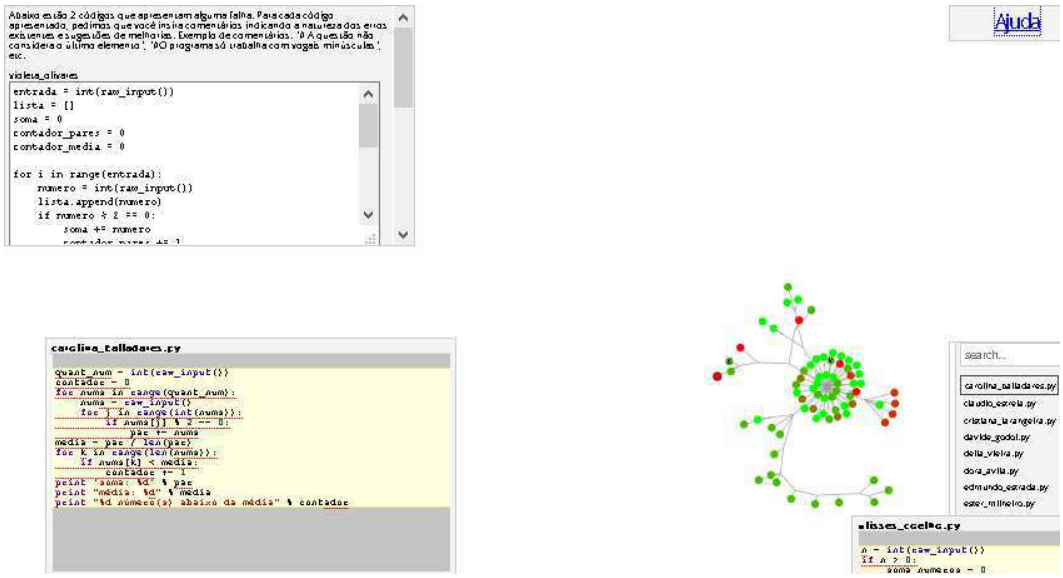


Figura 4.1: Ferramenta de comparação de códigos com a visualização proposta

Considerando a visualização e o ferramental disponível, construiu-se um experimento para avaliar seu uso num processo de avaliação entre-pares dos alunos.

4.2 Design do Experimento

Neste experimento, faz-se uso do feedback dado pela visualização proposta anteriormente para auxiliar a avaliação que os alunos fazem dos códigos um dos outros com o uso da informação oferecida.

4.2.1 Questões de Pesquisa

Quatro questões de pesquisa foram identificadas para este experimento. Estas questões buscam identificar se a visualização das comparações de códigos é útil ao melhorar a avaliação feita pelo aluno (Q1), se permite oferecer informação sobre uma questão avaliada, de forma que tal conhecimento adquirido possa ser imediatamente aplicado (Q2), se a visualização estimula os alunos a analisarem mais códigos, de forma serem expostos a diferentes estratégias (Q3) e se os alunos gostam da ferramenta proposta de avaliação (Q4).

Q1 A avaliação de código com a informação das relações entre códigos ajuda os alunos a fazer uma melhor avaliação dos códigos dos colegas?

Q2 Ao mostrar uma visualização da comparação de códigos durante a avaliação de códigos, os alunos melhoram sua capacidade imediata de produzir uma solução correta para uma questão semelhante à avaliada?

Q3 A visualização da comparação de códigos estimula os alunos a olharem mais códigos?

Q4 Os alunos gostam da ferramenta de avaliação?

4.2.2 Metodologia

O experimento foi realizado nas turmas de introdução à programação do semestre de 2014.1 do curso de bacharelado em Ciência da Computação da Universidade Federal de Campina Grande. Existem dois grupos de alunos com tratamentos diferentes, o primeiro grupo, de controle, foi exposto a códigos de alunos mas não tem informação da relação entre códigos através do grafo produzido pelos algoritmos de comparação de códigos. O segundo grupo, o experimental, recebeu acesso aos mesmos códigos, mas com a informação do grafo de relação entre estes códigos. O experimento foi realizado durante um horário normal de aula

em que os alunos foram instruídos a participar de uma atividade de codificação e de avaliação do código dos colegas.

Durante a realização do experimento, os alunos tiveram 25 minutos para o desenvolvimento de uma questão e mais 25 minutos para avaliação de outro código da mesma questão. Depois disso, os mesmos tempos são repetidos para uma nova questão e, por fim, o aluno tem 20 minutos para preenchimento de um formulário com perguntas sobre o experimento. Qualquer aluno que acabar uma etapa antes do limite pode passar para a próxima etapa imediatamente. Ao atingir o limite, os alunos são instruídos a enviarem o que tiverem feito no momento.

No experimento, ambos os grupos não avaliaram ou tiveram acesso a códigos dos colegas do período, mas sim de soluções produzidas em por alunos de períodos anteriores, além da própria solução que o aluno submete. Todas as identificações dos códigos de períodos anteriores foram substituídas por nomes fictícios. A atividade de avaliação foi caracterizada por duas etapas bem definidas:

- Um enunciado é apresentado com um espaço para que o aluno coloque seu código e outro espaço para colocar a entrada a ser usada na execução do código, como mostra a Figura 4.2. Ao executar o código, ainda na interface web, o resultado da execução, seja uma saída da execução ou saída de erro, é exibido em um terceiro espaço de código. Quando o aluno achar que a questão satisfaz o enunciado, ele pode salvá-la.

Figura 4.2: Primeira etapa da ferramenta de comparação de códigos

- Em seguida, é exibida uma tela com os códigos dos alunos. Existem dois grupos distintos de alunos: o de controle, que não recebe a informação sobre a relação entre códigos, e o grupo experimental, que tem esta informação. Para o aluno do grupo de controle é exibida a tela que está mostrada na Figura 4.3. Nela existem dois espaços (canto superior esquerdo) para que o aluno coloque a avaliação de dois códigos a serem definidos no experimento. A tela exibe duas listagens de códigos que podem ser selecionados para exibir a diferença entre eles. As cores dos códigos na listagem indicam sucesso (verde) ou falha (vermelho) da solução para a

questão. Os alunos do grupo experimental entram numa tela diferente, já descrita anteriormente na Figura 4.1. Os alunos são instruídos a colocar comentários sobre o código em avaliação, indicando erros e más-práticas, mas estão livres para fazer comentários positivos ou sugestões gerais. Todas as ações dos alunos são monitoradas e armazenadas para futura análise.

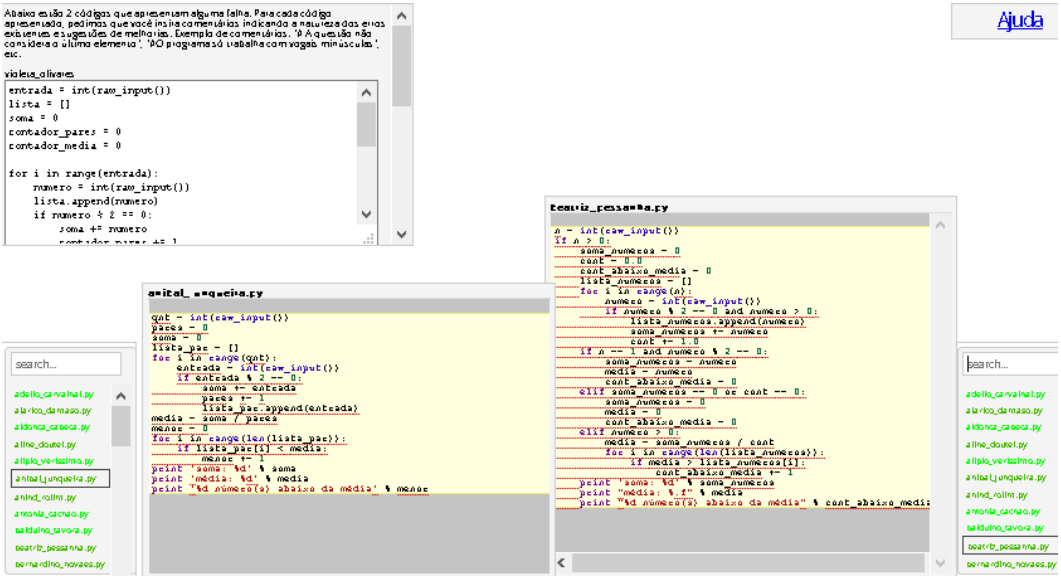


Figura 4.3: Segunda etapa da ferramenta de comparação de códigos para o grupo de controle

Estas duas etapas foram novamente repetidas para uma nova questão. Por fim, os alunos foram direcionados a um formulário para avaliação geral da experiência que passaram. Este formulário será utilizado para responder as questões de pesquisa Q1, sobre melhoria do processo de avaliação pelos alunos, e Q4, sobre aceitação da ferramenta.

O formulário utiliza da escala Likert de concordância (de 1, discordo fortemente, a 5, concordo fortemente) e começa com quatro afirmações sobre o aluno, para que possa se auto-avaliar. Estas afirmações servem com propósito de análise da amostra, identificando se os dois grupos avaliados são duas amostras comparáveis ao apresentarem características semelhantes de meta-conhecimento. As afirmações são as seguintes:

- Eu me considero um aluno estudioso.
- Aprender programação é fácil.
- Eu domino o conteúdo de programação I.

- Eu gosto de programar.

Em seguida, aparecem quatro afirmações sobre o processo de avaliação de códigos entre pares. Estas afirmações podem ser utilizadas para comparar diretamente o grupo de controle e o grupo experimental. Na medida que os dois grupos são equivalentes em termos de amostra, a única variação entre as respostas de cada grupo podem ser justificadas pela presença da informação da comparação de códigos. São afirmações sobre a visão do aluno da avaliação de códigos:

- Revisar outros códigos é importante para aprender programação.
- Revisar outros códigos me faz refletir sobre possíveis erros de um programa.
- Ter acesso a diferentes códigos me ajuda a avaliar um código em específico.
- Eu me considero apto para revisar os códigos dos meus colegas.

Posteriormente, são apresentadas cinco afirmações sobre a ferramenta e a estratégia de uso da mesma, procurando identificar a estratégia adotada pelo aluno para a avaliação de código e a satisfação do mesmo com a ferramenta. São estas afirmações:

- A ferramenta de avaliação de código é simples de usar.
- Foi fácil encontrar a informação que eu precisava para fazer a atividade de revisão de código.
- Eu procurei observar principalmente códigos muito semelhantes aos códigos que eu tinha que avaliar.
- Recomendo o uso da ferramenta de avaliação de código ao longo da disciplina de programação.
- No geral, eu estou satisfeito com a ferramenta.

Os alunos foram instruídos a não conversar com os colegas e a tirarem dúvidas referentes à ferramenta com os professores. Apenas o endereço da ferramenta de comparação de códigos era liberado para acesso pelos alunos. Fora isso, nenhuma outra restrição foi imposta, ficando os alunos livres a executarem o código e usarem qualquer outra ferramenta disponível na máquina.

4.2.3 Dados

Alunos

Inicialmente, 35 alunos do período 2014.1 da Universidade Federal de Campina Grande participaram do experimento. Estes estudantes estavam matriculados na disciplina durante a realização do experimento. Todos os alunos tinham avançado mais de 20% das unidades do curso. Estes eram escolhidos para o grupo de controle e grupo de experimento de forma aleatória, baseando-se no login dos mesmos. Todos os alunos do experimento haviam passado pelas 4 unidades iniciais desse curso, isto é, tinham no mínimo conhecimento sobre manipulações de listas.

Questões

Duas questões foram escolhidas para o experimento. Decidiu-se escolher uma questão ainda não disponibilizada para os alunos do período 2014.1 mas que havia sido utilizada em semestre anteriores. Ainda, foram escolhidos dois exercícios que exploram conteúdos parecidos, de forma a avaliar se a revisão com a informação adicional de similaridade de código apresenta efeito imediato na forma como os alunos resolveriam a segunda questão.

A primeira questão tem como nome “Média dos pares” e tem o enunciado descrito na Tabela 4.1.

Tabela 4.1: Questão Média dos Pares.

<table>
<tr><td colspan="3">Escreva um programa que leia uma série de números inteiros não negativos e imprima a soma dos números pares lidos. O programa deve imprimir ainda a quantidade de números que estão abaixo da média dos números pares lidos. Assuma que existirá pelo menos um número par.</td></tr>
<tr><td colspan="3">Entrada A primeira linha da entrada é um número inteiro N, N > 0, indicando a quantidade de números a serem lidos. As N linhas seguintes são os números da série a serem lidos.</td></tr>
<tr><td colspan="3">Saída Na saída, seu programa deve imprimir a soma dos números pares, a média inteira dos números pares e a quantidade de números da série que são menores do que o valor da média. Veja o formato da saída nos exemplos de entrada e saída apresentados.</td></tr>
<tr><td>Entrada
4
6
7
15
8</td><td>Saida esperada
soma: 14
média: 7
1 número(s) abaixo da média</td><td></td></tr>
</table>

A segunda questão tem como nome “média dos extremos” e está descrita na Tabela 4.2.

Tabela 4.2: Questão Média dos Extremos.

<table>
<tr><td colspan="2">Escreva um programa que utiliza a média dos extremos para classificar um conjunto de números inteiros. Considere que o conjunto sempre possui pelo menos dois números inteiros. A média dos extremos é definida como a média aritmética considerando o maior e o menor elemento desse conjunto de números. Por exemplo, se o menor inteiro for 3 e o maior inteiro for 7, a média dos extremos é 5.0.</td></tr>
<tr><td colspan="2">Entrada Na primeira linha da entrada, é lida a quantidade N de números do conjunto, N > 1. As N linhas seguintes correspondem aos N números inteiros do conjunto.</td></tr>
<tr><td colspan="2">Saída Na saída, seu programa deve imprimir o menor valor, o maior valor, a média dos extremos e os totais de números abaixo e acima da média dos extremos. Veja o formato da saída nos exemplos de entrada e saída apresentados.</td></tr>
<tr><td>Entrada
8
3
2
7
1
0
5
2
4</td><td>Saida esperada
Menor número: 0
Maior número: 7
Média dos extremos: 3.50
5 número(s) abaixo da média
3 número(s) acima da média</td></tr>
</table>

Para evitar ambiguidade e problemas com a comparação de valores, foi definido que nenhum dos valores utilizados como entrada da questão seria igual à média calculada. Os alunos pediram para elucidar dois aspectos durante o experimento: o tipo de entrada e o tipo de saída. Todos os alunos foram informados que a entrada era sempre de números inteiros e que a saída era livre para o primeiro enunciado, e exigia duas casas decimais de formatação

para o segundo enunciado.

Ainda, que toda a entrada é um número inteiro e que a saída sempre tem uma casa decimal (arredondamento simples por truncamento). Isto foi dito durante o experimento, apesar de nenhum teste considerar esses cenários.

Códigos para Revisão

Dois códigos que haviam sido submetidos por alunos de períodos anteriores foram selecionados para serem avaliados em cada questão pelos alunos. Estes códigos eram apresentados e avaliados pelos alunos apenas depois que o aluno codificasse uma solução para a questão especificada. Para a questão *"média dos pares"*, o primeiro código a ser avaliado pelos alunos, o Código 1, apresenta um problema ao considerar sempre 7 como média para fazer a contagem dos valores abaixo da média e por fazer divisão inteira no cálculo da média.

```
entrada = int(raw_input())
lista = []
soma = 0
contador_pares = 0
contador_media = 0
for i in range(entrada):
        numero = int(raw_input())
        lista.append(numero)
        if numero % 2 == 0:
                soma += numero
        contador_pares += 1
media = soma / contador_pares
for i in range(len(lista)):
        if lista[i] < 7:
                contador_media += 1
print "soma: %d" % soma
print "média: %d" % media
print "%d número(s) abaixo da média" % contador_media
```

Código Fonte 1: Primeiro código da questão *"média dos pares"*

Já o segundo código a ser avaliado pelos alunos, o Código 2, apresenta um problema na forma como é feito o cálculo da média, pois só considera os valores que são pares e também faz uso de uma divisão inteira.

```
quantidade = int(raw_input())
numeros = []
soma = 0
for n in range(quantidade):
        n = int(raw_input())
        if n % 2 == 0:
                numeros.append(n)
        soma += n
media = soma / (len(numeros))
abaixomed = 0
for i in range(len(numeros)):
        if numeros[i] < media:
                abaixomed += 1
print "soma: %d" % soma
print "média: %d" % media
print "%d número(s) abaixo da média" % abaixomed
```

Código Fonte 2: Segundo código da questão *"média dos pares"*

Para a questão "*a média dos extremos*", novamente dois códigos foram selecionados entre os códigos submetidos por alunos de períodos anteriores para serem usados no experimento. O primeiro código a ser avaliado pelos alunos, o Código 3 apresenta um problema ao fixar o limite inferior e superior em código, e não calculá-los de acordo com a entrada.

```
total = int(raw_input())
numeros = []
maior = 0
menor = 7
for i in range(total):
    num = int(raw_input())
    numeros.append(num)
    if num > maior:
            maior = num
    if num < menor:
            menor = num
media = (maior + menor) / 2.0
maior_m = 0
menor_m = 0
for i in range(len(numeros)):
    if numeros[i] > media:
            maior_m += 1
    if numeros[i] < media:
            menor_m += 1
print "Menor número: %i" % (menor)
print "Maior número: %i" % (maior)
print "Média dos extremos: %.2f" % (media)
print "%i número(s) abaixo da média" % (menor_m)
print "%i número(s) acima da média" % (maior_m)<
```

Código Fonte 3: Primeiro código da questão "*a média dos extremos*"

Já o segundo código avaliado pelos alunos, o Código 4, faz a divisão do maior e menor valor de forma inteira, o que retorna uma média errada para alguns casos.

```
qnt_num = int(raw_input())
lista = []
for n in range(qnt_num):
    numero = float(raw_input())
    lista.append(numero)
maior = 1
menor = 1
for num in lista:
    if num > maior:
            maior = num
    if num < menor:
            menor = num
media = (maior + menor)/2
acima = 0
abaixo = 0
for i in lista:
    if i > media:
            acima += 1
    if i < media:
            abaixo += 1
print """Menor número: %d
Maior número: %d
Média dos extremos: %.2f
%d número(s) abaixo da média
%d número(s) acima da média""" % (menor, maior, media, abaixo, acima)
```

Código Fonte 4: Segundo código da questão "*a média dos extremos*"

4.2.4 Coleta de Dados

Dos 35 alunos participantes, estes eram divididos aleatoriamente entre os dois grupos pela ferramenta. Destes, 21 foram alocados para o grupo experimental e 14 fizeram parte do

grupo de controle. Dos 14 alunos selecionados para o grupo de controle, nenhum concordou fortemente que se considerava um aluno estudioso. No grupo experimental, cinco alunos concordaram fortemente com essa afirmação.

Para tornar os dois grupos equivalentes em quantidade e nas métricas de auto avaliação, 7 alunos foram removidos do grupo experimental, sendo 5 os que concordaram fortemente com a afirmação de que eram alunos estudiosos e dois, aleatoriamente selecionados, que apenas concordaram com tal afirmação. Esta solução não altera os resultados obtidos na análise do experimento, mas uniformiza as duas amostras. Uma visão dos dados obtidos por aluno na amostra final encontra-se no Anexo E deste trabalho.

Assim, a população resultante apresentou valores semelhantes de média (considerando o intervalo de confiança, IC, apresentado) na escala Likert nos 4 aspectos que definem o auto conhecimento e participação na disciplina, como mostra a Tabela 4.3, apesar de apresentar uma pequena diferença no comportamento da mediana.

Tabela 4.3: Afirmações relacionadas a própria percepção do aluno como estudante de programação.

Afirmação	Controle			Experimental		
	Mediana	Δ	IC	Mediana	Δ	IC
1. Eu me considero um aluno estudioso.	3	$3,4$	$0,3$	$3,5$	$3,4$	$0,4$
2. Aprender programação é fácil.	3	$3,2$	$0,6$	3	$3,4$	$0,5$
3. Eu domino o conteúdo de programação I.	3	$3,3$	$0,4$	4	$3,6$	$0,3$
4. Eu gosto de programar.	5	$4,5$	$0,3$	5	$4,8$	$0,3$

Estes valores, apesar de colhidos pós-teste, não sofrem influência do experimento e, assim, são seletores amostrais válidos dada a população apresentada. Os dados coletados de códigos produzidos pelos alunos e o resultado das revisões de códigos, foram obtidos diretamente através da ferramenta e do formulário para imediato processamento e avaliação. O sistema também provê informações sobre eventos de interação com a ferramenta, como

abrir um código, fechar um código, selecionar outro código.

4.2.5 Processamento dos Dados

Para cada código submetido pelos alunos, foi calculada sua taxa de acerto de acordo com uma bateria de testes automáticos. Para cada código, foi gerado um valor entre 0 e 1 representando a porcentagem de testes em que os programas passaram. Ao avaliar se os alunos apresentam programas com maior taxa de acerto na segunda questão, espera-se responder a segunda questão de pesquisa, que avalia sobre a melhoria dos alunos em produzir uma solução correta imediatamente após uma experiência de avaliação com a visualização proposta.

Para cada revisão de códigos, os comentários gerados pelos alunos foram testados e classificados em três categorias distintas: *i*) **A**ceito, para comentários que, se considerados, ajudam a melhorar a corretude e conformidade do código com a especificação; *ii*) **E**rrado, para comentários que, se considerados, fariam com que o código ficasse ainda com menor conformidade com a especificação ou ficasse com conformidade inválida por novos motivos além dos originais; e, por fim *iii*) **N**eutros, que representam os comentários que são de aspectos que vão além de sua corretude de especificação, como o bom nome de variáveis, endentação e uso apropriado de espaços e linhas em branco. É com esta classificação de comentários que se avalia, objetivamente, a capacidade do aluno em produzir melhores comentários durante sua avaliação (Q1).

Por fim, avalia-se quantos códigos o aluno analisa durante o período de avaliação. Para evitar considerar códigos que foram selecionados, mas não avaliados, isto é, que o aluno ativou a visualização porque estava no caminho do mouse dele, foram considerados apenas os códigos que foram observadas por mais de um segundo, tempo suficiente para descartar eventos que foram apenas transições até o aluno chegar a um código de interesse. Também se fez uma diferença entre códigos observadas por 10 segundos ou mais, que indicam especial foco no que foi selecionado. Esta última informação é importante para avaliar a questão de pesquisa Q3, que procura entender se os alunos olham, ou não, mais códigos com a visualização proposta.

4.3 Resultados Obtidos

Q1. A avaliação de código com a informação das relações entre códigos ajuda os alunos a fazer uma melhor avaliação dos códigos dos colegas?

Os alunos do grupo experimental apresentaram, em média, melhor concordância $(3,7)$ com a afirmação sobre a auto-avaliação da sua capacidade de revisar/avaliar o código dos colegas quando comparados aos alunos do grupo de controle $(3,0)$. Essa característica e as demais características sobre o processo de avaliação de códigos entre pares investigadas no formulário, estão apresentadas na Tabela 4.4.

Tabela 4.4: Afirmações relacionadas ao processo de avaliação de código entre pares.

Afirmação	Controle			Experimental		
	Mediana	Δ	IC	Mediana	Δ	IC
5. Revisar outros códigos é importante para aprender programação.	5	$4,6$	$0,3$	5	$4,6$	$0,3$
6. Revisar outros códigos me faz refletir sobre possíveis erros de um programa.	5	$4,6$	$0,3$	5	$4,8$	$0,2$
7. Ter acesso a diferentes códigos me ajuda a avaliar um código em específico.	5	$4,4$	$0,5$	5	$4,4$	$0,5$
8. Eu me considero apto para revisar os códigos dos meus colegas.	3	$3,0$	$0,6$	4	$3,7$	$0,4$

Para avaliar a qualidade dos comentários gerados pelos alunos, tais comentários produzidos por eles foram classificados como **A**certados, **E**rrados ou **N**eutros. Esta classificação objetiva, permite olhar a natureza das sugestões apresentadas nas revisões dos alunos. De acordo com a Tabela 4.5 é possível visualizar os valores de cada questão (1 ou 2), código (novamente 1 ou 2) e categoria (A, E, N). Não é possível observar diferença

estatisticamente significativa entre os dois grupos para a quantidade de comentários nas diferentes classificações. Considerando o total de comentários feitos por cada grupo, o grupo experimental fez 57 comentários, contra 47 do grupo de controle.

Tabela 4.5: Média de comentários de diferentes categorias para diferentes códigos.

Código	**Categoria dos Comentários**	Controle	IC	Experimental	IC
1	Acertadas	0,8	0,2	0,8	0,3
	Erradas	0,3	0,3	0,2	0,2
	Neutras	0,4	0,4	0,5	0,3
2	Acertadas	0,5	0,3	0,4	0,3
	Erradas	0	0	0,1	0,2
	Neutras	0,3	0,2	0,5	0,3
3	Acertadas	1,1	0,5	1,1	0,5
	Erradas	0,1	0,1	0,2	0,3
	Neutras	0	0	0,2	0,2
4	Acertadas	1	0,5	1,1	0,5
	Erradas	0,3	0,3	0,4	0,3
	Neutras	0,1	0,1	0,3	0,3

4.3.1 Q2. Ao mostrar uma visualização da comparação de códigos durante a revisão de códigos, os alunos melhoram sua capacidade imediata de produzir uma solução correta para uma questão semelhante à avaliada?

Para isto, foi avaliada a corretude das questões do aluno considerando o total de aceitações obtidas em testes automáticos sobre o número total de testes automáticos. No geral, ambos os grupos se comportaram de mesma forma, como indicado na Tabela 4.6.

Tabela 4.6: Média de acerto de uma questão (0 a 1).

Questão	Controle	IC	Experimental	IC
1	$0,6$	$0,2$	$0,6$	$0,2$
2	$0,3$	$0,2$	$0,4$	$0,2$

4.3.2 Q3. A visualização da comparação de códigos estimula os alunos a olharem mais códigos?

Em ambas as questões avaliadas, como mostra a Tabela 4.7, há uma dominância de mais códigos observados no grupo experimental. Entretanto, o valor observado ainda é considerado baixo, especialmente para a segunda questão. Na segunda questão, o aluno praticamente não fez uso da análise prolongada de outros códigos, o que pode indicar que a utilização de outros códigos para a detecção do erro não se fez necessária como na primeira questão.

Tabela 4.7: Número médio de códigos observados pelos alunos divididos em duas categorias de tempo gasto durante a observação.

Tratamento	Tempo de leitura	Códigos avaliados - Q1	Códigos avaliados - Q2
Controle	1-10s	$2,8$	$2,0$
	10s+	$1,8$	$1,0$
Experimental	1-10s	$6,5$	$3,2$
	10s+	$2,6$	$1,0$

4.3.3 Q4. Os alunos gostam da ferramenta de avaliação de código?

A última questão de pesquisa foi respondida com a última avaliação gerada do formulário dos alunos. Os resultados estão apresentados na Tabela 4.8. É possível observar que o uso da comparação de código é marcado como de mais simples usabilidade pelo grupo experimental quando compara-se o resultado obtido no grupo de controle. Os demais aspectos não apresentaram diferença relevante entre os grupos explorados.

Tabela 4.8: Afirmações relacionadas à utilização da ferramenta.

Afirmação	Controle			Experimental		
	Mediana	Δ	IC	Mediana	Δ	IC
9. A ferramenta de avaliação de código é simples de usar.	$3,5$	$3,4$	$0,7$	4	$4,3$	$0,4$
10. Foi fácil encontrar a informação que eu precisava para fazer a atividade de revisão de código.	3	$3,7$	$0,6$	4	$3,7$	$0,7$
11. Eu procurei observar principalmente códigos muito semelhantes aos códigos que eu tinha que avaliar.	$3,5$	$3,2$	$0,6$	3	$2,9$	$0,7$
12. Recomendo o uso da ferramenta de avaliação de código ao longo da disciplina de programação.	$4,5$	$3,9$	$0,8$	5	$4,4$	$0,4$
13. No geral, eu estou satisfeito com a ferramenta.	4	$3,9$	$0,6$	$4,5$	$4,3$	$0,5$

4.4 Discussão

A visualização, mesmo que baseada numa estratégia simples, é uma das contribuições deste trabalho. O arcabouço ferramental produzido permite uma maneira rápida e fácil de visualização de códigos e da comparação destes. Da aplicação desse ferramental, quatro resultados merecem destaque.

Primeiramente, os alunos que fizeram uso desta visualização se consideram mais aptos à atividade de avaliação de códigos dos colegas. Com a informação das relações de códigos, o aluno pode identificar quais códigos observar e ter uma idéia geral de quais códigos utilizar para auxiliar sua avaliação.

O segundo aspecto a ser considerado nesta discussão é de que o uso da comparação de código não apresentou diferença em termos de qualidade do processo de avaliação.

O terceiro aspecto a ser considerado é a capacidade dos alunos não terem apresentado um melhor desempenho em uma questão parecida com aquela em que foi feita a avaliação. Isto indica que, para um curto período de tempo, a fixação sobre o conteúdo e corretude da questão avaliada não apresenta um impacto significativo.

Por fim, destaca-se a simplicidade de uso da visualização proposta, aceita por mais da metade da turma com nota máxima. Não adianta sugerir uma visualização ou qualquer outra proposta nova de feedback sem antes existir a aceitação da turma para o seu uso. Neste sentido, há potencial para o uso deste feedback ao longo de um curso de programação.

4.5 Ameaças à Validade

As três principais ameaças à validade deste trabalho que foram identificadas são as seguintes:

O efeito do uso inicial de uma ferramenta Os dois grupos foram expostos pela primeira vez a ambas as ferramentas e boa parte do tempo gasto reflete o tempo necessário para se acomodar à ferramenta em uso. No entanto, esta ameaça é reduzida já que afeta igualmente o grupo de controle e o grupo experimental.

Apenas duas questões foram avaliadas A questão escolhida pode não ser a melhor representante para a atuação de similaridade de códigos ou mesmo para o processo de revisão. Neste aspecto, é necessária a replicação do experimento em diferentes cenários, tentando trabalhar inclusive o efeito duradouro da técnica aplicada.

Foi analisada apenas uma estratégia de comparação de códigos Como descrito no capítulo anterior, o uso de uma determinada estratégia ou uma variação do limite de similaridade é suficiente para que haja maior ou menor aceitação das escolhas de similaridade feitas por estratégias automáticas. Para diminuir o número de fatores do experimento, adotou-se apenas um método, mas melhores resultados poderiam surgir ao selecionar outra estratégia automática a ser utilizada na revisão de código.

4.6 Trabalhos Relacionados

O trabalho de revisão de códigos não é algo novo. O trabalho de Sitthiworachart e Joy [Sitthiworachart and Joy 2004] apresenta uma proposta web de revisão de código que faz uso da informação de testes automáticos, mais a informação gerada pela revisão e qualidade das revisões feitas pelo aluno, para gerar uma nota e prover informação útil. Esta base foi utilizada por Joy et. al [Joy et al. 2005] para o BoSS, um sistema de revisão online que faz uso de técnicas de comparação de código, com o objetivo de detecção de plágio.

O CoMoTo [Meyer et al. 2011], é a ferramenta que mais se aproxima, em termos de visualização de relações de código, com a atual proposta deste trabalho. No CoMoTo, as similaridades dos códigos são exibidas de forma que o professor possa usar tal informação para detecção de plágio, no entanto, esta informação não é utilizada para a revisão de códigos entre os alunos.

Por fim, o trabalho de Chinn [Chinn 2005] é um dos poucos que faz uso de uma avaliação longitudinal sobre a natureza do processo de revisão de código, identificando que há ganho no processo de ensino-aprendizagem dos alunos, inclusive para as atividades que vão além da revisão de código. Neste sentido, o trabalho apresentando aqui é apenas um experimento inicial para este processo de avaliação longitudinal.

4.7 Sumário

Neste capítulo, foi proposta uma visualização de diferentes soluções para uma mesma questão que apresenta a informação da proximidade destas soluções. Em seguida, foi realizado um experimento para avaliar o uso desta visualização no processo de avaliação de códigos entre os alunos. Para tanto, duas questões foram selecionadas de forma a avaliar a capacidade do aluno em avaliar códigos considerando, ou não, a presença da visualização.

De forma objetiva, cada sugestão feita pelos alunos foi classificada como Acertadas, Neutros, Erradas, identificando assim a natureza da revisão. Tanto os alunos do grupo de controle como do experimental apresentaram o mesmo comportamento quanto a este aspecto objetivo de avaliação. Ainda, os dois grupos apresentaram praticamente as mesmas taxas de aceitação da questão produzida, considerando testes automáticos para avaliação das mesmas.

Com base no experimento realizado, a presença da visualização não fez com que os alunos olhassem significativamente mais códigos, mas fez com que os alunos se sentissem mais aptos a avaliar os códigos dos colegas ($3,7$ contra $3,0$ do grupo de controle, considerando uma escala Likert de 5 níveis). Os alunos também consideraram simples fazer uso da ferramenta com a visualização proposta ($4,3$ comparados com $3,4$).

Capítulo 5

Análise de Questões de Listas de Exercícios

O capítulo anterior trabalhou com o uso da similaridade de códigos para oferecer informação útil ao aluno durante o processo de revisão. Neste capítulo, o uso da similaridade é aplicado para identificar como se relacionam as questões de uma lista de exercícios e para gerar uma visualização desta comparação. Não só a similaridade de código é trabalhada, como o uso de técnicas para identificar o que cada questão explora em termos de sintaxe da linguagem.

5.1 Visualização das Questões de uma Lista

Para geração da visualização das questões de uma lista de exercícios, primeiramente busca-se identificar qual programa é mais representativo de uma questão. Para esta identificação, observa-se o conjunto de soluções propostas pelos alunos para então identificar uma solução que represente as características mais comuns presentes nestas soluções. De posse de uma solução de referência para cada questão, propõe-se uma visualização para avaliar a diferença entre diferentes soluções para as questões de uma lista.

5.1.1 Identificação da Solução Relevante

Para identificar a solução relevante de cada questão, faz-se uso novamente do algoritmo de comparação que utiliza a distância de Jaccard. Para isto, cada solução de uma questão passa

por uma etapa de extração de tokens. Nesta etapa, os tokens são extraídos diretamente dos códigos. Alguns destes termos são descartados por serem comuns a todos os documentos. São termos descartados desta análise os tokens extraídos dos códigos que representam atribuição de nome de variável (AssName), atribuição a uma variável (Assign) e uso de uma variável (Name). Comentários são descartados pelo interpretador sintático de Python.

Cada solução dos alunos é convertida em um multiconjunto (conjunto com repetições) contendo os tokens extraídos do código. E, para cada questão, as soluções daquele exercício são comparadas par-a-par utilizando a distância de Jaccard, calculada pela interseção dos multiconjuntos de tokens desses pares, sobre a união desses mesmos dois multiconjuntos. Novamente, isto retorna um valor entre 0 e 1 em que 0 representa soluções completamente distintas e 1 representa soluções idênticas para o algoritmo.

Assim, para uma determinada questão, a solução de referência é aquela mais próxima de todos os demais códigos, utilizando para isso a distância euclidiana. Ao utilizar a distância euclidiana, o programa selecionado como solução de referência é o melhor candidato a ter os tokens comumente utilizados nas demais soluções dos alunos de uma questão.

5.1.2 Visualização da Comparação por Similaridade

Para criar uma visualização da informação de comparação de soluções das diferentes questões, utiliza-se um mapa de calor associando cada par de soluções de referência. Na Figura 5.1 é possível ver um exemplo deste mapa de calor onde cada linha representa uma questão (em ordem) e cada coluna representa também cada exercício (em ordem). Na diagonal é possível ver uma cor mais clara da célula indicando uma similaridade máxima, de valor 1 e, quanto menor o valor de similaridade entre as demais questões, mais escura é a cor da célula. Na figura de exemplo é possível ver 5 questões sendo apresentadas e é possível observar que a quarta questão apresenta uma maior similaridade com a quinta questão. Já o quinto exercício apresenta maior semelhança com as questões 2, 3 e 4.

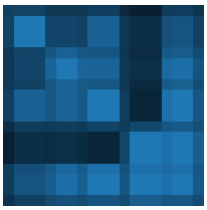


Figura 5.1: Exemplo de Mapa de Calor

5.2 Agrupamento Semântico

Enquanto a visualização de proximidade das questões pode ser útil para identificar quais questões são parecidas e como se relacionam ao longo de uma lista de exercícios, é possível fazer uso de técnicas de agrupamento semântico para extrair informação útil que complemente a visualização existente. Nas técnicas de agrupamento semântico, documentos compostos de termos são processados de forma identificar quais os tópicos explorados nestes documentos. Um tópico é um conceito semântico que pode ser descrito por um conjunto de termos relevantes. Por exemplo, este trabalho de tese é um documento em que cada palavra representa um termo. Este trabalho, e outros da área, poderiam ser identificados pelo tópico descrito pelos termos "programação", "ensino" e "aprendizagem". Intuitivamente, podemos dizer que este tópico versa sobre o ensino de programação.

Uma estratégia de agrupamento semântico através é a Análise Latente de Dirichlet (LDA). A LDA permite a identificação automática de tópicos avaliando a presença de termos mais comuns em um conjunto de documentos. É possível usar essa estratégia na identificação de tópicos para programas. Basta considerar cada programa como um documento e cada token como um termo. Neste cenário, um possível tópico poderia ser descrito com os termos: "print", "int" e "raw_input". Programas que exploram este tópico costumam ter estes termos presentes.

A LDA é um processo iterativo que necessita de dois parâmetros. O primeiro representa o total de tópicos a serem identificados. Para isso, a sugestão é fazer uso de uma quantidade próxima a de unidades planejadas a serem exploradas em uma disciplina de programação, já que cada unidade representa um tópico distinto sendo explorado em programação. O segundo parâmetro é o número de iterações do algoritmo, já que a cada iteração os pesos dos termos associados a cada tópico são alterados para melhor representarem o corpo de códigos

avaliados. Por representar uma base de dados pequena, o uso de um valor de iteração como a quantidade de documentos existentes permite uma passagem suficientemente adequada para a identificação de tópicos relevantes.

Como resultado, a LDA retorna um conjunto de tópicos, cada um associado a um simplex, ou seja, um vetor de termos com pesos, onde a soma destes pesos é 1. Os termos mais relevantes apresentam maior peso. Como exemplo, um tópico poderia apresentar como termos: $(0,6,\text{print})$, $(0,3,\text{raw\_input})$, $(0,1,\text{float})$. Nesta situação, os temas com este tópico costumam ter primariamente o uso do comando print, aparecendo, em ocorrência com o termo raw_input e, algumas poucas vezes, o uso da função float.

5.3 Estudo Qualitativo

A identificação, visualização e uso do agrupamento semântico são contribuições deste trabalho. Para avaliar o uso destas estratégias, define-se um estudo qualitativo bem definido em três etapas. Esse estudo busca avaliar uma lista de exercícios considerando a proposta apresentada.

A primeira etapa é a captura das questões e códigos a serem abordados. Apesar de ser possível realizar esta avaliação na lista, enquanto as aulas de um semestre ocorrem, optou-se por fazer uso de uma base de códigos consolidada, isto é, de um semestre já finalizado. Isto permite refletir sobre o que se estava explorando ao longo do semestre.

Escolhida a base de dados, a segunda etapa é identificar uma solução relevante, ou seja, representativa da produção dos alunos para cada questão. Em seguida, usa-se uma estratégia automática de identificação de tópicos para detectar como os tokens da linguagem são explorados ao longo da lista de exercícios. Na identificação de tópicos, cada documento (código) é associado a um conjunto de tópicos, cada um representando um conjunto de termos que comumente aparece nos documentos relacionados a determinado tópico. Estes termos são, neste trabalho, tokens da linguagem Python extraídos dos códigos dos alunos.

Por fim, aplica-se a proposta de comparação de código para identificar similaridade entre as soluções de referência das diferentes questões. Com esta comparação é possível ter uma visão geral da semelhança entre questões ao longo da lista.

A partir da análise de tópicos dos documentos, em conjunto com a avaliação da

comparação de códigos de diferentes questões, é possível identificar como os tópicos são explorados ao longo da disciplina, bem como tópicos predominantes de acordo com a produção dos alunos. É objetivo dessa análise permitir que o professor verifique como funciona a distribuição de tópicos dos conteúdos explorados, buscando assim melhorias para a construção da lista de exercícios.

5.3.1 Amostra do Estudo

Para a realização deste trabalho, fez-se uso da produção de 102 alunos de 5 turmas do curso de bacharelado em Ciência da Computação do primeiro semestre de 2012 da Universidade Federal de Campina Grande. Durante esse período, os alunos recebiam novos exercícios na medida que os conteúdos eram expostos. Estes exercícios eram adicionados a uma lista de exercícios única, que serviu como objeto de estudo desta parte do trabalho. Os cinco professores da disciplina colocavam exercícios livremente na lista, focando primeiramente em explorar os conteúdos recentemente vistos em sala, mas era permitido inserir questões de qualquer natureza.

No total, a lista de exercícios avaliada apresentou 193 questões que exigiram a codificação de programas em Python. Cada aluno poderia submeter livremente códigos para as questões que estavam disponíveis na lista de exercícios. Cada submissão era testada automaticamente de forma que, em segundos, o aluno tinha conhecimento sobre se a sua tentativa era aceita por uma bateria de testes automáticos proposta pelos professores. Caso a solução não passasse nos testes propostos, era retornada ao aluno a informação sobre a quantidade de testes em que o programa havia apresentado falhas. No final do semestre 2012.1 foram coletadas 23.692 respostas, ou uma média de 1,2 submissões por aluno para cada questão. Nesse semestre, foram explorados os seguintes tópicos de programação em Python:

- Shells
- Valores, expressões, variáveis
- Comandos e Programas
- Strings e Booleans

- Condicionais
- For
- While
- Funções
- Matrizes
- Tuplas
- Dicionários

5.3.2 Coleta de Dados

A coleta de dados foi direta, sendo a amostra já preparada para processamento.

5.4 Resultados

5.4.1 Identificação da Solução Relevante

Das 23.692 soluções avaliadas, 499 apresentavam erros de sintáxe que impossibilitavam a extração automática de termos e foram descartadas desta avaliação. A partir dos códigos restantes, 193 soluções relevantes foram extraídas, uma para cada questão. Os códigos obtidos foram avaliados de forma a verificar se representavam o que era esperado da produção do aluno. É importante observar que tanto os programas que passavam como os que não passavam nos testes automáticos propostos pelos professores foram avaliados. Isto permite identificar a solução relevante que é central em cada questão principalmente quando os alunos tentam, em sua maioria, alguma estratégia que não seja a melhor ou a esperada pelo professor.

5.4.2 Identificação de Tópicos

Para a escolha da quantidade de tópicos a serem identificados, considerou-se o número de unidades da disciplina (11).

Na Tabela 1 são apresentados os tópicos identificados pela LDA. Cada tema é descrito de acordo com os termos que o compõem e apresentam peso maior ou igual a 0,1. Ainda, na tabela é possível visualizar o número de questões em que este tópico identificado é de maior relevância. A ordem dos tópicos foram definidas de acordo com a ordem em que a primeira questão com aquele tema dominante aparecia na lista de exercícios.

Tabela 5.1: Tópico, quantidades de questões associadas e termos relevantes para a definição do tópico

Tópico	**Questões**	**Termos relevantes**
0	32	$(0, 44, print)$, $(0, 23, raw\_input)$, $(0, 19, float)$
1	35	$(0, 26, print)$, $(0, 23, int)$, $(0, 19, raw\_input)$
2	8	$(0, 40, ==)$, $(0, 29, mod)$, $(0, 16, if)$
3	12	$(0, 18, if)$, $(0, 15, len)$, $(0, 11, >=)$, $(0, 10, <)$, $(0, 10, return)$
4	27	$(0, 24, subscript)$, $(0, 10, split)$
5	40	$(0, 33, subscript)$, $(0, 12, len)$, $(0, 10, ==)$
6	14	$(0, 17, if)$, $(0, 14, break)$, $(0, 14, print)$, $(0, 13, ==)$, $(0, 11, while)$
7	3	$(0, 26, in)$, $(0, 18, return)$, $(0, 15, subscript)$, $(0, 10, if)$
8	32	$(0, 17, subscript)$, $(0, 10, for)$, $(0, 10, del)$, $(0, 10, list)$, $(0, 10, append)$
9	3	$(0, 22, lower)$, $(0, 10, for)$, $(0, 10, if)$
10	1	$(0, 22, split)$, $(0, 15, map)$, $(0, 14, raw\_input)$, $(0, 12, assignment\ tuple)$

Os tópicos 2, 7, 9 e 10 aparecem em menos de 5% das questões da lista. Nestes, o termo mais frequente representa operações que comumente não são exploradas diretamente por nenhuma das unidades, como é o caso de '==', in, 'lower' e 'split'. Isto indica que 15 das questões da lista exploram como tópicos, aspectos de não muito destaque em cursos de

programação (como 'mod', 'lower', e 'split').

Para discutir sobre a relevância dos demais tópicos, foi feito um gráfico associando cada questão a um tema. A Figura 5.2 apresenta essa distribuição de tópicos em cada questão, sendo o eixo X o número da questão e o eixo Y o tópico associado. É possível visualizar os tópicos 0 e 1 aparecendo no começo da lista de exercícios, seguido por questões que exploram os tópicos 4 e 6. Por fim as questões se alternam entre os tópicos 5 e 8. O tópico 3 aparece em questões dispersas ao longo da lista.

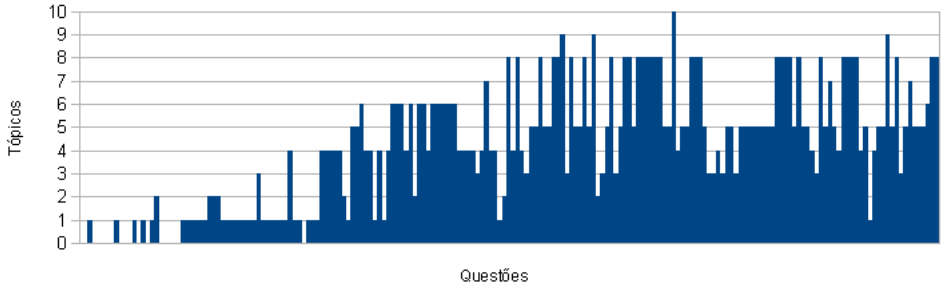


Figura 5.2: Questões e os tópicos detectados

Os dois primeiros tópicos explorados apresentam *print* como principal construção e um grande peso para o uso do *raw_input*. Depois destes dois tópicos, há uma concentração de questões que exploram o tópico 4. Este tópico apresenta acesso a itens e o uso da operação *split* como termos predominantes, o que indica o uso de códigos com strings. Pela estrutura planejada do curso, o comando condicional *if* deveria estar presente, no entanto, ele, por si, não parece um tópico como unicamente relevante tendo poucos representantes no tópico 3 especialmente no começo da disciplina.

Após condicionais, foi planejado para a disciplina explorar o conteúdo de laços de repetição com *for*. Novamente, não apareceu um tópico predominante com este termo, indicando que ele por si é explorado de forma diluída em outros tópicos, como a própria iteração em strings (tópico 4).

Alternando com as questões que fazem uso da análise de strings, o tópico 6 aparece como sendo predominantemente explorado. Neste tópico, o comando condicional aparece como termo principal, mas difere do tópico 3 pela presença dos termos *break*, *print*, == e *while*. Isto acontece porque, pelo conteúdo programado, o aluno passa agora a explorar o comando

de repetição *while*. É importante observar a presença do *if* e *break* nas questões planejadas de *while*. Ao examinar as soluções existentes, observa-se que os programas que fazem uso do termo *while* também costumam definir condições de parada adequadas (com *if* e *break*).

Em seguida, ao contrário do planejado pelo curso, não foi possível detectar um tópico único para o tratamento de funções. Esperava-se que, questões de criações de funções poderiam aparecer como tópico principal, com o uso do termo def (definição de funções). Isto indica que o uso de funções foi um conteúdo novamente explorado nos demais tópicos presentes. Aparece, no entanto, e até o final do curso, a predominância dos tópicos 5 e 8.

Em ambos os tópicos 5 e 8, é possível ver o termo subscript (acesso a item) como o mais relevante. Eles diferem do tópico 4 por terem como termos correlatos que não estão associados a um tipo de dados como strings. Ao analisar os códigos referentes a este tópico, descobriu-se que o estudo de funções está associado intimamente com acesso e alterações de listas e de matrizes (listas de listas). Ainda, por dicionários e tuplas serem exploradas da mesma forma do ponto de vista sintático da linguagem, eles não aparecem em si como tópicos relevantes.

5.4.3 Comparação de Similaridade entre Questões

Espera-se da similaridade entre-questões que as questões de mesmo tópico sejam semelhantes entre si. Para visualizar isto, nós desenhamos o mapa de calor de similaridade entre as questões da lista. A linha diagonal representa similaridade máxima (a questão com ela mesma) e a matriz é simétrica por ser um mapa de questões x questões. Para cada célula, quanto mais clara a cor, maior similaridade. Acima do mapa de calor, está o gráfico da distribuição de tópicos para cada questão como mostra a Figura 5.3. Ao cruzar a informação dos tópicos detectados e o gráfico de similaridade, é possível observar uma maior similaridade existente das questões de um mesmo tópico. Isto acontece pela própria estratégia de agrupamento, que faz uso de elementos sintáticos para identificação dos tópicos.

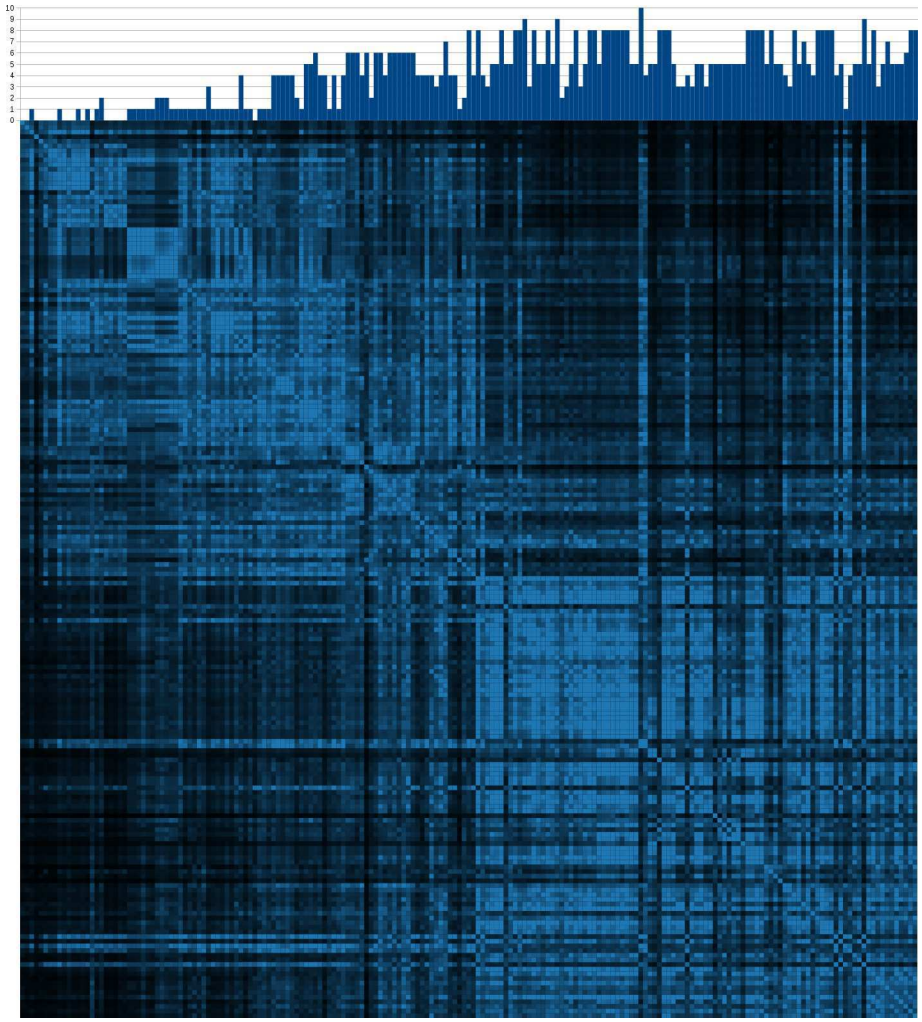


Figura 5.3: Similaridade entre questões e os tópicos associados

Entretanto, o resultado mais surpreendente acontece quando consideramos dois grandes blocos que são extremamente semelhantes entre si. Estes blocos, 1 e 2, estão destacados na Figura 5.4. Ao investigar as soluções de referência das questões destes blocos, nós encontramos que, exatamente na metade do curso, as questões deixavam de explorar aspectos de entrada-e-saída para explorar predominantemente funções. Enquanto não foi possível fazer a identificação automática do tópico semântico próprio para funções, a avaliação de similaridade ajuda para identificar a presença deste tópico de forma distribuída entre os exercícios.

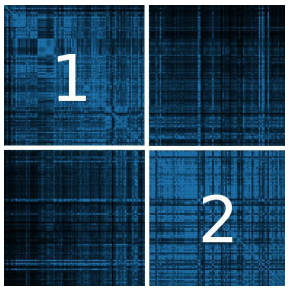


Figura 5.4: Dois grupos de questões detectados

5.5 Discussão

A própria avaliação dos tópicos encontrados permitiu observar como os alunos exploraram determinado tema da lista de exercícios. Foi comum que os alunos fizessem o uso do comando de impressão (print) em dois tópicos distintos. Em um destes tópicos, o print estava associado à função int de conversão de string para inteiro. Em outro tópico, o print aparecia associado à função float para converter strings em números com ponto flutuante. Assim, se estes dois tópicos foram detectados é porque há, na lista de exercícios, questões que exploram essencialmente o uso do print, ora com inteiros, e em outros momentos com conversão para ponto flutuante.

Ao associar cada questão da lista de exercícios com os tópicos principais daquelas questões, foi possível ver como estes tópicos eram explorados ao longo da disciplina. Com isso, permitiu-se que se identificasse temas importantes que não estavam sendo suficientemente explorados, principalmente a título de revisão, ou tópicos que eram cobertos excessivamente sem necessidade.

A experiência mais positiva com o experimento foi na proposta de visualização apresentada como mapa de calor. Com esta visualização, foi possível identificar grupos de questões próximas, grandes blocos de questões parecidas e pouco retorno (semelhança) às questões iniciais. De posse dessa informação, o professor pode tentar detectar onde estão os erros mais comuns dos alunos, se em algum destes grupos ou questões parecidas, avaliar os tópicos abordados e dar destaque a este conteúdo na sua disciplina.

Enquanto o estudo realizado faz a comparação baseando-se nas soluções propostas pelo aluno, seria possível aplicar o mesmo estudo utilizando soluções de referência geradas

pelo professor (gabarito). Poderia também comparar o quão próxima tais soluções são das soluções de referência, e identificar qual o comportamento dos tópicos e similaridade das questões utilizando as soluções geradas pelo professor.

5.6 Limitações do Estudo

O estudo realizado foi feito para um único semestre. Cada semestre ou lista de exercícios apresenta um potencial único para diferentes descobertas a serem feitas. Por exemplo, o comportamento de detecção de grandes agrupamentos pode não ser visualizável em outros semestres.

Por ser um estudo qualitativo, pouco se generaliza sobre as conclusões deste trabalho, em especial da qualidade da detecção de tópicos gerados, sendo necessária a realização de um experimento mais controlado para atestar a real qualidade dos resultados produzidos pela classificação de tópicos.

5.7 Trabalhos Relacionados

O uso da técnica de identificações de tópicos já foi abordado na literatura para a identificação de tópicos de um conjunto de classes (pacote) [Maletic and Marcus 2000] utilizando LSI. Este trabalho de doutorado faz uso de uma técnica moderna, a LDA, para a identificação de tópicos de diferentes questões. Ao contrário do que é feito na literatura, há uma etapa adicional, a de identificação da solução relevante, antes da etapa de identificação de tópicos.

A técnica de comparação de códigos em uma lista de exercícios já foi usada anteriormente por Maciel et. al [Maciel et al. 2013]. Mas este último trabalho não passava pela identificação de uma solução relevante e representativa para cada questão e nem estruturava o mapa-de-calor como visualização guia para a análise dessas comparações. Ainda, não combinava a informação do mapa-de-calor com a informação de identificação de tópicos.

5.8 Sumário

Este capítulo abordou o uso da comparação de código para avaliação de uma lista de exercícios por meio da análise das soluções produzidas pelos alunos às questões da lista. Para tanto, foi preciso inicialmente detectar entre as soluções produzidas pelos alunos aquelas consideradas como mais relevantes. Para tanto, foi utilizada a própria detecção de similaridade de códigos para conseguir capturar tal noção.

Gerado o código de referência para cada questão, utilizaram-se técnicas de detecção automática de tópicos para identificar os tópicos de cada questão, bem como os termos predominantes destes tópicos. Foi possível ver, por exemplo, o uso inicial do comando de impressão em tela nos tópicos iniciais da disciplina.

Em seguida, avaliou-se a similaridade entre questões através de um mapa de calor. Esta forma visual permitiu identificar dois grandes grupos na lista de exercício explorada, bem como a presença de pequenos grupos contínuos, isto é, a tendência de publicar exercícios com soluções parecidas.

Capítulo 6

Conclusões

Este trabalho de tese propõe o uso de técnicas automáticas para a comparação e agrupamento semântico de programas, bem como a visualização destas comparações, seja de programas de uma mesma questão ou de diferentes questões. Inicialmente se buscou observar a concordância entre a forma de comparar das estratégias automáticas e a de professores. Foi descoberto que os professores não necessariamente concordam tanto entre si na maneira de comparar códigos e que estratégias automáticas podem capturar esta noção do professor, se definido um parâmetro adequado que determina o que tais estratégias devem considerar como soluções próximas.

Em seguida, foi feito o uso da comparação de código para enriquecer a experiência da avaliação de códigos entre os alunos por meio de um suporte ferramental desenvolvido ao longo deste trabalho de doutorado. Num experimento controlado com duas questões a serem avaliadas, os alunos não apresentaram uma melhora na qualidade da revisão com o uso dessa informação adicional, mas se sentiram mais aptos a revisar códigos e consideraram a ferramenta fácil de ser utilizada.

Por fim, foi feito um estudo qualitativo do uso de comparação de códigos para avaliar a própria lista de exercícios. Neste estudo, foi possível detectar, através inicialmente da detecção automática de tópicos, a distribuição de unidades e como certas questões exploravam certos elementos sintáticos da linguagem. Também, através da detecção de similaridade entre códigos, foi possível detectar padrões na publicação de exercícios em uma lista, como a publicação de exercícios em bloco e pouco retorno aos conteúdos iniciais.

6.1 Contribuições

A primeira contribuição do trabalho está na pesquisa sobre comparações de código. A pesquisa constrói uma função de comparação que, nas estratégias automáticas, ajuda a modelar o conceito de códigos que estão muito próximos e podem ser considerados similares. Adicionando este conceito, é possível capturar, com estratégias automáticas, a noção de comparação utilizada pelo professor.

A segunda contribuição do trabalho está na visualização das soluções de um problema de maneira automatizada por meio do suporte ferramental apresentado neste trabalho. Esta visualização permite identificar diferentes soluções e aquelas soluções que são parecidas. O aluno pode fazer uso desta visualização para ter feedback e para guiar seu processo de avaliação. O uso de tal visualização aumenta a confiança do aluno na sua capacidade de avaliar código, e uma ferramenta com tal visualização é simples de ser utilizada.

A terceira contribuição da tese está na visualização das relações entre soluções das diferentes questões da lista de exercícios e do agrupamento semântico destas soluções. É possível identificar quais são as soluções que representam primariamente as questões da lista e é possível, utilizando um mapa de calor, saber como estas questões se relacionam, levando a uma reflexão sobre a própria lista de exercícios propostos. Ainda, com a identificação dos tópicos das questões, é possível identificar os tópicos que são muito ou pouco explorados numa lista de exercícios.

6.2 Trabalhos Futuros

A experimentação proposta por este trabalho foi limitada a um conjunto pequeno de alunos e questões. É trabalho futuro fazer uso das contribuições desta tese em um sistema web. Com o uso massivo, pretende-se investigar o comportamento dos resultados apresentados em diferentes turmas, níveis de maturidade dos alunos, questões e outros cenários, bem como investigar novas possibilidades de interação, visualização e apresentação do feedback. O efeito do feedback gerado pode ser mais facilmente testado num ambiente massivo permitindo avaliar em um grupo maior de professores e alunos a utilidade dos resultados gerados por este trabalho. Para professores, por exemplo, pretende-se analisar o quanto um

suporte ferramental como o que é proposto neste trabalho para visualização de soluções pode ser útil no processo de correção de diferentes soluções de alunos e na geração de feedback para esses alunos.

Outro estudo planejado é a investigação do efeito do uso contínuo do feedback provido pela comparação de códigos. Identificar se o aluno, ao ser submetido diversas vezes ao ambiente de avaliação de código entre pares, faz melhor uso da informação da comparação de códigos.

Pretende-se submeter diferentes corpus de exercícios ao processo de comparação de soluções e de identificação de tópicos buscando identificar diferentes estratégias existentes entre as diferentes universidades ou sistemas online de exercícios. Com este resultado espera-se a construção de um cátalogo de práticas de construções de listas e permitir que as questões possam ser identificadas e analisadas com a informação complementar do tópico a que ela pertence.

6.3 Outras Implicações da Tese

O três estudos apresentados corroboram com a tese de que técnicas de comparação de códigos podem ser utilizadas para a comparação geral de uma população de códigos, bem como de que é possível representar tais comparações através de visualizações a serem utilizadas para gerar feedback ao aluno e informação que auxilia o diagnóstico do professor sobre os exercícios de programação produzidos.

O uso das estratégias automáticas de comparação de códigos, com um parâmetro limite para a detecção do que é semelhante, pode implicar em melhores resultados para as pesquisas já consolidadas em detecção de plágio e avaliação automática de código. Um professor poderia, por exemplo, através da execução da ferramenta apresentada no primeiro experimento, ter informação de qual abordagem mais se aproxima do seu estilo de comparar códigos, de forma fazer uso dessa técnica nos contextos já citados de avaliação automática e detecção de plágio.

Apesar da validação do experimento com uma quantidade pequena de dados, a mesma comparação apresenta-se como escalável para MOOCs, o que permitiria a identificação de comportamentos semelhantes de soluções numa escala ainda maior. Esta mesma avaliação

em larga escala é ainda mais benéfica para sistemas que recebem questões de forma livre para a avaliação de tópicos automática ou a identificação de questões semelhantes que poderiam vir a ser sugeridas pelos alunos. Isto gera informação que permite rever o programa ou os execícios alocados ou sugeridos num MOOC.

Pensando na personalização do ensino, as técnicas de comparação de códigos e identificação de tópicos podem ser usadas para recomendar códigos de exemplo sobre determinado tema ou recomendar questões para os alunos baseando-se em questões similares feitas pelos mesmos. Neste sentido, a própria experiência em um MOOC e em outras ferramentas de submissão de códigos podem se tornar cada vez mais significativas e proveitosas.

A estratégia de visualização pode ser utilizada em outros contextos. A visualização de soluções de uma questão recebe uma matriz com os pares de códigos e o valor de semelhança entre os mesmos. Nada impede que sejam utilizadas outras métricas de semelhança para gerar visualizações de diferentes aspectos de um conjunto de soluções. Por exemplo, é possível colocar como métrica de diferença entre códigos, o valor absoluto da diferença dos tempos de execuções destas soluções para uma entrada em comum. Neste modelo, soluções com desempenhos parecidos estarão agrupadas próximas uma das outras. O professor pode usar mesmo a diferença de notas atribuídas manualmente para os códigos de forma a dar para turma um feedback de soluções consideradas ideais, mesmo que sintaticamente distintas.

Apêndice A

Webshell

O primeiro trabalho de pesquisa, publicado no WEI 2013, explora o acompanhamento passo-a-passo do processo de codificação do aluno. Nessa pesquisa, desenvolveu-se uma ferramenta de programação on-line, o Webshell, que permitia ao aluno executar, testar e submeter aos professores códigos referentes a exercícios de programação.

A partir dos dados gerados pelo Webshell, foi possível construir um gráfico representando a quantidade de caracteres ao longo do tempo de codificação, indicando, em tracejados verticais, os momentos em que o aluno executa seu código. Como exemplo, na Figura A.1, é possível visualizar um aluno que, para resolver um determinado problema, levou aproximadamente 1200 segundos e que construiu uma solução de pouco mais do que 300 caracteres. Esse estudo, ainda preliminar, permite a visualização de diferentes momentos da codificação. É possível ver na mesma figura que o aluno não imediatamente começa codificando sua solução, mas passa por um período de inatividade que representa o momento que o aluno lê o enunciado da questão. Em seguida, é possível ver um aumento do número de caracteres até o momento que o aluno executa pela primeira vez seu código, encontra um possível erro e procura trabalhar na resolução deste erro.

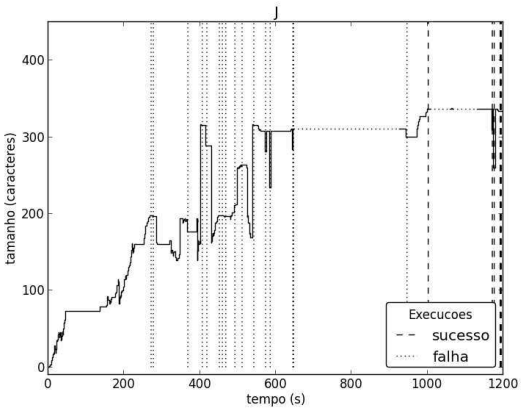


Figura A.1: Exemplo da análise do processo de codificação do aluno capturado pelo Webshell.

Enquanto a pesquisa sobre os resultados gerados por este trabalho ainda continua, a mesma é relevante para ajudar no entendimento do processo de compreensão da codificação do aluno. Em um experimento controlado, foi possível visualizar três etapas bem definidas do processo de codificação: a compreensão do problema, implementação, e verificação e teste da solução. A visualização do próprio processo de codificação permite também identificar alunos que possam estar passando por problemas de codificação, seja ao identificar que o código está sofrendo diversas inserções e remoções em pouco espaço de tempo, seja pela ausência da atividade de edição de código durante um longo período de tempo, por exemplo."

Apêndice B

Hábitos de Estudo e Desempenho do Aluno

Outro trabalho, publicado também no WEI 2013, busca compreender a relação entre o desempenho dos estudantes de uma turma de ensino de programação e o hábito de estudo desses alunos. Tais hábitos de estudo representam o tempo total de horas dedicadas para a resolução de exercícios propostos, o número de exercícios resolvidos, e a quantidade de dias com trabalho dos estudantes. Como resultados, confirmou-se resultado existente na literatura de que a quantidade de exercícios resolvidos é um fator extremamente relevante e correlacionado ao desempenho do aluno (correlação de $0,76$). Ainda, o tempo total gasto representa apenas a terceira maior correlação com o desempenho do aluno (correlação de $0,48$), sendo o estudo diário a segunda maior correlação com o desempenho dos estudantes (correlação de $0,62$).

Apêndice C

TSTView

Por fim, o terceiro trabalho derivado trata do TSTView, que é uma ferramenta de acompanhamento dos alunos em sala de aula a partir dos dados de resoluções de exercícios dos alunos. Numa turma de programação os alunos resolviam uma lista de exercícios e enviavam suas respostas para um testador automático. O TSTView, por sua vez, pega os resultados dos exercícios resolvidos pelos alunos, processa, e exibe em duas visões para serem utilizadas pelo professor e pelo aluno. Na visão do professor, exibida na Figura C.1, era possível acompanhar o histórico de exercícios resolvidos por um conjunto de alunos (uma turma), visualizando rapidamente quais as questões resolvidas pelos alunos, quais as questões que os alunos tentaram resolver mas não conseguiram êxito e qual o código submetido em cada tentativa de resolução.

Figura C.1: Tela do TSTView para acompanhamento dos alunos na visão do professor.

Com este feedback, o professor podia tomar uma ação rápida para determinar uma intervenção na turma. Nesse trabalho, avaliou-se o impacto do uso da ferramenta durante o acompanhamento de mini-testes. Durante um exercício avaliativo, os professores

determinavam um limite de tempo para a resolução de uma determinada questão. Um dos resultados encontrados é que o professor comumente não planejava um tempo adequado para a resolução do exercício para a turma. Por vezes, um tempo adicional era fornecido para a realização da avaliação e este tempo foi útil para que mais alunos pudessem conseguir submeter uma solução a ser avaliada. Era com o acompanhamento da tarefa que o professor podia visualizar a quantidade de respostas submetidas até o momento e tomar esta ação.

O TSTView também permite o acompanhamento individual na visão do aluno, como mostra a Figura C.2. Cada aluno tinha acesso ao seu histórico e, através deste, podia acompanhar seu progresso individual e comparar com o ritmo de produção de sua turma. Essa visão também permite o diálogo do professor num acompanhamento individual. Se um aluno apresentava um desempenho insatisfatório nos minitestes, isto era confrontado com seu histórico de submissões e sua prática de estudo durante os dias letivos. Quantas vezes o aluno tenta submeter exercícios por semana? Como isto se compara com a turma? Confrontado com isto, uma das primeiras ações era dialogar com o aluno para que o mesmo refletisse sobre seu processo de aprendizagem.

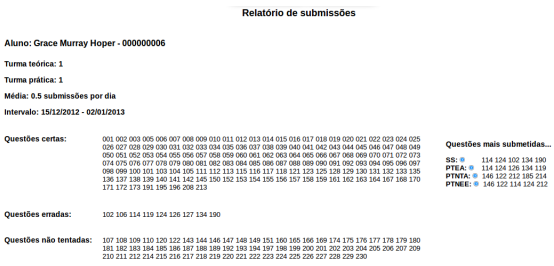


Figura C.2: Tela do TSTView para acompanhamento individual dos alunos.

Apêndice D

Dados do Experimento de Comparação de Códigos

Neste anexo, são apresentados na Tabela D.1 os valores dos votos que os professores realizavam, no experimento descrito no Capítulo 3, para decidir, dada uma tripla de códigos R, C1 e C2, qual código era mais similar: C1 (-1), C2 (1) ou ambos são igualmente similares a R (0). Cada código votado pode ser descrito pelo seu tipo, onde **sem** representa o código direto, sem funções; **fun** representa o código com uma função, e; **rec** que é o código que calcula Fibonacci a partir de uma chamada recursiva. Cada código também vem com a variação utilizada (ou nada, caso seja o código original).

Tabela D.1: Votos dos professores para cada tripla de códigos

REF	C1	C2	Professores										
rec s2d	rec s3a	sem s1a	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
rec s2d	rec s3e	rec s4b	0	1	1	1	-1	0	1	-1	1	0	0
rec s2d	rec s3e	sem s2a	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
rec s2d	sem s2b	fun s3d	1	0	1	1	0	0	1	1	0	0	0
rec s2d	sem s2d	rec s3e	1	1	1	1	1	1	1	1	1	1	1
rec s3e	rec s2a	fun s3d	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
rec s3e	rec s3a	fun s3e	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
rec s3e	rec s3d	rec s2b	0	0	0	1	0	0	0	0	0	0	0
rec s3e	rec s4b	rec s2d	0	0	1	0	1	0	-1	0	1	0	0
rec s3e	fun s3e	rec s3a	1	1	1	1	1	1	1	1	1	1	1
rec s4b	rec s2d	fun s3d	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
rec s4b	fun s1a	rec s2d	1	1	1	1	1	1	1	0	1	1	1
rec s4b	fun s2d	sem s3c	0	0	0	-1	-1	-1	-1	-1	-1	0	0
rec s4b	sem s3e	sem s4b	0	0	0	0	0	0	1	1	0	0	0
rec s4b	sem s3e	rec s3e	1	1	1	1	1	1	0	1	1	1	1
fun s2d	rec s1a	fun s3e	1	1	1	1	1	1	1	1	1	1	1
fun s2d	rec s3a	rec s3c	0	0	0	0	0	0	0	0	1	0	0
fun s2d	fun s3e	sem s3e	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
fun s2d	sem s2b	rec s3a	0	-1	-1	1	-1	1	0	1	-1	0	-1
fun s2d	sem s3a	rec s1a	0	-1	-1	0	-1	-1	1	1	-1	0	1
sem s2b	rec s1a	rec s3e	0	0	0	0	0	0	0	0	0	0	0
sem s2b	rec s3a	rec s2b	0	0	0	0	0	0	0	0	0	0	0
sem s2b	fun s3d	rec s3e	-1	-1	-1	-1	-1	0	0	-1	-1	0	0
sem s2b	sem s3d	sem s3c	0	0	0	-1	0	0	0	0	-1	0	0
sem s2b	sem s4b	rec s3e	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
sem s3a	rec s1a	fun s3e	0	1	1	0	1	0	0	1	1	0	0
sem s3a	rec s2d	rec s1a	0	0	0	0	0	0	0	0	0	0	0
sem s3a	rec s2d	rec s2a	0	0	0	0	0	0	0	-1	0	0	0
sem s3a	rec s3a	rec s3e	0	0	0	0	0	0	0	1	0	0	0
sem s3a	rec s3a	sem s3b	1	1	1	1	1	1	1	1	1	1	1
sem s3a	rec s3c	rec s2d	0	0	0	0	0	0	0	-1	0	0	0
sem s3a	rec s3d	sem s2d	1	1	1	1	1	1	0	1	1	1	1
sem s3a	fun s1a	sem s4d	1	1	-1	-1	1	0	1	0	-1	1	1
sem s3a	fun s2d	fun s4d	0	-1	0	0	0	1	0	0	-1	0	0
sem s3a	fun s3d	sem s2b	1	1	1	1	1	1	1	0	1	1	1
sem s3a	sem	rec s4b	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1
sem s3a	sem s1a	rec s2d	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
sem s3a	sem s1a	rec s3a	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
sem s3a	sem s2d	sem s4b	0	0	0	0	0	-1	0	0	0	0	0
sem s3a	sem s4d	rec s3e	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1

Apêndice E

Dados do Experimento da Avaliação de Códigos entre-pares

Neste anexo, são apresentados na Tabela E.1 os valores extraídos do experimento realizado no Capítulo 4.

Tabela E.1: Dados dos alunos no experimento de avaliação de códigos entre pares

		Alunos																												
Tratamento	Grupo	E	C	C	C	E	E	E	C	E	C	C	C	E	E	E	E	C	C	E	E	E	C	C	E	C	C	E	C
Formulário	1	3	3	4	3	4	4	3	4	2	2	3	4	3	4	4	2	3	3	4	3	4	4	3	4	4	4	3	3
	2	5	4	2	5	3	4	4	3	5	5	1	3	4	3	3	1	3	2	3	3	3	3	4	4	3	3	3	4
	3	3	4	4	4	2	4	4	3	3	2	3	3	3	4	3	4	2	3	4	4	4	4	3	4	4	3	4	4
	4	5	5	4	5	5	5	5	5	5	5	4	4	5	5	5	3	3	4	5	5	4	5	5	5	5	4	5	5
	5	5	5	5	5	5	5	5	4	5	5	5	4	4	5	4	5	4	5	5	4	5	5	3	4	4	5	4	5
	6	5	3	5	4	5	5	4	5	5	5	5	4	5	5	5	5	4	5	5	5		5	4	4	5	5	5	5
	7	3	2	5	4	4	5	5	4	4	5	5	4	3	5	5	5	3	5	5	5		5	4	5	5	5	3	5
	8	2	1	4	3	4	4	5	2	3	5	2	3	3	4	3	5	2	4	4	3	4	3	2	4	3	4	4	4
	9	3	4	4	2	4	4	5	4	5	1	2	3	4	4	3	5	2	4	5	5		5	3	4	3	5	5	5
	10	5	3	3	2	3	3	1	3	5	3	5	3	3	5	2	5	3	4	5	5	4	5	5	4	3	5	2	5
	11	1	4	3	1	2	1	4	4	1	3	4	2	3	4	3	5	3	4	4	3	4	4	2	2	2	5	4	4
	12	5	2	5	5	3	4	5	5	5	1	4	3	4	5	5	5	5	4	3	4	5	5	4	3	1	5	5	5
	13	5	2	4	4	2	4	5	5	5	2	4	3	5	5	4	5	5	4	3	5	4	5	4	4	3	5	4	5
1.1	A	1	1	0	1	1	0	1	1	1	1	1	0	0	1	0	1	1	1	1	0	2	1	1	1	1	1	1	0
	E	1	0	0	0	0	1	0	1	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	2
	N	0	1	0	0	1	0	0	0	2	0	0	1	1	0	1	0	0	3	1	1	0	0	0	0	0	0	0	0
1.2	1.2A	0	1	0	1	1	0	1	0	0	1	0	0	0	1	1	0	0	1	0	0	0	1	1	0	1	0	1	0
	1.2E	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1.2N	0	0	0	0	0	0	0	0	1	0	0	1	0	0	1	2	1	1	1	1	0	0	0	0	0	0	1	1
2.1	2.1A	1	2	0	2	0	2	1	0	2	2	0	0	0	2	0	2	1	0	0	1	2	2	1	1	2	2	2	1
	2.1E	0	0	0	0	1	0	0	0	0	0	0	0	0	0	2	0	0	1	0	0	0	0	0	0	0	0	0	0
	2.1N	0	0	0	0	0	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
2.2	2.2A	1	2	0	2	0	2	0	0	2	2	0	0	0	2	2	2	0	0	0	1	2	2	1	0	1	2	2	2
	2.2E	0	0	0	0	2	0	1	0	0	0	0	0	0	1	1	1	1	2	0	0	0	0	0	0	0	1	0	0
	2.2N	0	0	0	1	0	0	0	0	0	0	0	0	0	2	1	0	0	0	0	0	0	0	0	0	0	0	1	0
Corretude	Q1	0,00	0,00	0,50	1,00	0,00	0,75	0,00	1,00	0,50	0,75	0,00	0,75	1,00	1,00	1,00	1,00	0,00	0,00	0,75	0,75	0,00	1,00	1,00	0,00	1,00	0,00	1,00	0,75
	Q2	0,22	0,00	0,00	0,00	0,00	0,00	0,00	0,22	0,22	0,33	0,22	0,00	0,33	0,33	0,22	1,00	0,11	0,00	0,00	0,11	1,00	1,00	0,22	0,78	0,00	1,00	1,00	1,00

Bibliografia

[Bancroft and Roe 2006] Bancroft, P. and Roe, P. (2006). Program annotations: Feedback for students learning to program. In *Proceedings of the 8th Australasian Conference on Computing Education - Volume 52*, ACE '06, pages 19–23, Darlinghurst, Australia, Australia. Australian Computer Society, Inc.

[Ben-Ari 1998] Ben-Ari, M. (1998). Constructivism in computer science education. *SIGCSE Bull.*, 30(1):257–261.

[Benford et al. 1993] Benford, S., Burke, E., and Foxley, E. (1993). Learning to construct quality software with the ceilidh system. *Software Quality Journal*, 2(3):177–197.

[Blei et al. 2003] Blei, D. M., Ng, A. Y., and Jordan, M. I. (2003). Latent dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022.

[Borges, M. A. F. 2000] Borges, M. A. F. (2000). Avaliação de uma Metodologia Alternativa para a Aprendizagem de Programação. In *Workshop de Educação em Computação, Congresso anual da SBC 2000*, Curitiba, Brasil. SBC.

[Cardoso 2011] Cardoso, A. C. S. (2011). Feedback em contextos de ensino-aprendizagem on-line. *Linguagens e Diálogos*, 2(2):17–34.

[Chamillard and Braun 2000] Chamillard, A. T. and Braun, K. A. (2000). Evaluating programming ability in an introductory computer science course. In *Proceedings of the thirty-first SIGCSE technical symposium on Computer science education*, SIGCSE '00, pages 212–216, New York, NY, USA. ACM.

[Chinn 2005] Chinn, D. (2005). Peer assessment in the algorithms course. In *Proceedings of*

the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, ITiCSE '05, pages 69–73, New York, NY, USA. ACM.

[Clancy et al. 2003] Clancy, M., Titterton, N., Ryan, C., Slotta, J., and Linn, M. (2003). New roles for students, instructors, and computers in a lab-based introductory programming course. In *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '03, pages 132–136, New York, NY, USA. ACM.

[Cummins et al. 2010] Cummins, S., Burd, L., and Hatch, A. (2010). Tag based feedback for programming courses. *SIGCSE Bull.*, 41(4):62–65.

[Dalmolin et al. 2009] Dalmolin, L. C. D., Nassar, S. M., Bastos, R. C., and Mateus, G. P. (2009). A concept map extractor tool for teaching and learning. *Advanced Learning Technologies, IEEE International Conference on*, 0:18–20.

[Daly and Waldron 2004] Daly, C. and Waldron, J. (2004). Assessing the assessment of programming ability. In *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '04, pages 210–213, New York, NY, USA. ACM.

[Darling-Hammond 2008] Darling-Hammond, L. (2008). Teacher learning that supports student learning. *Teaching for intelligence*, pages 92–93.

[Donaldson et al. 1981] Donaldson, J. L., Lancaster, A.-M., and Sposato, P. H. (1981). A plagiarism detection system. In *ACM SIGCSE Bulletin*, volume 13, pages 21–25. ACM.

[Douce et al. 2005] Douce, C., Livingstone, D., and Orwell, J. (2005). Automatic test-based assessment of programming: A review. *Journal on Educational Resources in Computing (JERIC)*, 5(3):4.

[du Boulay and Sothcott 1987] du Boulay, B. and Sothcott, C. (1987). Artificial intelligence and education; vol. 1: Learning environments and tutoring systems. chapter Computers Teaching Programming: An Introductory Survey of the Field, pages 345–372. Ablex Publishing Corp., Norwood, NJ, USA.

[Glassman et al. 2014] Glassman, E. L., Scott, J., Singh, R., Guo, P., and Miller, R. (2014). Overcode: Visualizing variation in student solutions to programming problems at scale.

In *Proceedings of the Adjunct Publication of the 27th Annual ACM Symposium on User Interface Software and Technology*, UIST'14 Adjunct, pages 129–130, New York, NY, USA. ACM.

[Glassman et al. 2015] Glassman, E. L., Scott, J., Singh, R., Guo, P. J., and Miller, R. C. (2015). Overcode: Visualizing variation in student solutions to programming problems at scale. *ACM Trans. Comput.-Hum. Interact.*, 22(2):7:1–7:35.

[Hage and P. Vugt 2010] Hage, J. R. and P. Vugt, N. (2010). A comparison of plagiarism detection tools. In *Technical Report UU-CS-2010-015*. Department of Information and Computing Sciences Utrecht University, Utrecht, The Netherlands.

[Ihantola et al. 2010] Ihantola, P., Ahoniemi, T., Karavirta, V., and Seppälä, O. (2010). Review of recent systems for automatic assessment of programming assignments. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*, Koli Calling '10, pages 86–93, New York, NY, USA. ACM.

[Joy et al. 2005] Joy, M., Griffiths, N., and Boyatt, R. (2005). The boss online submission and assessment system. *J. Educ. Resour. Comput.*, 5(3).

[Lemos 1979] Lemos, R. S. (1979). Teaching programming languages: A survey of approaches. *SIGCSE Bull.*, 11(1):174–181.

[Lister and Leaney 2003] Lister, R. and Leaney, J. (2003). Introductory programming, criterion-referencing, and bloom. In *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '03, pages 143–147, New York, NY, USA. ACM.

[Maciel et al. 2013] Maciel, D., França, A., and Soares, J. (2013). Sistema de apoio a atividades de laboratório de programação via moodle com suporte ao balanceamento de carga e análise de similaridade de código. *Revista Brasileira de Informática na Educação*, 21(01).

[Maletic and Marcus 2000] Maletic, J. I. and Marcus, A. (2000). Using latent semantic analysis to identify similarities in source code to support program understanding. In *Tools*

with Artificial Intelligence, 2000. ICTAI 2000. Proceedings. 12th IEEE International Conference on, pages 46–53.

[Manning et al. 2008a] Manning, C. D., Raghavan, P., and Schütze, H. (2008a). *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA.

[Manning et al. 2008b] Manning, C. D., Raghavan, P., and Schütze, H. (2008b). *Introduction to information retrieval*, volume 1. Cambridge University Press Cambridge.

[Marshall 2012] Marshall, L. (2012). A comparison of the core aspects of the acm/ieee computer science curriculum 2013 strawman report with the specified core of cc2001 and cs2008 review. In *Proceedings of Second Computer Science Education Research Conference*, CSERC '12, pages 29–34, New York, NY, USA. ACM.

[Meyer et al. 2011] Meyer, C., Heeren, C., Shaffer, E., and Tedesco, J. (2011). Comoto: The collaboration modeling toolkit. In *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education*, ITiCSE '11, pages 143–147, New York, NY, USA. ACM.

[Mihaescu et al. 2006] Mihaescu, R., Levy, D., and Pachter, L. (2006). Why neighbor-joining works. *CoRR*, abs/cs/0602041.

[Moro et al. 2013] Moro, L. F., Lopes, A. M. Z., Delbem, A. C. B., and Isotani, S. (2013). Os desafios para minerar dados educacionais de forma rápida e intuitiva: o caso da DAMICORE e a caracterização de alunos em ambientes de eLearning. In *II Workshop de Desafios da Computação Aplicada à Educação, DesafIE! 2013*, Maceió, Brasil. SBC.

[Naudé 2007] Naudé, K. A. (2007). *Assessing program code through static structural similarity*. PhD thesis, Nelson Mandela Metropolitan University.

[Ohmann 2013] Ohmann, A. (2013). Efficient Clustering-based Plagiarism Detection using IPPDC. *Thesis, College of Saint Benedict/Saint John's University, Minnesota, Estados Unidos*.

[Pea and Kurland 1984] Pea, R. D. and Kurland, D. M. (1984). On the cognitive effects of learning computer programming. *New ideas in psychology*, 2(2):137–168.

[Pears et al. 2007] Pears, A., Seidman, S., Malmi, L., Mannila, L., Adams, E., Bennedsen, J., Devlin, M., and Paterson, J. (2007). A survey of literature on the teaching of introductory programming. *SIGCSE Bull.*, 39(4):204–223.

[Petit et al. 2012] Petit, J., Giménez, O., and Roura, S. (2012). Jutge.org: An educational programming judge. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, SIGCSE '12, pages 445–450, New York, NY, USA. ACM.

[Rahman et al. 2008] Rahman, K., Nordin, M. J., and Che, W. (2008). Automated programming assessment using the pseudocode comparison technique: Does it really work? In *Information Technology, 2008. ITSim 2008. International Symposium on*, volume 3, pages 1–4. IEEE.

[Roberts et al. 1999] Roberts, E., Shackelford, R., LeBlanc, R., and Denning, P. J. (1999). Curriculum 2001: Interim report from the acm/ieee-cs task force. *SIGCSE Bull.*, 31(1):343–344.

[Robins et al. 2003] Robins, A., Rountree, J., and Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer Science Education*, 13(2):137–172.

[Roy 2009] Roy, C. K. (2009). *Detection and analysis of near-miss software clones*. PhD thesis, Queen's University, Kingston, Ont., Canada, Canada. AAINR65337.

[Roy and Cordy. 2007] Roy, C. K. and Cordy., J. R. (2007). A survey on software clone detection research. In *Technical Report 541*. Queen's University at Kingston.

[Sahami et al. 2013] Sahami, M., Roach, S., Cuadros-Vargas, E., and LeBlanc, R. (2013). Acm/ieee-cs computer science curriculum 2013: reviewing the ironman report. In *Proceeding of the 44th ACM technical symposium on Computer science education*, pages 13–14. ACM.

[Saikkonen et al. 2001] Saikkonen, R., Malmi, L., and Korhonen, A. (2001). Fully automatic assessment of programming exercises. In *Proceedings of the 6th Annual Conference on Innovation and Technology in Computer Science Education*, ITiCSE '01, pages 133–136, New York, NY, USA. ACM.

[Saitou and Nei 1987] Saitou, N. and Nei, M. (1987). The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular biology and evolution*, 4(4):406–425.

[Sanches et al. 2011] Sanches, A., Cardoso, J. M., and Delbem, A. C. (2011). Identifying merge-beneficial software kernels for hardware implementation. In *Reconfigurable Computing and FPGAs (ReConFig), 2011 International Conference on*, pages 74–79. IEEE.

[Shackelford et al. 2006] Shackelford, R., McGettrick, A., Sloan, R., Topi, H., Davies, G., Kamali, R., Cross, J., Impagliazzo, J., LeBlanc, R., and Lunt, B. (2006). Computing curricula 2005: The overview report. *SIGCSE Bull.*, 38(1):456–457.

[Sherman et al. 2013] Sherman, M., Bassil, S., Lipman, D., Tuck, N., and Martin, F. (2013). Impact of auto-grading on an introductory computing course. *J. Comput. Sci. Coll.*, 28(6):69–75.

[Shute 2008] Shute, V. J. (2008). Focus on formative feedback. *Review of educational research*, 78(1):153–189.

[Sitthiworachart and Joy 2004] Sitthiworachart, J. and Joy, M. (2004). Effective peer assessment for learning computer programming. In *Proceedings of the 9th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, ITiCSE '04, pages 122–126, New York, NY, USA. ACM.

[Tan et al. 2005] Tan, P.-N., Steinbach, M., and Kumar, V. (2005). *Introduction to Data Mining, (First Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[Tirronen and Isomöttönen 2011] Tirronen, V. and Isomöttönen, V. (2011). Making teaching of programming learning-oriented and learner-directed. In *Proceedings of the 11th Koli Calling International Conference on Computing Education Research*, Koli Calling '11, pages 60–65, New York, NY, USA. ACM.

[Wing 2006] Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, 49(3):33–35.

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

[Zhang and Shasha 1989] Zhang, K. and Shasha, D. (1989). Simple fast algorithms for the editing distance between trees and related problems. *SIAM J. Comput.*, 18(6):1245–1262.