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

Minimizando o esforço de avaliação em disciplinas de programação introdutória utilizando agrupamentos adaptáveis

Alexandre de Andrade Barbosa

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: Metodologia e Técnicas da Computação

Evandro de Barros Costa
(Orientador)

Campina Grande, Paraíba, Brasil

B238m Barbosa, Alexandre de Andrade.
`Minimizando o esforço de avaliação em disciplinas de programação introdutória utilizando agrupamentos adaptáveis_ / Alexandre de Andrade Barbosa. - Campina Grande, 2018.
183 f. : il. color.

Tese (Doutorado em Ciência da Computação) - Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática, 2018.
"Orientação: Prof. Dr. Evandro de Barros Costa".
Referências.

1. Ensino de Programação. 2. Agrupamento de Soluções. 3. Ciência da Computação. I. Costa, Evandro de Barros. II. Título.

CDU 004.43:37.04(43)

FICHA CATALOGRÁFICA ELABORADA PELO BIBLIOTECÁRIO GUSTAVO DINIZ DO NASCIMENTO CRB - 15/515

"MINIMIZANDO O ESFORÇO DE AVALIAÇÃO EM DISCIPLINAS DE PROGRAMAÇÃO INTRODUTÓRIA UTILIZANDO AGRUPAMENTOS ADAPTÁVEIS"

ALEXANDRE DE ANDRADE BARBOSA

TESE APROVADA EM 10/09/2018

EVANDRO DE BARROS COSTA, Dr., UFAL
Orientador(a)

JORGE CESAR ABRANTES DE FIGUEIREDO, Dr., UFCG
Examinador(a)

DALTON DARIO SEREY GUERRERO, Dr., UFCG
Examinador(a)

PATRICK HENRIQUE DA SILVA BRITO, Dr., UFAL
Examinador(a)

BALDOINO FONSECA DOS SANTOS NETO, Dr., UFAL
Examinador(a)

CAMPINA GRANDE - PB

Resumo

A habilidade de programar é uma das competências fundamentais na área da computação. Esse conhecimento é basilar para a compreensão de diversos outros conceitos necessários tanto para vida acadêmica, quanto para as atividades profissionais nesta área. Contudo, apesar de tal importância, é bastante comum encontrar problemas em ambientes de ensino e aprendizagem de programação.

Muitos são os fatores descritos como relacionados com as dificuldades existentes nas disciplinas de programação. Facilmente, em tais ambientes, encontramos alunos desmotivados, com dúvidas não esclarecidas e que não conseguem entender os conceitos necessários para prática de programação.

Tradicionalmente, em disciplinas de programação são adotadas atividades práticas de codificação. Nestas atividades, a análise das soluções propostas pelos alunos nestas atividades não é simples. Muitos fatores podem ser observados na avaliação de um programa, o que torna essa avaliação demorada, sujeita ao viés e aos erros do avaliador. Dessa forma, o professor, mesmo quando auxiliado por monitores, em geral, não será consegue realizar avaliações de forma rápida, ou, para agilizar o processo não irá fornecer *feedback* detalhado.

Na presente tese de doutorado, investigou-se a possibilidade de agrupar códigos fornecidos como soluções para um problema (exercício) de programação. O agrupamento de soluções foi proposto com a intenção de possibilitar o reúso de *feedbacks* de modo a minimizar o esforço de avaliação dispendido no contexto de ensino e aprendizagem introdutórios de programação.

A investigação ocorreu através da condução de um *survey*, realizado com o objetivo de compreender os processos relacionados com a avaliação de códigos em disciplinas introdutórias de programação, um quasi experimento e um experimento, realizadas com dois objetivos: (i) investigar se é possível agrupar códigos de modo que uma avaliação seja generalizada para os elementos de um mesmo grupo (*cluster*); e (ii) verificar se a avaliação generalizada teria resultados semelhantes aos fornecidos por avaliadores humanos.

Os resultados obtidos na avaliação de abordagem sugerem que é possível reutilizar *feedbacks* de modo que o esforço de avaliação dispendido seja minimizado. Embora alguns

ajustes ainda sejam necessários, foi possível observar que a concordância obtida entre as notas geradas e as fornecidas por um avaliador variam entre uma concordância razoável, no pior caso, para uma concordância perfeita, no melhor caso. Além disso, muitos dos *feedbacks* textuais generalizados com base nos textos fornecidos pelos avaliadores foram indicados como 'parcialmente aplicáveis' ou 'totalmente aplicáveis' aos códigos avaliados.

Abstract

The ability to program is one of the fundamental competences in the field of computer science. That knowledge is a building block to the understanding of several other concepts necessary for the academic life as well for the professional activities in that area. However, in spite of such importance, it is quite common to find problems in teaching and learning environments of programming.

Many are the factors described as related to the difficulties found in programming disciplines. In such environments, we easily find unmotivated students, whose doubts are not clarified and cannot understand the concepts necessary for programming practice.

Traditionally, in programming disciplines, practical coding activities have been adopted. In these activities, the analysis of the solutions proposed by the students in these activities is not simple. Many factors can be observed on the evaluation of a program, what makes the assessment time consuming, subjected to the evaluator's bias and errors. That way, the professor, even when aided by teacher assistants, generally, will not be able to perform assessments quickly or to streamline the process will not provide detailed feedback.

In the present thesis, we investigated the possibility of clustering codes provided as solutions to a programming problem (exercise). The clustering of solutions was proposed with the intention of allowing the reuse of feedbacks in order to minimize the evaluation effort expended in the context teaching and learning of introductory programming.

The research was carried out with a survey in order to understand the processes related to the evaluation of codes in introductory programming courses, a quasi experiment and an experiment, with two objectives: (i) investigate if it is possible to cluster codes so that an evaluation is generalized to the elements of the same group cluster; and (ii) to verify whether the generalized evaluation would have results similar to those provided by human evaluators.

The results obtained in the evaluation of the suggest approach shows that it is possible to reuse feedbacks so that the evaluation effort expended is minimized. Although, some adjustments are still necessary, it was possible to observe that the concordance obtained between generated grades and those provided by an evaluator, ranged from a reasonable agreement, in the worst case, to a perfect agreement, at best. In addition, many of the generalized textual

feedbacks based on the texts provided by the evaluators were indicated as ‘partially applicable’ or ‘fully applicable’ to the evaluated codes.

Agradecimentos

Agradeço a todos que de maneira direta ou indireta contribuíram para a conclusão de mais uma fase de minha vida.

Ao grupo do Huxley, em especial Rodrigo Paes, Romero Malaquias e Márcio Ribeiro, por disponibilizar os dados utilizados nesta pesquisa.

Aos bolsistas, monitores e colaboradores que trabalharam no LAMP e que ao longo de sua passagem no laboratório contribuíram com o desenvolvimento do trabalho.

Aos ex-alunos do curso de computação da UFAL no campus Arapiraca, em especial Raniery pelas observações que realizou.

Aos professores de computação da UFAL no campus Arapiraca e dos cursos de computação da UFPB em Rio Tinto e Mamanguape, com os quais tive/tenho a alegria de trabalhar.

Aos professores da UFAL (graduação) e UFCG (mestrado e doutorado) por todo o aprendizado proporcionado.

Ao orientador Evandro Costa, por acreditar em minha competência para realizar o trabalho, pela pressão nos momentos necessários e pelo apoio nos momentos certos.

A minha irmã que tanto me incentivou no estudo de uma língua tão importante em minha área.

A Cida pelo carinho e paciência.

Aos meus pais pelo incentivo, apoio, carinho, atenção, compreensão e por outros tantos motivos.

A minha esposa e minha filha, pela compreensão e carinho nas inúmeras vezes que não pude lhes dar a atenção merecida.

Ao Luke Skywalker (meu cachorro) pela companhia em tantas madrugadas no computador.

A Deus por me dar saúde e força para superar todas as dificuldades e por todas as pessoasque colocou em meu caminho.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Códigos Fonte

Capítulo 1

Introdução

1.1 Motivação

A habilidade de programar é uma das competências fundamentais na área da computação. Esse conhecimento é basilar para a compreensão de diversos outros conceitos necessários tanto para vida acadêmica, quanto para as atividades profissionais nesta área. Os períodos iniciais dos cursos da área de computação, em geral, englobam diferentes disciplinas que possuem como foco o estudo de conceitos de algoritmos e a implementação de programas.

Apesar de tal importância, é bastante comum encontrar problemas em ambientes de ensino e aprendizagem de programação. O cenário nacional foi analisado na revisão sistemática da literatura apresentada em [55], onde a taxa média de reprovação nas disciplinas de programação é de aproximadamente 32,6%. Em outros contextos, é possível encontrar taxas de reprovação de aproximadamente 50% [15; 1; 48]. No cenário internacional diversos trabalhos, tais como [70; 42; 41], apresentam situação semelhante. Além disso, em [43], é descrito que muitos dos alunos aprovados em disciplinas de programação não possuem as competências necessárias para o decorrer do curso e consequentemente, para a vida profissional.

Neste contexto, é muito comum encontrar alunos desmotivados e que não conseguem entender os conceitos necessários para prática de programação. Muitos são os fatores descritos na literatura como relacionados ao cenário de dificuldades nas disciplinas de programação. Entre os diversos motivos apresentados na literatura, como por exemplo em [2; 46; 23; 6; 60; 64], alguns dos mais citados são a dificuldade em relação ao entendimento de conceitos

abstratos, os problemas de interpretação do que é solicitado nos enunciados dos exercícios, a falta de aplicação dos conceitos provenientes da matemática, o atraso em relação ao *feedback*, a falta de tempo e motivação para se dedicar aos estudos e as complexidades inerentes ao uso de uma linguagem de programação, tal como sua sintaxe e semântica associada.

Uma vez que o *feedback* rápido e eficiente é de extrema importância para possibilitar o aprendizado de qualquer conceito [64], acreditamos que um acompanhamento individualizado, ou mais próximo, por parte do professor em relação aos alunos, poderia minimizar ou dirimir alguns dos problemas existentes. Alguns pesquisadores defendem que a dificuldade do professor em fornecer um acompanhamento individualizado a cada um dos alunos agrava e potencializa a ocorrência dos demais fatores citados [46; 60; 64].

1.2 O problema

Ao longo de uma disciplina, um estudante pode ser submetido a diversas atividades de avaliação. Tipicamente são utilizadas avaliações diagnósticas, aquelas que visam verificar o grau de conhecimento do aluno ao longo do processo de aprendizagem; avaliações formativas, aquelas que visam identificar dificuldades de compreensão do aluno ao longo do processo de aprendizagem; e avaliações somativas, aquelas que visam medir o conhecimento de um aluno ao final do processo de aprendizagem.

Utilizar apenas avaliações somativas faria com que o real estado de assimilação dos conteúdos só pudesse ser conhecido muito tempo após o início da disciplina. Desta forma, em disciplinas de programação, é recomendável adotar avaliações diagnósticas e formativas no início e periodicamente ao longo de todo o processo de ensino e aprendizagem.

Tradicionalmente em disciplinas de programação, nas avaliações diagnósticas e formativas, são adotadas atividades práticas de codificação. Nestas atividades, o professor seleciona um conjunto de problemas (exercícios) e, para cada um destes, os alunos devem submeter códigos (programas) como solução para cada problema.

Ao longo da avaliação dos códigos propostos como solução para um problema, é possível observar soluções similares. Uma estratégia para agilizar as avaliações poderia ser cadastrar um conjunto de soluções de referência e determinar se cada solução submetida por um aluno

é equivalente a uma das soluções de referência. Contudo, a determinação da equivalência de programas é um problema indecidível, para a classe de programas que contenham *loops* [67; 47; 31], ou seja, não é possível utilizar um meio computacional para inferir se dois programas são equivalentes. Ainda assim, em um contexto de ensino e aprendizagem mesmo que uma solução computacional seja possível para determinar se dois programas são equivalentes, outros parâmetros de qualidade podem ser observados nas soluções dos discentes.

Isto posto, mesmo com o uso de soluções computacionais, é necessário proceder com a avaliação de modo a adotar algum grau de supervisão humana. Neste contexto, a análise das soluções propostas é bastante trabalhosa. Normalmente há uma grande quantidade de alunos cursando as disciplinas. Dessa forma, a quantidade de avaliações necessárias pode corresponder ao resultado da multiplicação da quantidade de alunos pela quantidade de problemas, e caso seja possível reenviar soluções, pela quantidade reenvios realizada. Associado a isso, diversos parâmetros ainda podem ser observados na avaliação de um código, além de identificar se as saídas corretas são geradas para um conjunto de entrada, podem ser observados a estrutura do código, a eficiência da solução, a legibilidade do código, a presença ou ausência de documentação (comentários), entre outros parâmetros.

Uma das estratégias utilizadas para facilitar e agilizar as atividades de avaliação de código é a divisão de trabalho. Desta forma, o trabalho de avaliação é dividido entre o professor e um grupo de monitores ou, quando possível, entre todos os professores que ministram um conjunto de disciplinas relacionadas. Nesta situação, é necessário, ou desejável, utilizar mecanismos que possibilitem um processo de avaliação consistente/igualitário, ou seja, devem ser adotados os mesmos critérios e o mesmo rigor para todas as avaliações.

Contudo, mesmo quando auxiliado por monitores, o professor muitas vezes não consegue realizar avaliações adequadas de forma rápida, pois caso busque fornecer uma avaliação rápida, provavelmente ela carecerá de qualidade, por outro lado, ao tentar prover *feedback* de alta qualidade, ocorrerá uma sobrecarga de trabalho e demora na resposta.

Outra estratégia amplamente utilizada para facilitar e agilizar as atividades de avaliação de código é o uso de juízes online. Um juiz online consiste em uma aplicação que indica se um código é correto, ou não, com base em um conjunto de casos de teste [49]. Um caso de teste corresponde a especificação de conjunto de entradas e as respectivas saídas esperadas [20]. Dessa forma, se um código passa pelo conjunto de teste, ele é aceito como uma solução

correta, em outro caso o código é dito incorreto.

Essa abordagem, contudo, não avalia critérios de qualidade em um código, ou seja, se uma solução fornece o conjunto de saídas esperado, ela é considerada totalmente correta. Para ilustrar a limitação existente na abordagem citada, dado o enunciado de um problema de programação "Forneça todos os números pares no intervalo [1;10]", e observados os Códigos exibidos em 1.1, 1.2 e 1.3 como soluções propostas por discentes.

Código Fonte 1.1: ']Solução 1 para o problema 'Todos os pares no intervalo [1;10]'

```
1 print(2)
2 print(4)
3 print(6)
4 print(8)
5 print(10)
```

Código Fonte 1.2: ']Solução 2 para o problema 'Todos os pares no intervalo [1;10]'

```
1 for i in range(2,11,2):
2     print(i)
```

Código Fonte 1.3: ']Solução 3 para o problema 'Todos os pares no intervalo [1;10]'

```
1 for i in range(1,11):
2     if (i % 2 == 0) :
3         print(i)
```

As soluções exibidas em 1.1, 1.2 e 1.3 seriam avaliadas como totalmente corretas por um juiz online. Contudo, dependendo dos critérios de um avaliador, estas soluções poderiam ser avaliadas de modo diferente.

Observadas as descrições anteriores, é possível afirmar que o acompanhamento individualizado é inviável no cenário descrito. Sendo a mencionada avaliação muito trabalhosa, ela é demorada e sujeita ao viés e aos erros de cada avaliador, caracterizando então um problema. Isso motiva a indagação 'como prover *feedback* individualizado e em tempo hábil para os alunos, sem sobrecarregar o docente?'

1.3 Objetivos

Muitas pesquisas têm sido desenvolvidas com o intuito de propor métodos, abordagens ou ferramentas para facilitar o acompanhamento das atividades em disciplinas de programação. Algumas destas pesquisas, tais como [46; 50; 59; 54; 21], propuseram o uso de mecanismos automáticos ou semi automáticos para análise de códigos.

Nesta pesquisa de doutorado, é objetivo geral investigar meios computacionais para reutilizar *feedbacks* fornecidos por avaliadores, reduzindo a quantidade de avaliações necessárias de modo que o esforço dispendido no processo de avaliação em disciplinas de programação introdutória seja minimizado.

Neste ponto é necessário descrever o significado utilizado para os termos 'avaliação', 'minimização do esforço', 'programação introdutória' e 'agrupamento adaptável'.

Compreende-se 'avaliação' como o processo de observar a solução e fornecer *feedback* ao discente. Este *feedback* pode ser dividido em uma classificação, tipicamente um critério (ex. A, B, C, D) ou nota (ex. valores entre 0 e 10) para a solução apresentada, e uma informação textual, podendo ocorrer variação de professor para professor, correspondendo, por exemplo, a um conjunto de observações para melhoria do código ou às justificativas para o critério ou nota fornecido.

Esforço pode ser interpretado de diferentes maneiras, pode-se entender esforço como a quantidade de força necessária para realização de uma atividade ou como o tempo dispendido na execução de um processo mental. Neste trabalho, a redução de esforço está relacionada a quantidade de avaliações necessárias para prover *feedback* para todos os discentes, através da observação dos códigos submetidos. Deste forma, a 'minimização do esforço' pode ser medida pela quantidade de avaliações que foram realizadas em comparação a quantidade total existente. Ao descrever 'minimização do esforço' entende-se que ocorre uma redução da quantidade de observações, não necessariamente correspondendo ao esforço minimo necessário para a atividade.

O contexto investigado nesta pesquisa está relacionado com disciplinas de programação introdutória. Nesta tese, compreende-se como 'programação introdutória' um ambiente de ensino e aprendizagem onde os discentes terão o primeiro contato com os conceitos de algoritmos adotando uma linguagem de programação estruturada. Tal ambiente pode ser presen-

cial ou à distância, sendo necessário que os códigos propostos como solução para exercícios possam ser fornecidos como entrada para um sistema computacional que implemente a abordagem.

Sendo assim, investigou-se à possibilidade de agrupar as soluções em código fornecidas para um problema com base em sua similaridade, de modo que as soluções muito próximas pudessem ter sua avaliação generalizada, minimizando assim o total de avaliações necessárias para julgar todo o conjunto de códigos submetido.

O agrupamento de códigos proposto como meio para minimizar o esforço de avaliação, explora a similaridade com base em características extraídas dos próprios códigos. Estas características correspondem a um conjunto de métricas descritas na área de Engenharia de Software.

Um agrupamento é um meio de organizar elementos de forma que estes possam pertencer a um grupo, de modo que cada grupo pode conter diversos elementos. Compreende-se então um 'agrupamento adaptável' como a possibilidade de variar o modo como se determina que um elemento seja inserido em um grupo.

Para atingir o objetivo geral descrito, um conjunto de objetivos específicos se apresentou como necessário na condução da pesquisa. Dessa forma, são então objetivos específicos relacionados com esta tese:

- identificar quais critérios são considerados por cada avaliador;
- investigar se há variação na avaliação entre avaliadores, ou seja, diferentes avaliadores podem variar seus critérios, ou seu rigor, e fornecer *feedbacks* distintos;
- explorar se é possível agrupar códigos de modo que um *feedback* objetivo (classificação/nota) seja generalizado;
- explorar se é possível agrupar códigos de modo que um *feedback* subjetivo (informação textual) seja generalizado;
- avaliar o grau de concordância de um avaliador em relação aos seus pares;
- avaliar o grau de concordância de cada avaliador em relação à abordagem proposta.

Consideramos necessário identificar o conjunto de critérios considerado por um avaliador, de modo que seja possível observar se as abordagens e ferramentas existentes são

suficientes. Além disso, essa identificação é imprencídivel para investigar se há variação na avaliação entre avaliadores.

Visto que a abordagem proposta foca no reuso de *feedbacks*, é necessário explorar se é possível agrupar códigos de modo que os *feedbacks* sejam generalizados e indagar junto aos avaliadores se estes concordam com o que foi determinado como *feedback* para um dado código. Isto posto, se mostra relevante idenfiticar se o grau de concordância de cada avaliador com os *feebacks* generalizados é adequado quando comprado com o grau de concordância obtido entre dois avaliadores humanos.

Na investigação conduzida, observou-se que concordância obtida entre notas geradas e as fornecidas por um avaliador variam de uma uma concordância razoável, no pior caso analisado, para uma concordância perfeita, no melhor caso analisado. Diferentes conjuntos de métricas foram identificados para cada avaliador em cada problema avaliado, ou seja observou-se variação dos critérios de avaliação adotados para diferentes avaliadores. A maioria dos *feedbacks* textuais gerados foi classificado como 'PARCIALMENTE aplicável' ou 'TOTALMENTE aplicável', ocorrendo diferentes situações em que tais clasificações foram fornecidas.

1.4 Estrutura do trabalho

O restante desta tese de doutorado está organizado da seguinte forma:

Capítulo 2 - a terminologia adotada e os conceitos necessários à compreensão da pesquisa são apresentados;

Capítulo 3 - os trabalhos relacionados ao tema pesquisado são descritos, buscando-se caracterizar onde esta tese se insere, bem como descrever quais os principais diferenciais abordados em relação a outras propostas;

Capítulo 4 - a proposta e os detalhes da condução das investigações realizadas são exibidas, neste contexto, são apresentados o projeto dos experimentos, os resultados, a análise dos resultados e a discussão relativa a estes;

Capítulo 5 - as contribuições originadas da pesquisa, as conclusões, e os trabalhos futuros planejados para dar continuidade à pesquisa são descritos.

Além dos capítulos citados, os seguintes apêndices também podem ser consultados:

Apêndice A - o questionário utilizado em uma das etapas desta pesquisa é exibido;

Apêndice B - os enunciados e as descrições de entrada e saída esperadas para cada um dos problemas analisados no quasi experimento são exibidos;

Apêndice C - os resultados obtidos separados por problema, para cada um dos problemas analisados no quasi experimento são exibidos;

Apêndice D - os enunciados e as descrições de entrada e saída esperadas para cada um dos problemas analisados no experimento são exibidos;

Apêndice E - os resultados obtidos separados por problema, para cada um dos problemas analisados no experimento são exibidos.

Capítulo 2

Fundamentação Teórica

Neste capítulo, são abordados os conceitos necessários para uma melhor compreensão deste trabalho. Tais conceitos são apresentados divididos em três seções, sendo elas: métricas de software, algoritmos de agrupamento e medidas estatísticas e de comparação. Para cada uma das seções, tem-se como objetivo descrever os conceitos utilizados e estabelecer a terminologia que foi adotada neste trabalho.

2.1 Medidas, métricas e medições na Engenharia de Software

Diversas métricas foram propostas na Engenharia de Software. Há muito debate sobre para quais aplicações uma medida pode ser utilizada e sobre quais medidas utilizar para atingir um determinado objetivo [53; 62].

Os termos medida, métrica e medição, por vezes, são empregados indistintamente na área, contudo existem algumas diferenças entre eles [53]. Os termos medida, métrica e medição são definidos em [53] conforme as descrições exibidas a seguir:

Definição 2.1.1. Medida: fornece uma indicação quantitativa de uma extensão, quantidade, dimensão, capacidade ou tamanho de algum atributo de um produto, ou de um processo [53].

Definição 2.1.2. Medição: ato de determinação de uma medida [53].

Definição 2.1.3. Métrica: função padrão utilizada na obtenção de medidas de produto ou de processo [53].

Nesta tese, os termos métrica e medição, serão utilizados com base nas definições apresentadas em [53]. O termo medida será utilizado com uma pequena variação em relação a definição apresentada em [53], sendo este definido como a seguir:

Definição 2.1.4. Medida: corresponde a um valor, ou conjunto de valores, obtido pela aplicação de uma métrica, podendo este valor representar uma quantidade, dimensão, capacidade ou outro atributo de um produto, ou de um processo.

Ao serem observadas as definições anteriores, podemos descrever que realizamos medições adotando métricas para obtenção de medidas em relação ao software. No contexto de ensino e aprendizagem introdutório em programação, o software se resume a um único código fonte.

Métricas podem ser utilizadas como indicadores de produto e de processo. Em geral, as métricas servem para auxiliar processos de desenvolvimento de software, sendo utilizadas para estimar custo, planejar cronograma, especificar escopo, entre outros fins.

Uma medida de software representa um meio de verificar se um software possui determinada propriedade. Uma medida pode ser categorizada de diferentes formas, podendo ser dividida em duas categorias: medidas diretas e indiretas. As medidas diretas (quantitativas) são aquelas que representam uma quantidade observada, tais como custo, esforço, número de linhas de código, tempo de execução e número de defeitos. Medidas indiretas (qualitativas) são aquelas que exigem análise e estão relacionadas com a funcionalidade, qualidade, complexidade e manutenibilidade. Outras classificações são possíveis, estas contudo fogem ao escopo deste trabalho.

Uma vez que o foco deste trabalho é o contexto de ensino e aprendizagem introdutório em programação, não há interesse no processo de desenvolvimento adotado, então são utilizadas medidas extraídas apenas dos códigos submetidos como soluções de exercícios.

As métricas adotadas foram selecionadas por serem aquelas mais comumente descritas na literatura, de modo que estas fossem úteis para diferenciar características dos códigos que pudessem ser relacionadas aos critérios descritos pelos avaliadores. Outras métricas são disponibilizadas nos meios utilizados para extração das medidas de software, métricas relacionadas por exemplo, com a manutenabilidade do código, ou a propenção que um código possui a apresentar *bugs*, entre outras. Julgamos que tais métricas são relacionadas com atributos desejáveis em contextos mais avançados, tal como Engenharia de Software, e não no

contexto de ensino e aprendizagem de programação. Sendo assim, tais métricas não foram utilizadas e não serão apresentadas neste capítulo.

Algumas medidas são geradas considerando exclusivamente o código (ex. quantidade de operadores), enquanto outras representam uma relação existente entre dois códigos (ex. similaridade de *tokens*). As seções seguintes contêm a descrição do conjunto de medidas disponível, até o momento, para uso na presente pesquisa.

2.1.1 Métricas de corretude

Duas métricas relacionadas com o grau de corretude de uma solução estão disponíveis. São elas:

- Corretude sintática - representa se um código possui alguma construção não permitida pelas regras gramaticais da linguagem de programação. Um valor *booleano* é utilizado para indicar a presença ou ausência de erros de sintaxe identificados em um programa;
- Corretude funcional - representa o grau de aderência de um código com os requisitos de um problema. Os requisitos são representados através de um conjunto de testes [20] que contém uma quantidade variável de casos de teste. Um valor numérico no intervalo $[0, 100]$ é utilizado para indicar a corretude funcional do código.

A métrica 'corretude sintática' é importante pois diferencia as soluções que não podem ser executadas daquelas que podem. Desta forma, tal métrica pode ser útil ao realizar os agrupamentos dos códigos, separando códigos de acordo com a descrição anterior.

A métrica 'corretude funcional' é importante pois diferencia as soluções que atingiram um nível de sucesso em relação ao que é esperado pela execução dos códigos ao observar as respectivas saídas. Desta forma, tal métrica pode ser útil ao realizar aproximar os códigos que tenham um nível de sucesso similar.

Um exemplo ilustrativo é exibido utilizando os Códigos 2.1, 2.2, 2.3, 2.4 e 2.5, teríamos os valores de corretude sintática e corretude funcional exibidos na Tabela 2.1.

Código Fonte 2.1: Solução 1 para o problema 'Todos os pares no intervalo'

```
1 print(2)
2 print(4)
```

Tabela 2.1: Medidas de corretude sintática e corretude funcional para os Códigos 2.1, 2.2, 2.3, 2.4 e 2.5

	Códigos				
Métricas	2.1	2.2	2.3	2.4	2.5
Corretude sintática	True	True	True	False	True
Corretude funcional	100%	100%	100%	-	80%

```
3 print(6)
4 print(8)
5 print(10)
```

Código Fonte 2.2: Solução 2 para o problema 'Todos os pares no intervalo'

```
1 for i in range(2,11,2):
2     print(i)
```

Código Fonte 2.3: Solução 3 para o problema 'Todos os pares no intervalo'

```
1 for i in range(1,11):
2     if (i % 2 == 0) :
3         print(i)
```

Código Fonte 2.4: Solução 4 para o problema 'Todos os pares no intervalo'

```
1 for i in range(1,11,2)
2     if (i % 2 = 0)
3         print i
```

Código Fonte 2.5: Solução 5 para o problema 'Todos os pares no intervalo'

```
1 for i in range(1,10):
2     if (i % 2 == 0) :
3         print(i)
```

Os Códigos 2.1, 2.2 e 2.3 possuem todas as construções válidas de acordo com a sintaxe da linguagem Python, os valores obtidos na saída correspondem a todos os valores esperados. O Código 2.4 possui construções inválidas de acordo com a sintaxe da linguagem Python

3, assim não é possível avaliar computacionalmente a corretude funcional do código. A mensagem exibida no código 2.6 é apresentada ao tentar interpretar o Código 2.4, apenas o primeiro erro de sintaxe é exibido, apesar de existirem outros erros de sintaxe no código. O Código 2.5 possui todas as construções válidas de acordo com a sintaxe da linguagem Python 3, os valores obtidos na saída correspondem a parte dos valores esperados. Uma vez que o valor '10' está ausente da saída correspondente a execução do Código 2.5, o valor de 80% é obtido para corretude funcional.

Código Fonte 2.6: Mensagem de erro de sintaxe exibida para a solução 4 para o problema 'Todos os pares no intervalo'

```
1 File "code-introducao-04.py", line 1
2     for i in range(1,11)
3                         ^
4 SyntaxError: invalid syntax
```

2.1.2 Métrica de complexidade

Uma única métrica de complexidade foi adotada na Códigos 2.6 deste trabalho. A complexidade ciclomática, ou complexidade de McCabe [40], corresponde ao número de caminhos linearmente independentes de um código, onde cada estrutura de controle (ex. condicionais e repetições) utilizada aumenta a quantidade de fluxos alternativos. A complexidade ciclomática pode ser computada através da geração do grafo de fluxo de controle (GFC) do código (função, método, classe). Um GFC contém nós e arestas direcionadas. Os nós correspondem a grupos indivisíveis de comandos. As arestas conectam dois nós se o segundo puder ser executado imediatamente após o primeiro. A quantidade de áreas existente no GFC corresponde a complexidade ciclomática. A Figura 2.1 exibe o GFC correspondente ao Código 2.3, onde cada nó exibido corresponde a linha de código com a mesma numeração.

A métrica 'complexidade ciclomática' é importante pois diferencia as soluções de acordo com a quantidade de desvios existentes em um código. Desta forma, por exemplo, as soluções podem ser agrupadas de acordo com a quantidade de desvios que estas possuem. As medidas obtidas podem diferenciar códigos que não utilizem desvios e outros que utilizam. Além disso, é possível agrupar soluções que tenham utilizado algumas construções de forma exagerada (ex. existência de muitas condições aninhadas) ou muito pobre (ex. ausência de

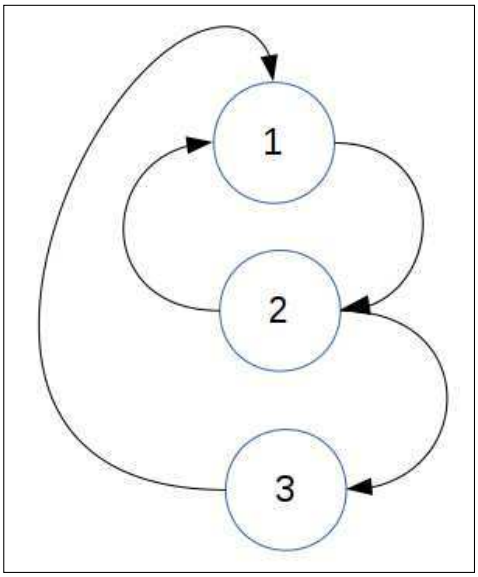


Figura 2.1: Grafo de fluxo de controle para o Código 2.3, a complexidade ciclomática corresponde a três, pois existem três áreas distintas no grafo.

repetições).

Um exemplo ilustrativo é exibido utilizando os Códigos 2.1, 2.2, 2.3, 2.4 e 2.5 para ilustrar as medidas obtidas pela aplicação da métrica que computa complexidade ciclomática, teríamos os valores exibidos na Tabela 2.2.

Tabela 2.2: Medidas de complexidade ciclomática para os códigos 2.1, 2.2, 2.3, 2.4 e 2.5

	Códigos				
Métrica	2.1	2.2	2.3	2.4	2.5
Complexidade ciclomática	1	2	3	-	3

A complexidade ciclomática de um código que não possui decisões ou repetições, tal como o Código 2.1, é um, pois existe somente um caminho possível de execução. Nesta caso, o GFC corresponderia a um único nó sem arestas. Para um código que possui somente uma estrutura de decisão ou repetição, tal como o Código 2.2, existem dois caminhos possíveis, aquele quando a condição é avaliada em verdadeiro ou a repetição é executada, e aquele quando a condição é avaliada em falso ou a repetição não é executada. Nos códigos 2.3 e 2.5, temos complexidade ciclomática igual a três, pois tem-se uma instrução de repetição e uma

instrução de decisão, criando assim um caminho alternativo em cada uma destas instruções. A complexidade ciclomática do Código 2.4 não pode ser computada, pois existem erros de sintaxe presentes no código.

2.1.3 Métricas de Halstead

As métricas de Halstead visam identificar propriedades mensuráveis no código, bem como relações entre estas propriedades [32]. As medidas de Halstead são computadas considerando-se o programa como uma coleção de *tokens*, classificados em dois tipos: operadores; ou operandos. A partir destes são computadas as seguintes medidas:

- Número de operadores distintos ($n1$) - computa a quantidade de operadores (ex. operadores matemáticos) não considerando repetições;
- Número de operandos distintos ($n2$) - computa a quantidade de operandos (ex. variáveis) não considerando repetições;
- Número total de operadores ($N1$) - computa a quantidade de operadores (ex. operadores matemáticos) considerando repetições;
- Número total de operandos ($N2$) - computa a quantidade de operandos (ex. variáveis) considerando repetições;
- Vocabulário ($n = n1+n2$) - corresponde à soma dos operadores e operandos distintos;
- Tamanho ($N = N1 + N2$) - corresponde à soma dos operadores e operandos totais;
- Volume ($V = N \times \log_2 n$) - corresponde à quantidade de informação no programa, proporcional ao tamanho necessário para armazenar o código;

O subconjunto das métricas de Halstead foi selecionado devido a sua utilidade em diferenciar as soluções de acordo com a quantidade de declarações de variáveis, atribuições e realização de cálculos diversos. Desta forma, por exemplo, as soluções podem ser agrupadas de acordo com a quantidade de variáveis que foram declaradas. As medidas obtidas com o uso destas métricas podem diferenciar soluções que tenham utilizado, por exemplo, variáveis de forma exagerada ou que tenham realizado cálculos com a adoção de muitos operadores.

Um exemplo ilustrativo é exibido utilizando os Códigos 2.1, 2.2, 2.3, 2.4 e 2.5 para ilustrar as medidas obtidas pela aplicação das métricas de Halstead, teríamos os valores exibidos na Tabela 2.3.

Tabela 2.3: Medidas relacionadas às métricas de Halstead para os códigos 2.1, 2.2, 2.3, 2.4 e 2.5

	Códigos				
Métricas	2.1	2.2	2.3	2.4	2.5
Número de operadores distintos ($n1$)	0	0	2	-	2
Número de operandos distintos ($n2$)	0	0	4	-	4
Número total de operadores ($N1$)	0	0	2	-	2
Número total de operandos ($N2$)	0	0	4	-	4
Vocabulário ($n = n1 + n2$)	0	0	6	-	6
Tamanho ($N = N1 + N2$)	0	0	6	-	6
Volume ($V = N \times \log_2 n$)	0	0	15.50	-	15.50
Dificuldade ($D = \frac{n1}{2} * \frac{N2}{n2}$)	0	0	1.0	-	1.0
Esforço ($E = V * D$)	0	0	15.50	-	15.50
Tempo ($T = \frac{E}{18}$)	0	0	0.86	-	0.86

De maneira simplificada, as métricas de Halstead consideram operadores, chamadas de funções, operadores matemáticos, lógicos e relacionais, e consideram operandos, identificadores de variáveis, constantes e valores obtidos/retornados pela aplicação dos operadores. Não são considerados para o cálculo das métricas funções que sejam palavras reservadas, bem como seus parâmetros/argumentos, estruturas de controle, sejam de decisão ou repetição, caso estes não contenham operações em seus conteúdos.

O Código 2.1 possui apenas chamadas para a função $print$, que é ignorada pois se trata de uma palavra reservada e não contém operações em nenhum dos parâmetros. Desta forma, todas as medidas obtidas correspondem a 0 (zero). De modo similar, o Código 2.2 possui apenas a estrutura de controle for e chamadas para as funções $range$ e $print$. Desta forma, todas as medidas obtidas também correspondem a 0 (zero). Para o Código 2.4, não é possível computar as métricas, pois o código possui erros de sintaxe.

Para os códigos 2.3 e 2.5, a computação das métricas de Halstead ocorre de modo similar, pois a única diferença nestes código é um parâmetro passado para função $range$, que não é considerada no cálculo da métrica. Desta forma, são considerados operadores $\%$ e $==$, correspondendo ao valor 2 (dois) obtido para o número de operadores distintos e o número total de operadores. Ainda para os códigos 2.3 e 2.5, são considerados operandos a variável i, as constantes 2 e 0 e o resultado obtido na operação $i\%2$.

2.1.4 Métricas de contagem de linhas

As métricas de contagem de linhas visam medir o tamanho de um programa, em geral, com intuito de estimar produtividade ou esforço. Diversas métricas de contagem de linhas são descritas na literatura, o conjunto de diferentes métricas se justifica, pois a forma como um código foi escrito influencia no valor de medida obtido.

São métricas de contagem de linhas:

- LOC (*lines of code*) - número total de linhas;
- LLOC (*logical lines of code*) - número total de linhas de código com operações lógicas;
- SLOC (*source lines of code*) - número total de linhas de código;
- Linhas de comentário - número total de linhas de comentário;
- *String* de múltiplas linhas - número total de linhas em uma *string* de múltiplas linhas;
- Linhas em branco - número total de linhas em branco.

As métricas de contagem de linhas auxiliam em diferenciar as soluções de acordo com o tamanho total do código gerado. Os códigos serão agrupados de acordo com as quantidades de linhas para cada um das respectivas propriedades (ex. linhas de código ou linhas em branco). Códigos que possuem uma quatidade de linhas podem indicar que o discente não está explorando determinadas construções (ex. repetições).

Um exemplo ilustrativo é exibido utilizando os Códigos 2.1, 2.2, 2.3, 2.4 e 2.5 para ilustrar as medidas obtidas pela aplicação da métrica de contagem de linhas, teríamos os valores exibidos na Tabela 2.4.

Tabela 2.4: Medidas relacionadas com a contagem de linhas para os códigos 2.1, 2.2, 2.3, 2.4 e 2.5

	Códigos				
Métricas	2.1	2.2	2.3	2.4	2.5
LOC	5	2	3	3	3
LLOC	5	2	3	-	3
SLOC	5	2	3	-	3
Linhas de comentário	0	0	0	0	0
Linhas em branco	0	0	0	0	0

Considerando as regras de construção de códigos para linguagem Python, onde a indentação é determinante para o modo como o código será interpretado, e somente um código corretamente identado é considerado sintaticamente correto. As variações de resultados obtidos como medidas na aplicação das métricas de contagem de código não é significativa, ou seja, em geral as métricas expressam as mesmas medidas.

Desta forma, para o Código 2.4, não é possível computar todas as métricas pois o código possui erros de sintaxe. Para os outros códigos, a relação das medidas com a quantidade de linhas apresentada é facilmente observada. Nenhum dos códigos possui linhas de comentário ou linhas em branco, assim as respectivas métricas possuem medidas iguais a 0 (zero). As medidas apresentadas para as métricas *LOC*, *LLOC* e *SLOC* não possuem variações.

2.1.5 Métricas de similaridade

Na literatura, podem ser encontradas diversas técnicas para avaliação de similaridade. Alguns trabalhos, tais como [11] e [44], utilizam tais meios para realizar a detecção de plágio, enquanto que outros trabalhos, como [21] e [39], estes meios são usados para descrever o grau de similaridade de duas entidades.

Nesta pesquisa, foram usadas três estratégias de avaliação de similaridade: Coeficiente Jaccard [65], distância de edição de texto (*Text Edition Distance*) [36; 38] e distância de edição de árvore (*Tree Edition Distance*) [73]. Todas têm como base a sintaxe de *tokens*

extraídas da *Abstract Syntax Tree of a Python code*[1], que é a representação em árvore da estrutura sintática de um algoritmo escrito em *Python*.

O coeficiente de Jaccard é uma técnica utilizada para calcular a semelhança entre conjuntos. Dados dois conjuntos, A e B, a divisão do tamanho da intersecção de A e B pelo tamanho da união de A e B gera um valor que representa a similaridade existente entre os conjuntos. Tal descrição é representada na equação 2.1.

$$Coeficiente_{Jaccard}(A, B) = \frac{|A \bigcap B|}{|A \bigcup B|} \tag{2.1}$$

Para o cálculo do coeficiente de Jaccard para os códigos, os conjuntos A e B correspondem aos conjuntos de *tokens* gerados a partir dos dois programas comparados.

Utilizando os códigos 2.1, 2.2, 2.3, 2.4 e 2.5 para ilustrar as medidas obtidas pela aplicação da métrica de similaridade baseada no coeficiente de Jaccard, teríamos os valores exibidos na Tabela 2.5.

Tabela 2.5: Similaridade baseada no coeficiente de Jaccard para os códigos 2.1, 2.2, 2.3, 2.4 e 2.5

Código	2.1	2.2	2.3	2.4	2.5
2.1	1.00	0.31	0.31	–	0.31
2.2	0.31	1.00	0.54	–	0.54
2.3	0.31	0.54	1.00	–	0.96
2.4	–	–	–	–	–
2.5	0.31	0.54	0.96	–	1.00

A similaridade baseada na distância de edição de texto [36] é uma medida de dissimilaridade entre *strings*. Para sua determinação, é computada a quantidade mínima de operações (inserção, deleção ou substituição) para transformar uma *string* em outra. Diferentes técnicas podem ser adotadas na implementação desta medida de dissimilaridade, comumente a distância de Levenshtein [38] é empregada.

Para o domínio de programação, a abordagem baseia-se na realização das operações sobre os *tokens* de um determinado código. Desta forma, dados dois códigos, A e B, a distância

[1] https://docs.python.org/2/library/ast.html

de Levenshtein entre A e B fornece a quantidade mínima de operações para que os *tokens* de A e B sejam iguais. Os valores obtidos são sensíveis aos tamanhos dos códigos comparados, sendo assim, é necessário realizar uma normalização. A normalização é efetuada calculando-se o valor máximo de operações necessárias para códigos totalmente distintos. A Equação 2.2 exibe o cálculo da *Text Edition Distance*, onde $levenshtein(A, B)$ corresponde distância de Levenshtein entre A e B, e $max(|A|, |B|)$ corresponde ao número máximo de operações possíveis entre A e B. Uma vez que é desejada a similaridade entre A e B, o valor de dissimilaridade obtido é subtraído do valor máximo de similaridade.

$$Distance_{Text}(A, B) = 1 - levenshtein(A, B)/max(|A|, |B|) \tag{2.2}$$

Utilizando os códigos 2.1, 2.2, 2.3, 2.4 e 2.5 para ilustrar as medidas obtidas pela aplicação da métrica de similaridade baseada na distância de edição de texto, teríamos os valores exibidos na Tabela 2.6.

Tabela 2.6: Similaridade baseada na distância de edição de texto para os códigos 2.1, 2.2, 2.3, 2.4 e 2.5

Código	2.1	2.2	2.3	2.4	2.5
2.1	1.00	0.23	0.25	–	0.25
2.2	0.23	1.00	0.56	–	0.54
2.3	0.25	0.56	1.00	–	0.98
2.4	–	–	–	–	–
2.5	0.25	0.54	0.98	–	1.00

A similaridade baseada na distância de edição de árvore [73] é uma medida de dissimilaridade de estruturas de árvores. O valor obtido com esta medida corresponde à quantidade mínima de operações (inserção, deleção ou substituição) para transformar uma árvore em outra. Neste caso, as operações são realizadas sobre os nós de uma árvore.

Para o domínio de programação, as árvores geradas correspondem às árvores sintáticas dos programas comparados. Desta forma, dados dois códigos, A e B, a distância entre A e B fornece a quantidade mínima de operações para que as árvores de A e B sejam iguais. Os valores obtidos são sensíveis aos tamanhos dos códigos comparados, sendo assim, é neces-

sário realizar uma normalização. A normalização é efetuada calculando-se o valor máximo de operações necessárias para códigos totalmente distintos. A Equação 2.3 exibe o cálculo da *Tree Edition Distance*, onde $distance(A, B)$ corresponde quantidade de operações para transformação das árvores de A e B, e $max(|A|, |B|)$ corresponde ao número máximo de operações possíveis entre A e B. Uma vez que é desejada a similaridade entre A e B, o valor de dissimilaridade obtido é subtraído do valor máximo de similaridade.

$$Distance_{Tree}(A, B) = 1 - distance(A, B)/max(|A|, |B|) \tag{2.3}$$

Utilizando os códigos 2.1, 2.2, 2.3, 2.4 e 2.5 para ilustrar as medidas obtidas pela aplicação da métrica de similaridade baseada na distância de edição de árvore, teríamos os valores exibidos na Tabela 2.7.

Tabela 2.7: Similaridade baseada na distância de edição de árvore para os códigos 2.1, 2.2, 2.3, 2.4 e 2.5

Código	2.1	2.2	2.3	2.4	2.5
2.1	1.00	0.41	0.56	–	0.56
2.2	0.41	1.00	0.82	–	0.80
2.3	0.56	0.82	1.00	–	0.98
2.4	–	–	–	–	–
2.5	0.54	0.80	0.98	–	1.00

2.2 O algoritmo de agrupamento *Kmeans*

Algoritmos de agrupamento organizam grupos de elementos levando em conta sua similaridade. Desta forma, é correto afirmar que os elementos de um mesmo grupo são mais similares entre si do que em relação aos elementos de outros grupos. Diferentes algoritmos de agrupamento podem ser encontrados na literatura. O algoritmo *Kmeans* foi selecionado por ser um dos mais citados na literatura.

A Aprendizagem de Máquina (AM), um ramo da Inteligência Artificial (IA), possui como objetivo o desenvolvimento de meios computacionais que possibilitem a criação de sistemas

com capacidade de aprendizado [25]. As técnicas de AM são, em geral, classificadas em duas categorias, sendo estas o aprendizado supervisionado e o aprendizado não-supervisionado. Nas técnicas de aprendizado supervisionado, os algoritmos são treinados a partir de classificações fornecidas por um especialista. Após treinamento, o algoritmo irá generalizar novos exemplos de acordo com o que aprendeu [25]. Nas técnicas de aprendizado não supervisionado, os algoritmos geram grupos (*clusters*) baseando-se em características dos objetos que serão classificados [25].

Kmeans é um algoritmo de aprendizado não supervisionado. O funcionamento do algoritmo se baseia em centros de gravidade (centróides), ajustados a cada iteração, os quais são utilizados para definir os elementos pertencentes aos grupos. Inicialmente, é necessário definir a quantidade de centróides (k), opcionalmente podem ser fornecidos os valores iniciais dos centróides; caso não fornecidos estes são selecionados aleatoriamente, e a função de cálculo da distância que será utilizada, tipicamente a distância euclidiana é adotada. A cada iteração do algoritmo, os centróides são recalculados e é realizada uma nova associação dos elementos aos grupos, novas iterações ocorrem até que os centróides não sejam alterados. Na Figura 2.2, exibida em [51], o funcionamento do algoritmo é ilustrado, nesta figura os elementos são pontos e os centróides são 'x'.

A Figura 2.3 exibe a execução do algoritmo *kmeans* sobre os Códigos 2.1, 2.2, 2.3, 2.4 e 2.5, adotando $k = 3$ e duas métricas em cada vetor de propriedades. Os centróides são representados pelo símbolo $\times$, os elementos (códigos) são representados por círculos e as cores representam os grupos aos quais um elemento pertence. O Código 2.4 não é utilizado em nenhuma execução, uma vez que os erros de sintaxe impedem a extração de métricas.

Na Figura 2.3a, são utilizadas as métricas corretude funcional e complexidade ciclomática. O grupo representado pela cor azul possui um único elemento, o Código 2.5, o grupo representado pela cor vermelha possui os Códigos 2.2 e 2.3, finalmente, o grupo representado pela cor verde possui um único elemento, o Código 2.1.

Na Figura 2.3b, são utilizadas as métricas número de operadores distintos e número de operandos distintos. Nesta execução apenas dois grupos foram criados, um grupo representado pela cor azul e outro pela cor vermelha. Os códigos 2.3 e 2.5 pertencem ao grupo azul, e os Códigos 2.1 e 2.2 pertencem ao grupo vermelho.

Na Figura 2.3c, são utilizadas as métricas linhas contendo código e similaridade baseada

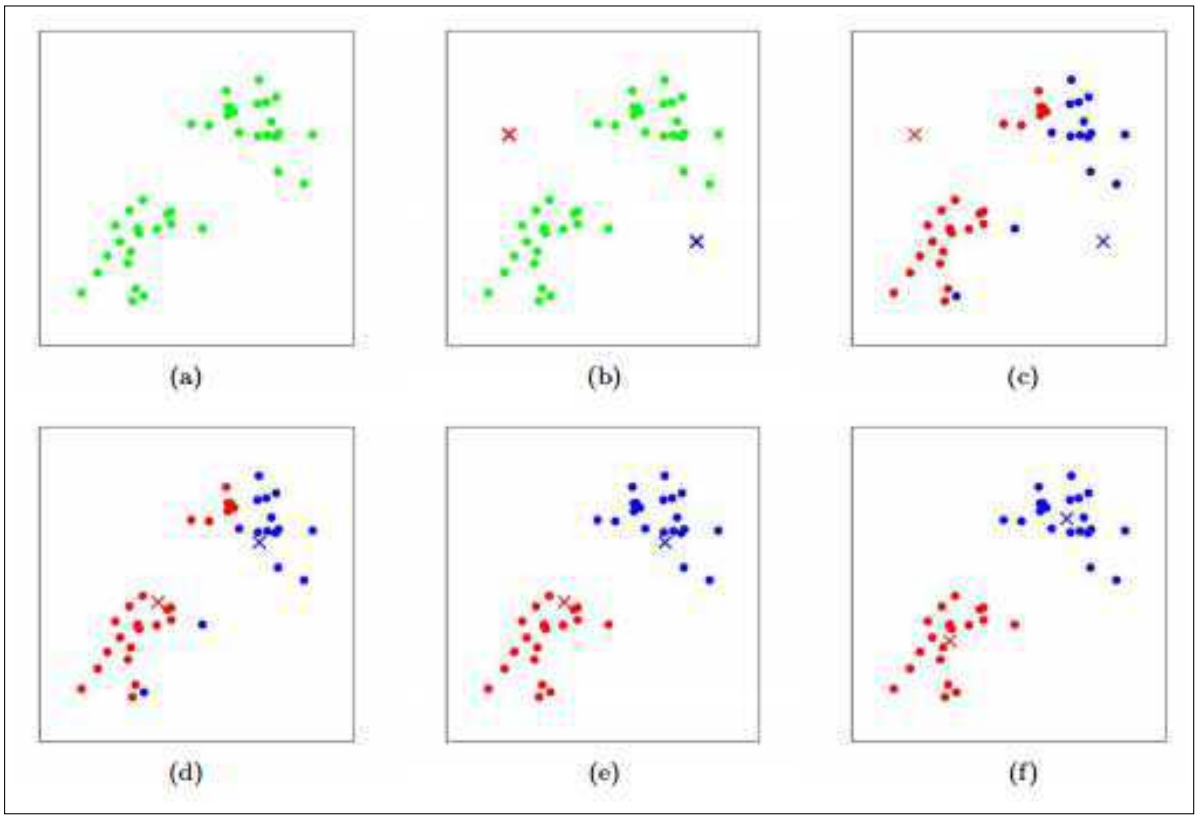


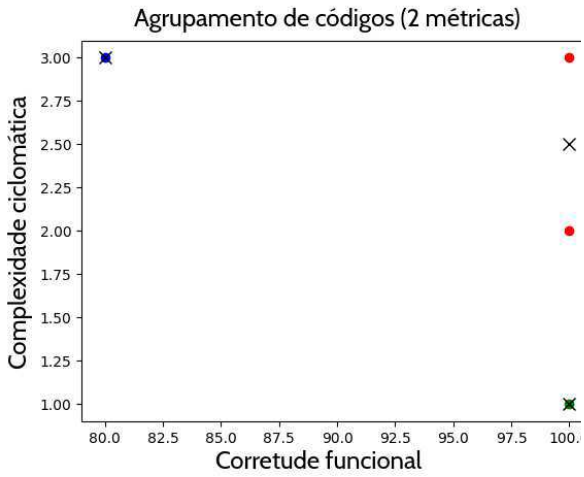
Figura 2.2: Execução do algoritmo *kmeans*. (a) conjunto de dados original. (b) centróides escolhidos aleatoriamente. (c-f) iterações do algoritmo, onde novos centróides são selecionados e os elementos são atribuídos para novos grupos representados em diferentes cores.

Fonte: [51]

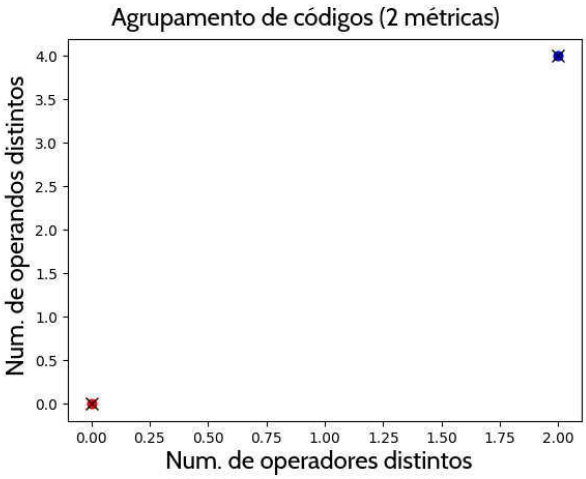
na distância de edição de árvore. O grupo representado pela cor azul possui um único elemento, o Código 2.1, o grupo representado pela cor vermelha possui os códigos 2.3 e 2.5, finalmente, o grupo representado pela cor verde possui um único elemento, o Código 2.2.

A visualização dos agrupamentos é possível quando são utilizadas até três métricas, neste caso os grupos seriam representados em um gráfico no espaço 3-d. Quando uma quantidade maior do que três métricas é utilizada, a visualização dos grupos se torna complexa, pois seria necessário utilizar diversos gráficos, sejam estes em espaços 2-d ou 3-d.

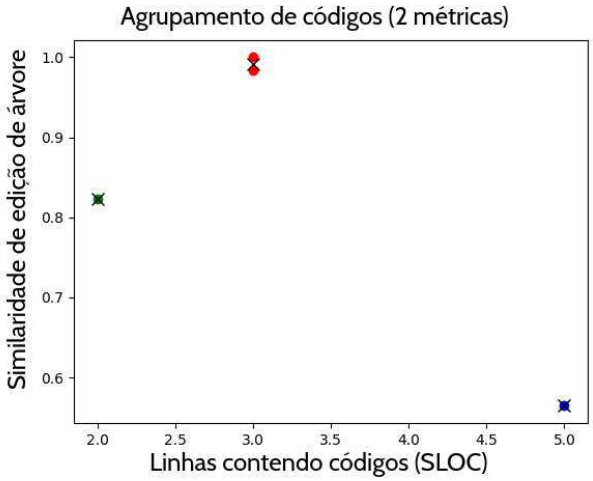
Existem diversas variações de implementação do algoritmo *Kmeans*, tais como, *Kmedians*, onde é utilizada a mediana e não a média; *Fuzzy Cmeans*, onde cada elemento possui um grau de pertinência a cada *cluster* existente, entre outras. A implementação padrão fornecida pelo pacote *Scientific Computing Tools for Python* (*SciPy*) [58] foi utilizada na execução dos



(a) *Kmeans* adotando $k = 3$ e as métricas 'corre-tude funcional' e 'complexidade ciclomática' em cada vetor de propriedades.



(b) *Kmeans* adotando $k = 3$ e as métricas 'número de operadores distintos' e 'número de operandos distintos' em cada vetor de propriedades.



(c) *Kmeans* adotando $k = 3$ e as métricas 'similaridade de edição de árvore' e 'número de linhas contendo cóigos (SLOC)' em cada vetor de propriedades.

Figura 2.3: Execução do algoritmo *kmeans* para os códigos 2.1, 2.2, 2.3, 2.4 e 2.5.

experimentos.

2.3 Medidas estatísticas e de comparação

Três medidas estatísticas foram adotadas para analisar o quão próximas são as avaliações geradas de acordo com a proposta apresentada nesta tese, quando comparadas às avaliações fornecidas pelos avaliadores especialistas. Tais medidas são: o Kappa de Cohen (*Cohen's Kappa*) [10], a correlação de Pearson [3] e a distância total de notas.

Para justificar e ilustar a utilização de cada uma das medidas, são apresentadas avaliações fornecidas por avaliadores especialistas fictícios, sendo os valores exibidos na Tabela 2.8. As avaliações fornecidas por $AF1$ e $AF2$ correspondem a números gerados aleatoriamente através de um programa *Python*. As outras avaliações foram criadas com o intuito de ilustrar o uso das medidas estatísticas e de comparação.

Tabela 2.8: Avaliações fictícias fornecidas por avaliadores fictícios

Avaliadores	$AF1$	$AF2$	$AF3$	$AF4$	$AF5$
	0.00	8.00	6.00	6.00	5.00
	7.00	0.00	6.00	6.00	5.00
	4.00	10.00	7.00	7.00	6.00
	2.00	6.00	8.00	8.00	7.00
	1.00	9.00	8.00	8.00	7.00
	3.00	3.00	10.00	10.00	9.00
	0.00	1.00	3.00	3.00	2.00
	5.00	0.00	9.00	9.00	8.00
	1.00	9.00	7.00	7.00	6.00
	1.00	4.00	7.00	7.00	6.00

2.3.1 *Kappa de Cohen*

Sejam dados dois conjuntos de classificações fornecidos por dois especialistas A e B, deseja-se saber o quanto A e B concordam entre si. A intensidade da concordância entre A e B

pode ser dada pela medida do coeficiente *Kappa de Cohen*. A medida obtida corresponde ao grau de concordância além do que seria esperado pelo acaso. O *Kappa de Cohen* é adequado para obter a concordância entre classificações fornecidas por especialistas par a par.

Para ilustrar o grau de concordância ao acaso, tenha-se a situação descrita a seguir. Sejam dois conjuntos de códigos avaliados pelos avaliadores especialistas A e B, sendo as notas para cada um destes códigos fornecidas com base no valor obtido através do lançamento de um dado. A concordância esperada para os dois conjuntos de classificações fornecidos por A e B seria equivalente a concordância esperada ao acaso dos lançamentos corresponderem ao mesmo valor.

O coeficiente de *Kappa* é obtido de acordo com a Equação 2.4, onde P_o é a proporção de observações, determinada de acordo com a Equação 2.5, P_c é a proporção de concordância ao acaso, determinada de acordo com a Equação 2.6, e P é o número de instâncias observadas.

$$Kappa_{Cohen} = \frac{P_o - P_c}{P - P_c} \tag{2.4}$$

$$P_o = \frac{total\ de\ concordancias}{total\ de\ concordancias + total\ de\ disconcordancias} \tag{2.5}$$

$$P_c = \sum_{i=1}^{n} (p_{i1} \times p_{i2}) \tag{2.6}$$

Sendo, na Equação 2.6, n o total de instâncias observadas, i um índice (entre 1 e n), p_{i1} a proporção da ocorrência de i para o avaliador 1 e p_{i2} a proporção da ocorrência de i para o avaliador 2.

Os valores de *Kappa* podem variar no intervalo $[-1; 1]$, onde: 0 (zero) indica que há concordância, mas ela é igual àquela esperada pelo acaso; valores negativos sugerem discordância, sem interpretação de intensidade; e valores positivos sugerem concordância, onde quanto maior o valor obtido, maior será a intensidade de concordância. Na Tabela 2.9, é apresentada a interpretação relacionada com cada intervalo de valores obtidos para *Kappa*.

Dessa forma, para a medida *Kappa de Cohen*, a hipótese nula testada corresponde à ausência de concordância, ou a concordância igual a que seria obtida ao acaso. Então, temos H_0 a concordância esperada ao acaso, onde $kappa = 0$, e H_1 uma concordância maior do que seria esperado ao acaso, onde $kappa > 0$. Para determinar se a concordância obtida ocorre

Tabela 2.9: Interpretação dos valores para *Kappa de Cohen*.

Intervalo	**Interpretação**
$kappa < 0$	não existe concordância
$kappa = 0$	existe concordância, porém ela é igual à esperada pelo acaso
$0 < kappa < 0.2$	concordância leve
$0.2 \leq kappa < 0.4$	concordância razoável
$0.4 \leq kappa < 0.6$	concordância moderada
$0.6 \leq kappa < 0.8$	concordância substancial
$0.8 \leq kappa < 1.0$	concordância quase perfeita
$kappa = 1$	concordância perfeita

ao acaso ainda é necessário observar o $p-valor$ em comparação a α, o valor de significância. Para os testes realizados nesta pesquisa, foi utilizado $\alpha = 0.05$. Quando $p - valor \leq \alpha$, o nível de concordância indicado por kappa não é devido ao acaso, assim é possível rejeitar H_0. Quando $p-valor > \alpha$, o nível de concordância indicado por kappa pode ocorrer devido ao acaso, assim não é possível rejeitar H_0, seriam necessários mais dados para obter uma evidência significativa.

Aplicando o Kappa de Cohen sobre as avaliações fornecidas, os valores de concordância para cada par de avaliadores fictícios são exibidos na Tabela 2.10. Também são exibidos os $p - valor$ para cada medida de Kappa de Cohen obtida.

A concordância obtida para $AF1$ e $AF2$ correspondeu a 0.01, correspondendo a uma concordância leve. Contudo, o $p-valor$ de 0.902 não permite a rejeição de H_0, ou seja, não é possível afirmar que a concordância encontrada não ocorra devido ao acaso.

Para os avaliadores especialistas fictícios $AF3$ e $AF4$, a concordância obtida correspondeu a 1.00, correspondendo a uma concordância perfeita. Neste caso, o $p-valor$ de $4e-11$ permite a rejeição de H_0, ou seja, é possível afirmar que a concordância encontrada não ocorre devido ao acaso.

Todos os outros pares de avaliadores obtiveram uma concordância negativa, o que pode ser descrito como a ausência de concordância. Em todos os casos, o $p-valor$ é superior a α, ou seja, não é possível afirmar que a concordância encontrada não ocorra devido ao acaso.

Tabela 2.10: Valores de Kappa de Cohen obtidos para cada par de avaliadores especialistas fictícios

Avaliadores	$AF1$	$AF2$	$AF3$	$AF4$	$AF5$
$AF1$	1.00 p-valor = 1e-12	0.01 p-valor = 0.902	-0.04 p-valor = 0.446	-0.04 p-valor = 0.446	-0.05 p-valor = 0.420
$AF2$	0.01 p-valor = 0.902	1.00 p-valor = 1e-12	-0.08 p-valor = 0.319	-0.08 p-valor = 0.319	-0.06 p-valor = 0.364
$AF3$	-0.04 p-valor = 0.446	-0.08 p-valor = 0.319	1.00 p-valor = 1e-12	1.00 p-valor = 4e-11	-0.17 p-valor = 0.142
$AF4$	-0.04 p-valor = 0.446	-0.08 p-valor = 0.319	1.00 p-valor = 4e-11	1.00 p-valor = 1e-12	-0.17 p-valor = 0.142
$AF5$	-0.05 p-valor = 0.420	-0.06 p-valor = 0.364	-0.17 p-valor = 0.142	-0.17 p-valor = 0.142	1.00 p-valor = 1e-12

2.3.2 Correlação de Pearson

O coeficiente de correlação de Pearson é uma medida do grau de correlação entre duas variáveis e qual a direção (positiva ou negativa) desta correlação. O valor da correlação de Pearson (ρ) pode ser calculado através da Equação 2.7.

$$\rho = \frac{cov(X, Y)}{\sqrt{var(X) \times var(Y)}} \tag{2.7}$$

Onde X e Y são séries de valores, $cov(X, Y)$ corresponde a covariância entre X e Y e $var(X)$ $var(Y)$ corresponde a variância de X e de Y respectivamente.

Os valores do coeficiente de correlação de Pearson podem variar no intervalo $[-1, 1]$, onde: os valores negativos indicam uma relação linear decrescente entre as variáveis; o valor zero indica a inexistência de uma relação linear; e os valores positivos indicam uma relação linear crescente entre as variáveis. Na Tabela 2.11, é apresentada a interpretação relacionada com cada intervalo de valores obtidos para correlação de Pearson (ρ).

Calculando o coeficiente de correlação de Pearson sobre as avaliações fornecidas na Tabela 2.8, os valores obtidos são apresentados na Tabela 2.12.

A correlação obtida para $AF1$ e $AF2$ foi equivalente a -0.452, correspondendo a uma

Tabela 2.11: Interpretação dos valores para correlação de Pearson.

Intervalo	**Interpretação**
$0.0 \leq \rho < 0.3$	correlação desprezível
$0.3 \leq \rho < 0.5$	correlação fraca
$0.5 \leq \rho < 0.7$	correlação moderada
$0.7 \leq \rho < 0.9$	correlação forte
$\rho \geq 0.9$	correlação muito forte
$\rho = 1.0$	correlação perfeita

Tabela 2.12: Valores do coeficiente de correlação de Pearson obtidos para cada par de avaliadores fictícios.

Avaliadores	$AF1$	$AF2$	$AF3$	$AF4$	$AF5$
$AF1$	1.00	-0.452	0.315	0.315	0.315
$AF2$	-0.452	1.00	0.103	0.103	0.103
$AF3$	0.315	0.103	1.00	1.00	1.00
$AF4$	0.315	0.103	1.00	1.00	1.00
$AF5$	0.315	0.103	1.00	1.00	1.00

correlação negativa fraca, ou seja, existe uma relação linear decrescente entre as avaliações. A correlação obtida para $AF1$ em relação aos avaliadores $AF3$, $AF4$ ou $AF5$ foi equivalente a 0.315, correspondendo também a uma correlação negativa fraca, ou seja, existe uma relação linear decrescente entre as avaliações. Para o avaliador especialista fictício $AF2$, em relação aos avaliadores $AF3$, $AF4$ ou $AF5$, foi equivalente a 0.103, correspondendo a uma correlação positiva desprezível.

Considerando os avaliadores $AF3$, $AF4$ e $AF5$ par a par, tem-se as correlações equivalentes a 1.00, ou seja, uma correlação perfeita entre as avaliações. Neste caso, é possível interpretar que as variações de avaliações são iguais em todas as avaliações realizadas.

2.3.3 Distância total de notas

Dados dois conjuntos de notas A e B, a distância total de notas entre estes pode ser calculada de acordo com a Equação 2.8, onde a_i representa cada nota existente no conjunto A, e b_i representa cada nota existente no conjunto B. Os conjuntos comparados devem possuir o mesmo tamanho, dessa forma, uma nota inexistente em um conjunto deve ser ignorada no outro.

$$DistanciaNotas = \sum_{i=1}^{tam} |a_i - b_i| \tag{2.8}$$

Calculando a distância total de notas sobre as avaliações fornecidas na Tabela 2.8, os valores obtidos são apresentados na Tabela 2.13.

Tabela 2.13: Valores das distâncias de notas obtidas para cada par de avaliadores fictícios.

Avaliadores	$AF1$	$AF2$	$AF3$	$AF4$	$AF5$
$AF1$	0	50	49	49	41
$AF2$	50	0	37	37	35
$AF3$	49	37	0	0	10
$AF4$	49	37	0	0	10
$AF5$	41	35	10	10	0

A distância total de notas para $AF1$ e $AF2$ correspondeu ao valor 50, em um conjunto de dez notas isso significaria uma diferença média de cinco pontos entre cada uma das avaliações. Um valor similar foi obtido quando as avaliações de $AF1$ foram comparadas a $AF3$ e $AF4$. Considerando $AF1$ e $AF5$, obteve-se o valor 41 em um conjunto de dez notas, isso significaria uma diferença média de aproximadamente quatro pontos entre cada uma das avaliações.

Considerando $AF2$ em relação aos avaliadores $AF3$, $AF4$ e $AF5$, a distância total de notas correspondeu aos valores 37, 37 e 35 em um conjunto de dez notas, isso significaria uma diferença média de aproximadamente três pontos e meio entre cada uma das avaliações.

Para os avaliadores especialistas fictícios $AF3$ e $AF4$, não há distância total de notas, uma vez que o conjunto de avaliações é o mesmo. Considerando $AF3$ e $AF4$ em relação ao avaliador $AF5$, a distância total de notas correspondeu ao valor 10 em um conjunto de

dez notas, isso significaria uma diferença média de apenas um ponto entre cada uma das avaliações.

2.3.4 Considerações sobre as medidas estatísticas e de comparação

Observando na Tabela 2.8 o exemplo das avaliações fornecidas e os valores obtidos para cada uma das medidas adotadas, podemos notar que quando uma concordância perfeita ocorre, todas as outras medidas têm também seus melhores resultados associados.

Contudo, ao obter uma concordância que não seja perfeita, é necessário observar os resultados de correlação e distância de notas para esclarecer se existe grande distinção entre as avaliações.

Quando os avaliadores $AF1$ e $AF3$ são comparados, observa-se uma correlação equivalente a 0.315, correspondendo também a uma correlação negativa fraca, e uma distância total de notas de 49. Desta forma, é possível afirmar que os avaliadores $AF1$ e $AF3$ realizam avaliações de forma muito distinta.

Quando os avaliadores $AF3$ e $AF5$ são comparados, observa-se uma correlação equivalente a 1.00, correspondendo também a uma correlação positiva perfeita, e uma distância total de notas de 10. Desta forma, é possível afirmar que os avaliadores $AF3$ e $AF5$ realizam avaliações de forma muito similar.

Capítulo 3

Trabalhos relacionados

Neste capítulo, são descritos os trabalhos relacionados ao tema pesquisado, busca-se caracterizar onde este trabalho se insere, bem como descrever quais os principais diferenciais abordados nesta pesquisa em relação a outras propostas.

Trabalhos relacionados ao contexto de geração automática de avaliações, *automatic grading* e *automatic feedback*, não são recentes. O trabalho de Hext e Winings [34], além do trabalho de Forsythe e Wirth [27], por exemplo, datam da década de sessenta. A proposta apresentada em [34] descreve a substituição do sistema de processamento em lotes, usado para calcular notas, por um sistema que, entre outras funcionalidades, utiliza uma abordagem de avaliação para linguagens *Algol*, *Nigol* e *Assembly*, similar aos atuais juízes online. No artigo [27], descreve-se duas abordagens de avaliação, uma baseada na geração de dados aleatórios e verificação das saídas esperadas, e outra que procura propriedades desejáveis no código do programa, sendo a única linguagem suportada *Algol*.

Com a popularização e evolução da área de computação, diversas abordagens para a geração automática de avaliações foram propostas. Muitas destas continuam explorando ideias similares aos trabalhos de Hext e Winings [34] e Forsythe e Wirth [27], onde são fornecidas entradas, determinadas ou aleatórias, e são observadas as saídas. Outros trabalhos têm investigado o uso de conceitos da Inteligência Artificial, tais como [45; 9; 63; 29], da Engenharia de Software, como exemplo [4; 16], ou na análise de propriedades extraídas dos códigos, tais como [56; 22; 54; 16], para propor métodos e ferramentas de apoio aos ambientes de ensino e aprendizagem em programação.

No processo de revisão bibliográfica conduzido pelo autor deste trabalho, buscou-se iden-

tificar para quais objetivos a análise de código é adotada como apoio ao professor em disciplinas introdutórias de programação. Nas seções seguintes é apresentado o estado da arte pesquisado em [13] e atualizado posteriormente.

3.1 Avaliadores de códigos

A principal estratégia identificada na literatura como meio para facilitar e agilizar a avaliação de códigos é o uso de juízes online. Citamos [35; 49; 72; 24] como exemplos de trabalhos relacionados ao contexto de juízes online.

Em [69], um juiz online é definido como um sistema online que executa ao menos um dos seguintes passos de avaliação: (a) coleta e compila/interpreta os códigos; (b) avalia soluções com base em um conjunto de testes; e (c) computa um *score* de avaliação

Um juiz online utiliza uma abordagem baseada em um conjunto de casos de teste, para indicar se um código é correto. Um caso de teste corresponde à especificação de conjunto de entradas e as respectivas saídas esperadas [20]. Sendo assim, as aplicações do tipo juiz online consideram corretos os códigos que fornecem as saídas esperadas para cada entrada fornecida, em outro caso o código é dito incorreto.

A abordagem utilizada por juízes online não avalia critérios de qualidade em um código, em geral, se uma solução fornece o conjunto de saídas esperado ela é considerada totalmente correta.

Diversas iniciativas buscam aprimorar a abordagem de juízes online. Algumas aplicações, tal como [49], utilizam meios para julgar se um código é parcialmente correto, sendo empregado para isso o percentual de aprovação no conjunto de testes.

Outras propostas de juízes online complementam a indicação de sucesso, ou falha, fornecida pelos testes com dicas que auxiliem o aluno a identificar e corrigir os erros. Tais dicas são associadas a um caso de teste com base na experiência do testador/professor em determinar a provável causa da falha.

Sistemas de recomendação de questões, tal como [7], foram propostos como meio de selecionar problemas que mais se adequem ao momento cognitivo do discente. Diversas técnicas são utilizadas para gerar recomendações.

Outras estratégias descritas com uma menor frequência na literatura como meios para

apoiar a avaliação de códigos, são: o uso de análise estática sobre os códigos; a análise estrutural, em geral baseada em grafos; o uso de ?arvores sintáticas; outras.

3.2 Avaliadores de similaridade entre códigos

Uma das principais preocupações em ambientes de ensino e aprendizagem de programação tem sido a detecção de plágio, muito esforço tem sido empregado em desenvolver algoritmos e abordagens de detecção para essa finalidade. Contudo, a similaridade entre códigos também tem sido explorada em outros trabalhos, com objetivos distintos da detecção de plágio.

Os trabalhos de Gaudêncio [56], Biggers [5] e Li [39] são citados como exemplo de propostas que utilizam a análise de similaridades finalidades distintas da detecção de plágio.

Li [39] e Biggers [5] exploram métricas de software para a identificação de similaridades sob uma ótica mais próxima do ensino de Engenharia de Software. Busca-se, nestes trabalhos, identificar a aderência do código ao que foi especificado como requisito de software.

No trabalho de Gaudêncio [56] é investigado se avaliadores automáticos (algoritmos de avaliação de similaridade) podem comparar códigos de alunos tão bem quanto avaliadores humanos. Como resultados do experimento realizado, obteve-se uma concordância entre professores no intervalo de 62% a 95% e, entre professores e algoritmos, as taxas de concordância com o melhor resultado é sempre superior a 90%. Esses resultados motivaram a criação de ferramentas de visualização [22] que buscam explorar a proximidade dos códigos como meio de facilitar a atividade de avaliação.

3.3 Abordagens baseadas em agrupamento e/ou métricas de software

Dentro do contexto de avaliação de códigos, investigamos os trabalhos que buscam empregar o conceito de similaridades entre os códigos como meio de auxiliar o processo de avaliação. Observando os trabalhos apresentados nos últimos anos relacionados com este objetivo, encontramos propostas baseadas no uso de técnicas de Inteligência Artificial (IA) e abordagens que se baseiam em métricas de software.

Diversas técnicas de IA têm sido empregadas, não só com a finalidade de agrupar soluções similares. Facilmente é possível encontrar trabalhos que exploram regressão linear, *random forests*, algoritmos de agrupamento, *SVMs*, entre outras. Muitas vezes, o uso de técnicas de IA é realizado em conjunto com métricas de software, as quais, em geral, são utilizadas como entrada para uma ou mais destas técnicas.

A seguir, os trabalhos que utilizam IA e/ou métricas de software, de modo a auxiliar a atividade de avaliação de código, e que exibem maior similaridade com a proposta desta tese serão descritos. Busca-se, ao longo de cada apresentação, pontuar quais os principais diferenciais abordados nesta pesquisa em relação a outras propostas.

Na pesquisa conduzida por Silva, em [59], é proposto um arcabouço capaz de combinar diferentes tipos de analisadores de código. Nos analisadores utilizados, podem ser observados construtos, estrutura do código, corretude sintática, corretude semântica e outros parâmetros. O arcabouço foi usado para avaliar 816 códigos *Python*, propostos como soluções submetidas por discentes de um curso de Sistemas de Informação. As soluções foram analisadas com base nos seguintes tópicos: variáveis e comandos de entrada e saída; operadores e expressões; estruturas de seleção; e estruturas de repetição. Como resultado, foi possível identificar grupos de estudantes, como alunos que não conseguiram construir soluções estruturalmente coesas. Também foi possível identificar alunos que superam as expectativas como alunos iniciantes em programação. Neste trabalho, buscou-se analisar os códigos para agrupar os alunos em relação aos conhecimentos identificados em cada conjunto de submissões.

Os trabalhos de [9; 45; 71] são parte das pesquisas conduzidas pelo grupo liderado pelo professor Armando Fox, da UC Berkeley. Nestas pesquisas, são utilizados o algoritmo de agrupamento *OPTICS*, a métrica ABC [26] e similaridade baseada em distância de árvores, com o intuito de auxiliar, através de dicas, estudantes matriculados em cursos de Engenharia de Software a produzir código com maior 'qualidade' ou, de acordo com os autores, 'bons estilos de programação'. As dicas fornecidas são classificadas em três grupos: dicas de abordagem, dicas sintáticas e esqueletos de código. As dicas de abordagem descrevem meios para o estudante atacar o problema de forma mais eficiente. Dicas sintáticas apresentam formas de uso de estruturas da linguagem (ex. estrutras de repetição ou condição) que melhorem o estilo do código. Já os esqueletos de código são partes de submissões de outros

estudantes que apresentam uma solução melhor. Os experimentos reportados nos trabalhos incluem turmas presenciais da UC Berkeley e exercícios que ocorreram na disciplina "*Engineering Software as a Service*" disponibilizada no portal edX [1]. Neste trabalho, o foco é sobre alunos que já possuem competências básicas na área de programação, de modo que estes adotem melhores práticas, de acordo com conceitos de ES, durante o desenvolvimento de programas utilizando técnicas mais avançadas.

A abordagem descrita por Srikant, em [63], é uma das que mais se aproxima da proposta aqui apresentada. Neste trabalho, busca-se aplicar algoritmos de aprendizagem de máquina para identificar quais propriedades são levadas em consideração na avaliação de um problema. No artigo onde a pesquisa é descrita, é proposto um conjunto fixo de propriedades, as quais são identificadas a partir de entrevistas com avaliadores humanos. Para verificar a corretude da abordagem proposta, um conjunto de avaliadores humanos realizou a correção utilizando uma rubrica, proposta pelos autores do artigo. Para os casos onde não havia concordância entre os avaliadores, estes deveriam conversar e entrar em consenso para fornecer um resultado único. Como resultado, é gerada uma avaliação numa escala de 1 até 5 para as submissões avaliadas. Os resultados mostram a ocorrência de correlação alta entre as avaliações fornecidas pela abordagem e a avaliação consensual dos avaliadores humanos. Segundo descrito em [63], até o desenvolvimento de sua pesquisa, não foram encontrados outros trabalhos relacionados ao contexto de pesquisa, que possuam propostas similares à exibida no artigo. Neste trabalho, a avaliação utilizada na pesquisa deve ser consensual entre os avaliadores, ou seja, não há adaptação aos possíveis diferentes critérios.

Nas pesquisas desenvolvidas por Elena Glassman [30; 29; 33], são propostas abordagens e ferramentas que visam prover *feedback* sobre soluções submetidas por alunos na resolução de problemas, os quais foram propostos como exercícios em disciplinas de programação. Em [30], é proposta uma abordagem onde discentes geram *feedback* que poderá ser útil a outros discentes. A abordagem proposta foca em coletar dicas a cada ciclo de resolução de *bugs* que o aluno se depara. Quando outro discente passa por uma situação semelhante, as dicas geradas são apresentadas. Na tese de doutorado [29], a pesquisadora investiga uma abordagem baseada no agrupamento de trechos de código, os quais representam as variações entre as soluções apresentadas. Exibindo os trechos de código representativos de cada grupo,

[1] www.edx.org/

para auxiliar o professor na geração de *feedback*, casos de teste e na criação de rubricas para avaliação. O *feedback* capturado foca em nomes de variáveis para soluções corretas após a avaliação de um juiz online. Em uma pesquisa mais recente [33], é proposta uma abordagem para auxiliar um aluno a corrigir *bugs* em códigos e gerar uma solução correta. Os códigos são agrupados de acordo com os *bugs* apresentados, para cada grupo são geradas correções utilizando transformações automáticas de código e alterações sugeridas pelos professores. Nestas pesquisas, os aspectos de avaliação são relacionados à correção sintática de códigos, ou são aplicadas melhorias, relacionadas a aspectos de legibilidade, sobre códigos funcionalmente corretos. Além disso, são observados trechos de códigos, uma solução não é observada como um todo.

No trabalho de Eliane Araújo [17; 18; 16], assim como nas pesquisas de Elena Glassman, são propostas abordagens e ferramentas que visam prover *feedback* sobre soluções submetidas por alunos na resolução de problemas, os quais foram propostos como exercícios em disciplinas de programação. As pesquisas são desenvolvidas sobre códigos considerados funcionalmente corretos. Busca-se um meio de exibir e incentivar discentes a melhorar as soluções propostas, mesmo quando estas são consideradas funcionalmente corretas. O *feedback* é gerado com base em um conjunto de métricas de software e sua relação com uma solução de referência fornecida. Nestas pesquisas, os aspectos de avaliação são observados para códigos funcionalmente corretos. A abordagem emprega um conjunto de métricas semelhante ao utilizado nesta tese, mas não utiliza agrupamentos dos códigos.

3.4 Contribuições propostas e diferenciais em relação ao estado da arte

A principal contribuição existente nesta tese de doutorado é minimizar o esforço de avaliação de forma adaptada aos critérios de cada avaliador. Compreendendo avaliação como o processo de observar a solução e fornecer *feedback* objetivo (classificação ou nota) e *feedback* subjetivo (informação textual) ao discente.

O trabalho de Srikant [63] propõe um método baseado em aprendizagem de máquina que identifica propriedades levadas em consideração na avaliação de um problema, ou seja, podemos descrever que o trabalho é adaptável ao problema avaliado, enquanto o método

aqui proposto é adaptável ao avaliador e ao problema. Outro diferencial em relação ao trabalho exibido em [63] ocorre em relação à generalização do *feedback* textual, característica inexistente no trabalho citado.

As pesquisas desenvolvidas pelo grupo de Berkeley, [9; 45; 71], também possuem similaridade com a proposta aqui apresentada. Contudo, o contexto de aplicação destas é diferente, o foco do grupo de Berkeley é em relação às boas práticas de programação, ou Engenharia de Software, enquanto o método aqui proposto possui como foco o aprendizado de programação. A melhoria dos códigos é esperada como resultado principal do grupo liderado pelo professor Armando Fox, um menor esforço de avaliação é o principal resultado esperado na presente tese.

Nenhum dos trabalhos anteriores utiliza técnicas de aprendizagem de máquina para fornecer *feedback* adaptável, em relação ao avaliador que fornece este *feedback*. Diversas técnicas que obtiveram diferentes níveis de sucesso foram propostas, contudo estas são fixas em relação ao modo como os códigos são avaliados ou diferem na abordagem ou nos objetos de estudo sobre os quais são aplicadas.

As pesquisas desenvolvidas pela pesquisadora Elena Glassman [30; 29; 33] possuem objetivos semelhantes aos desta tese, contudo as abordagens e o foco são diferentes. Na abordagem proposta em [29], são utilizados agrupamentos em relação a trechos de código comuns às submissões, o *feedback* gerado se refere a variáveis existentes no código e apenas soluções funcionalmente corretas são observadas. Já nas pesquisas [30; 33], o foco é corrigir códigos sintaticamente incorretos, uma das abordagens agrupa os códigos de acordo com os *bugs* apresentados.

De forma similar, as pesquisas conduzidas pela pesquisadora Eliane Araújo [17; 18; 16] possuem objetivos semelhantes aos desta tese, do mesmo modo as abordagens e o foco são diferentes nestas pequisas. O conjunto de métricas semelhante é empregado na avaliação de códigos em ambas as pesquisas. Contudo, na abordagem proposta por Eliane Araújo, as análises são conduzidas a partir de relações entre o código analisado e uma solução de referência, nesta pesquisa as métricas são empregadas no agrupamento das soluções. O foco do trabalho de Eliane Araújo é prover meios para incentivar o aluno a submeter soluções com melhor qualidade, nesta pesquisa desejamos fornecer ao professor um meio para distribuir *feedback* aos alunos.

Desta forma, acreditamos que a presente pesquisa possui relevância e originalidade para que se mostre compatível com o esperado de uma tese de doutorado.

Capítulo 4

Abordagem de agrupamento de códigos

Neste capítulo, são apresentados os detalhes da condução da pesquisa que investiga a minimização do esforço dispendido na avaliação de códigos, através do uso de agrupamento de códigos baseados em métricas de software. Nas seções seguintes, são apresentados o repositório de dados utilizado e o ambiente associado a este, a descrição dos dados e dos avaliadores especialistas, a metodologia empregada, os resultados obtidos, as limitações da proposta e as possíveis ameaças à validade, relacionadas com cada uma das etapas desta tese.

4.1 Repositório de dados e o ambiente *The Huxley*

O repositório de dados sobre o qual foram extraídos os problemas (exercícios) e as soluções propostas pelos discentes (códigos) inclui aproximadamente 1800 problemas, utilizados em cerca de 1600 listas de exercícios, propostas como atividades em mais de 300 turmas pertencentes a cerca de 150 instituições de ensino. As submissões para estes problemas totalizam mais de 750.000 códigos submetidos, sendo o histórico de submissões mantido para cada aluno e problema.

O referido repositório de dados é parte do ambiente *The Huxley* [49] e foi disponibilizado para execução das pesquisas relacionadas com a presente tese de doutorado através de contrato de pesquisa entre a empresa que desenvolveu e mantém o sistema e o autor desta tese.

O ambiente *The Huxley* é uma ferramenta web de apoio a atividades de ensino e aprendizagem de programação. O sistema possui funcionalidades úteis ao professor e aos alunos

destas disciplinas.

Através do sistema, o professor pode criar listas de exercícios de forma rápida, acompanhar o desempenho de seus alunos, observar a quantidade de problemas resolvidos, bem como a porcentagem de acertos/erros por tipo de questão e por aluno, além de receber uma lista de prováveis ocorrências de plágio nas submissões.

Para os alunos, é possível submeter códigos em diferentes linguagens de programação, e receber *feedback* da avaliação. Este *feedback* se baseia em um conjunto de testes de aceitação, onde a cada submissão o aluno pode observar sobre quais testes sua solução passou e para quais essa não forneceu as saídas esperadas.

Figura 4.1: Tela de descrição de um problema.

Na Figura 4.1, é apresentada a visão que o aluno possui ao observar a descrição de um problema. O aluno pode observar a descrição do problema e as restrições relacionadas à entrada e saída dos dados, além de um exemplo de execução onde são exibidas a entrada

fornecida e a saída esperada para aquela entrada.

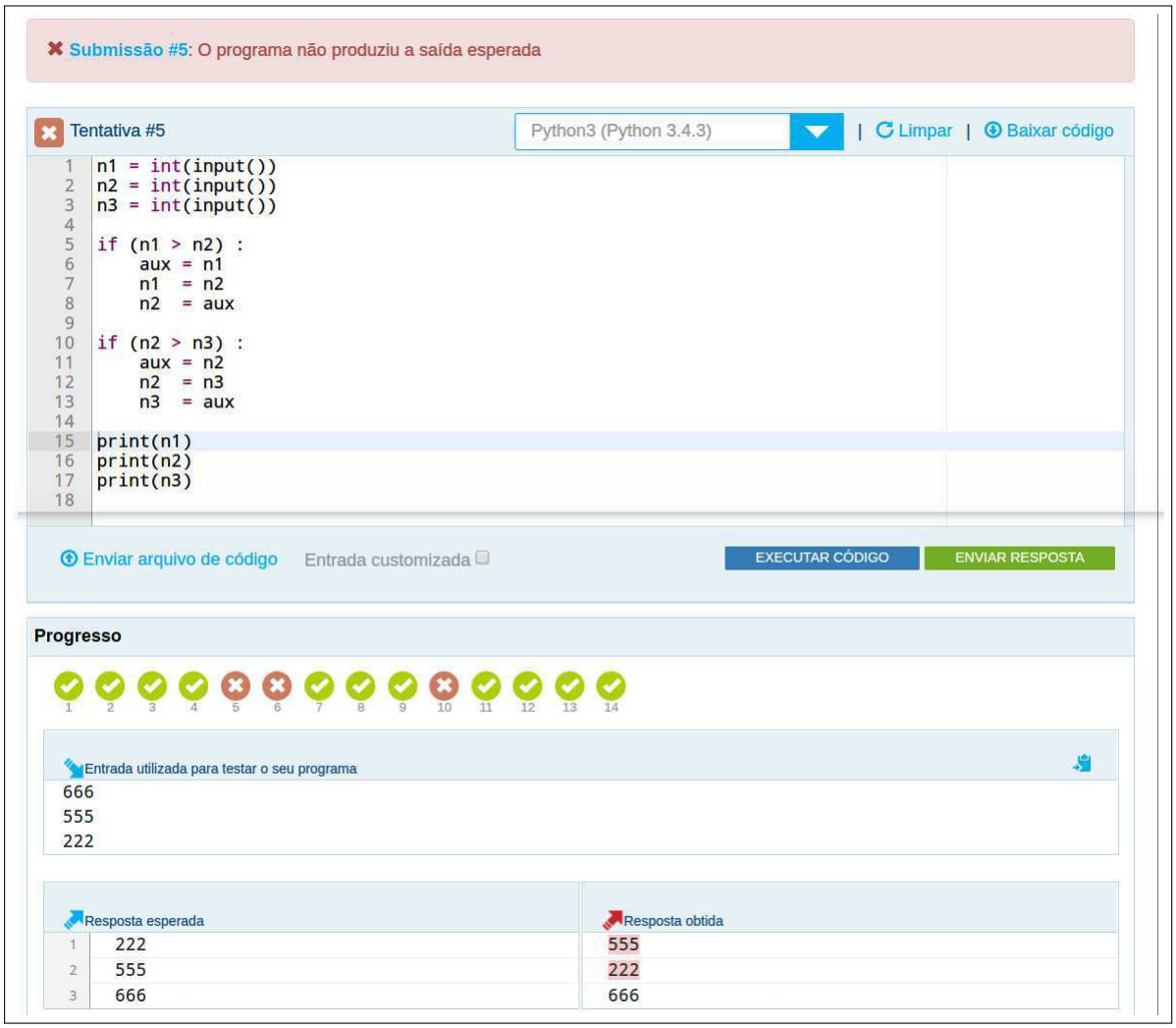


Figura 4.2: Tela de submissão e descrição de erros.

Na Figura 4.2, é apresentada a tela de submissão de códigos. Nesta tela, o aluno pode selecionar um arquivo de código desenvolvido em seu computador ou utilizar o editor de código do ambiente. Para submeter uma solução, o aluno deve fornecer um código e indicar a linguagem e a versão adotada, e então clicar sobre o botão de envio para aguardar o retorno do sistema. O sistema fornece a indicação dos casos de teste que passaram e quais falharam, possibilitando que o aluno observe a entrada fornecida no teste e as saídas gerada e esperada.

Uma visão geral de uso do sistema pode ser descrita através dos seguintes processos:

1. O professor cria uma lista de exercícios, selecionando questões da base do sistema

ou propondo novos exercícios. Cada exercício deve conter elementos descritivos (ex. título, enunciado, entradas e saídas esperadas) e testes de aceitação;

2. O aluno acessa a lista de exercício, seleciona um dos problemas, observa sua descrição e pode submeter um código como solução;

3. A submissão é avaliada utilizando um conjunto de testes, e o resultado é exibido ao aluno. Caso ocorra falha em algum teste, detalhes do erro são exibidos, sendo fornecida ao aluno uma dica para solução do problema. O aluno pode submeter outras soluções, quantas vezes desejar, desde que seja respeitada a data limite de submissão especificada pelo professor no momento da criação da lista de exercícios;

4. O histórico de submissões de todos os alunos da disciplina fica disponível para consulta do professor.

4.2 A abordagem de agrupamentos adaptativos

Seja proposto um problema e fornecido um conjunto de construções de uma linguagem de programação para codificar as soluções para este problema, é esperado que as soluções implementadas utilizem as construções e que em muitas destas ocorra um grau de similaridade no uso de tais construções. A abordagem proposta se baseia na ideia de que a similaridade no uso destas construções possibilita o agrupamento de soluções através das medidas extraídas a partir dos códigos.

O conceito de similaridade é subjetivo, portanto, para um avaliador duas entidades podem ser similares, enquanto para outro as mesmas entidades não apresentem similaridades. A abordagem proposta busca se adaptar a esta subjetividade, procurando identificar quais características presentes nos códigos são observadas por cada avaliador. As características de um código são representadas por métricas de software, descritas no Capítulo 2.

Para ilustrar o uso das métricas de forma adaptável na abordagem de agrupamento, é apresentado o seguinte exemplo. Considerando o enunciado de um problema de programação “Forneça todos os números pares no intervalo [1;10]”, e observados os códigos exibidos em 4.1, 4.2, 4.3 e 4.4 como soluções propostas por discentes.

Código Fonte 4.1: Solução 1 para o problema ‘Todos os pares no intervalo’

```
1 print(2)
2 print(4)
3 print(6)
4 print(8)
5 print(10)
```

Código Fonte 4.2: Solução 2 para o problema ‘Todos os pares no intervalo’

```
1 for i in range(2,11,2):
2     print(i)
```

Código Fonte 4.3: Solução 3 para o problema ‘Todos os pares no intervalo’

```
1 for i in range(1,11):
2     if (i % 2 == 0) :
3         print(i)
```

Código Fonte 4.4: Solução 5 para o problema ‘Todos os pares no intervalo’

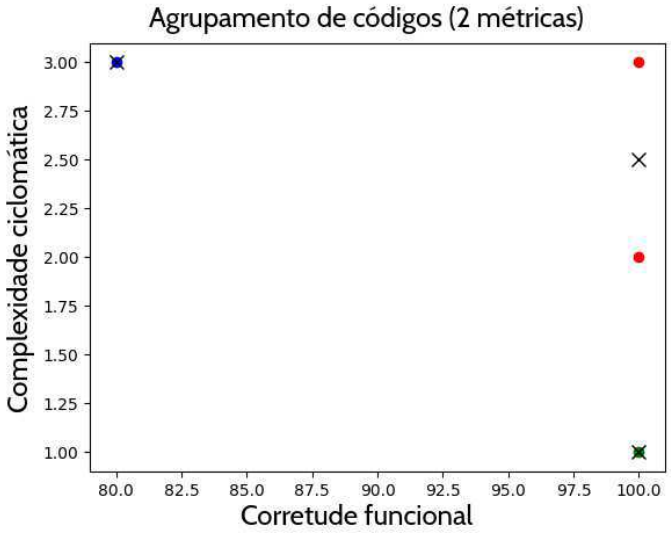
```
1 for i in range(1,10):
2     if (i % 2 == 0) :
3         print(i)
```

Um avaliador que observe os códigos considerando se as saídas corretas são geradas, e associando isso ao uso de instruções condicionais e de repetição, poderia ter seu critério representado pelas métricas ‘corretude funcional’ e ‘complexidade ciclomática’. As medidas obtidas pela aplicação das métricas citadas sobre os Códigos 4.1, 4.2, 4.3 e 4.4 são exibidas na Tabela 4.1.

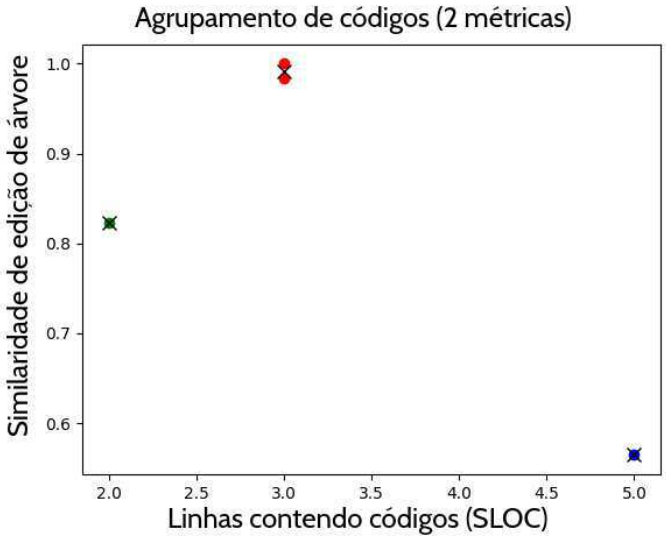
Tabela 4.1: Medidas de corretude sintática e corretude funcional para os Códigos 4.1, 4.2, 4.3 e 4.4

	Códigos			
Métricas	4.1	4.2	4.3	4.4
Corretude funcional	100%	100%	100%	80%
Complexidade ciclomática	1	2	3	3

Adotando as métricas ‘corretude funcional’ e ‘complexidade ciclomática’, o agrupamento gerado com o algoritmo *Kmeans*, usando $k = 3$, é exibido na Figura 4.3a. As cores determinam o grupo ao qual um elemento pertence. Este agrupamento poderia ser adequado para o avaliador descrito, mas não corresponder aos critérios de similaridade entre as soluções adotados por um outro avaliador.



(a)



(b)

Figura 4.3: Execução do algoritmo *kmeans* para os Códigos 4.1, 4.2, 4.3 e 4.4.

Um outro cenário de agrupamento ocorre ao observarmos os mesmos códigos, porém para um segundo avaliador especialista. Sendo os critérios deste avaliador especialista re-

lacionados com a observação do tamanho total da solução, associado com a estrutura de instruções condicionais e de repetição utilizadas. Este critério poderia ser relacionado com as métricas 'quantidade de linhas contendo código (*SLOC*)' e 'similaridade de edição de árvore'. As medidas obtidas pela aplicação das métricas citadas sobre os Códigos 4.1, 4.2, 4.3 e 4.4 são exibidas na Tabela 4.2.

Tabela 4.2: Medidas de corretude sintática e corretude funcional para os Códigos 4.1, 4.2, 4.3 e 4.4

	Códigos			
Métricas	4.1	4.2	4.3	4.4
Quantidade de linhas contendo código (*SLOC*)	5	2	3	3
Similaridade de edição de árvore	0.56	0.82	0.98	1.00

Adotando as métricas 'quantidade de linhas contendo código (*SLOC*)' e 'similaridade de edição de árvore', o agrupamento gerado com o algoritmo *Kmeans*, usando $k = 3$, é exibido na Figura 4.3b. As cores determinam o grupo ao qual um elemento pertence. A métrica 'similaridade de edição de árvore' foi obtida através da comparação das soluções exibidas com uma solução de referência, neste caso igual ao Código 4.3.

A solução de agrupamento proposta pode se adaptar aos critérios adotados por cada avaliador especialista na avaliação das submissões para um problema. Uma vez que o conjunto de propriedades representa, com algum grau de proximidade, os critérios adotados por um avaliador especialista, a seleção de diferentes propriedades corresponde a adaptação aos diferentes critérios de cada avaliador.

A redução do esforço de avaliação ocorreria, pois nos cenários descritos, cada avaliador realizaria a avaliação de apenas uma solução de cada grupo. Desta forma, as avaliações das outras soluções existentes no grupo corresponderiam à generalização da avaliação realizada. No cenário simplificado apresentado, um avaliador realizaria a avaliação de 3 soluções e não das 4 soluções apresentadas.

4.2.1 Execução da abordagem

As seguintes etapas, ilustradas na Figura 4.4, devem ser executadas na abordagem de agrupamentos adaptativos proposta:

- Extração das medidas sobre os códigos (Etapa 1);
- Identificação das propriedades para agrupamento (Etapa 2);
- Geração de grupos ou *clustering* (Etapa 3).

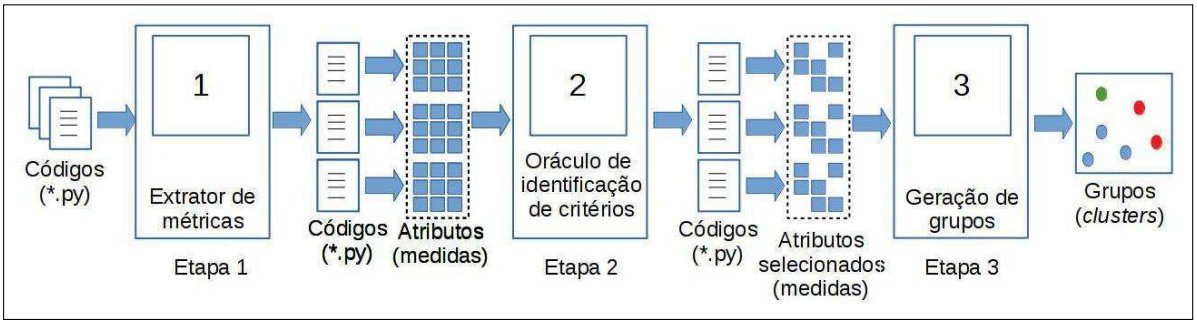


Figura 4.4: Ilustração das etapas da abordagem de agrupamentos adaptativos.

A extração das medidas sobre os códigos corresponde à computação de cada uma das métricas de software adotadas. Desta forma, para cada código, o conjunto de métricas é executado e o valor medido é salvo como um atributo do código. Todas as métricas que não correspondem aos cálculos de similaridade podem ser extraídas considerando unicamente o código analisado. As métricas que envolvem cálculos de similaridade necessitam de um código de referência para comparação deste em relação ao código analisado. Um subconjunto destes atributos será utilizado como entrada para o algoritmo de agrupamento, essa seleção ocorre na etapa seguinte.

Nesta etapa, deve-se identificar as propriedades utilizadas para gerar os agrupamentos (*clusters*). Diferentes abordagens podem ser utilizadas nesta etapa, por isso representamos esta etapa com um oráculo, a saída produzida deve ser o conjunto de propriedades (métricas de software) que represente os critérios adotados por um avaliador especialista.

O processo de seleção de métricas pode ocorrer através da indicação das métricas por parte do especialista, ou indicação de critérios que possam ser relacionados com ao menos

uma métrica. Por exemplo, o avaliador pode descrever que o tamanho total da solução proposta é importante para sua avaliação, uma métrica que por ser relacionada ao critério é a quantidade de linhas contendo código.

Este mesmo processo também pode ser realizado através de uma seleção automatizada. Neste caso, é preciso utilizar avaliações já realizadas pelo avaliador para identificar um conjunto de métricas que se enquadre melhor na distribuição dos elementos em cada grupo.

A geração de grupos ou *clustering* foi realizada fazendo-se uso do algoritmo *Kmeans*, outros algoritmos de agrupamento podem ser utilizados, contudo este estudo comparativo não faz parte do escopo desta pesquisa. As propriedades identificadas na etapa anterior são fornecidas como entrada para o algoritmo de agrupamento, o qual também deve receber o valor 'k', correspondendo ao número máximo de *clusters* (grupos) que podem ser criados. Cada grupo criado é identificado com um valor numérico no intervalo $[0, k-1]$, sendo este valor representado graficamente por uma cor. Cada grupo engloba elementos com características similares (medidas próximas), não tendo sua identificação (o valor numérico do grupo) nenhuma semântica associada. Desta forma, dependendo do conjunto de propriedades adotado, um mesmo código pode pertencer a grupos diferentes.

4.2.2 Escopo e limitações da abordagem proposta

Tradicionalmente, em disciplinas de programação, são adotadas atividades práticas de codificação. A cada especificação de exercícios, o professor se depara com dois objetivos: prover aos alunos um conjunto de exercícios úteis ao aprendizado; e ponderar o custo e esforço relacionados com a avaliação dos códigos que serão submetidos. Diversas pesquisas já tiveram como foco abordagens e ferramentas para facilitar as atividades de avaliação em disciplinas de programação. Como exemplos destas, tem-se: os juízes online [49; 8; 57], ferramentas que verificam a corretude de um código utilizando um conjunto de testes de aceitação; detectores de plágio [44; 12], abordagens que identificam o grau de similaridade existente entre programas, além de diversos tipos de analisadores de código.

A presente pesquisa visa minimizar o esforço de avaliação, tendo foco no professor. Neste processo de avaliação, o professor deve conseguir, sem que seja necessário um grande esforço, fornecer um *feedback* objetivo (critério ou nota) e um *feedback* subjetivo (informação textual) aos alunos que submeteram códigos como solução para um problema.

Sendo assim, não é parte desta pesquisa avaliar se o aluno está aprendendo melhor ou mais rápido ao observar os *feedbacks* fornecidos. Também não foi realizada nenhuma intervenção junto ao professor, de modo a sugerir mudanças em relação ao *feedback* fornecido.

Algumas limitações ocorrem na abordagem proposta, tais limitações estão mais relacionadas ao uso de métricas para representar as propriedades de agrupamento.

Uma vez que a abordagem tem como passo inicial a etapa de extração de métricas, códigos com problemas de sintaxe não são completamente analisados. Algumas métricas, como por exemplo a contagem de linhas, podem ser computadas independente da corretude sintática de um código. Contudo, a maioria das métricas adotadas exige que o código seja sintaticamente correto para que essa possa ser computada.

Códigos que executem computações totalmente distintas podem ter medidas equivalentes. Observando os Códigos 4.5 e 4.6, podemos descrever que em 4.5 é computado o valor do fatorial de um número fornecido, e em 4.6 é computada a lista de valores da sequência de Fibonacci menores que um número fornecido. Mesmo estes códigos realizando computações distintas, suas medidas para diversas métricas são iguais, como exibido na Tabela 4.3.

Código Fonte 4.5: Código Python que computa o valor do fatorial de um número fornecido pelo usuário

```
1 n = int(input("Digite o valor: "))
2 fat = 1
3 for i in range(2, n+1):
4     fat = fat * i
5     i = i + 1
6 print(fat)
```

Código Fonte 4.6: Código Python que computa a lista de valores da sequência de Fibonacci menores que um número fornecido

```
1 n = int(input("Digite o valor: "))
2 a = 0
3 b = 1
4 while b < n:
5     print("%d" % b)
6     a, b = b, a+b
```

Tabela 4.3: Médias obtidas sobre os Códigos 4.5 e 4.6 quando aplicadas às métricas indicadas.

	Métricas												
Códigos	Complexidade ciclomática	*LOC*	*LLOC*	*SLOC*	Linhas de comentário	Linhas em branco	Número de operadores distintos	Número de operandos distintos	Número total de operadores	Número total de operandos	Vocabulário	Tamanho	Volume
4.5	2	6	6	6	0	0	2	4	3	6	6	9	23.26
4.6	2	6	6	6	0	0	3	4	3	6	7	9	25.26

Um conjunto de métricas representa, com algum grau de proximidade, os critérios adotados por um avaliador especialista. Critérios que possuem uma relação direta com uma métrica, ou com um conjunto de métricas, refletem em melhores resultados. Critérios que não possuam relação com métricas de software, não poderão ser representados fielmente na abordagem de avaliação proposta. Como exemplo, o raciocínio adotado no processo de solução empregado pelo aluno, não se traduz em termos de medidas de um código tomado isoladamente, seria necessário observar todos os códigos na ordem cronológica em que foram desenvolvidos e construir relações entre estes. Para este tipo de avaliação é necessário intervenção humana.

4.2.3 Etapas de análise da abordagem proposta

Após apresentar uma descrição, as etapas de execução, o escopo e as limitações da abordagem de agrupamentos adaptativos proposta nesta tese. Nas seções seguintes, são apresentadas as etapas realizadas como meios de avaliação da abordagem de agrupamento, sendo estas:

- **Survey sobre ensino de programação** - survey realizado com o objetivo de compre-

ender os processos relacionados com a avaliação de códigos em disciplinas introdutórias de programação, os dados foram capturados através de um questionário online distribuído aos participantes;

- **Quasi experimento** - quasi experimento realizado com dois objetivos: investigar se é possível agrupar códigos de modo que um *feedback* objetivo (classificação/nota) seja generalizado; e avaliar se o *feedback* generalizado forneceria resultados semelhantes aos de avaliadores humanos;

- **Experimento baseado nos critérios informados pelos avaliadores especialistas** - experimento corresponde a continuidade do quasi-experimento descrito, de modo que restrições adicionais foram aplicadas, para adequar o rigor da pesquisa a um experimento. A investigação também foi ampliada para investigar se é possível agrupar códigos de modo que um *feedback* objetivo (classificação/nota) e um *feedback* subjetivo (informação textual) seja generalizado; e avaliar se o *feedback* generalizado forneceria resultados semelhantes aos de um avaliadores humanos. Além disso, as propriedades de agrupamento foram selecionadas com base na relação com os critérios informados pelos avaliadores especialistas;

- **Experimento baseado em seleção automática de atributos dos códigos** - experimento realizado sobre os mesmos cenários do experimento anterior, com a alteração dos meios de seleção dos atributos, os quais passaram a ser selecionados automaticamente e não mais com base na relação com os critérios informados pelos avaliadores especialistas.

4.3 Um *survey* sobre ensino de programação

O objetivo da condução do *survey* foi compreender os processos relacionados com a avaliação de códigos em disciplinas introdutórias de programação. Dessa forma, o *survey* se relaciona ao objetivo especifico relacionado com a identificação de quais critérios são considerados por cada avaliador.

A investigação também explorou a identificação de quais problemas, ou dificuldades,

poderiam estar relacionados ao ensino e aprendizagem de programação, sob a ótica do professor/monitor.

O questionário completo utilizado na condução do *survey* é apresentado no Apêndice A. O conjunto de critérios e os problemas, ou dificuldades, que poderiam estar relacionados ao ensino e aprendizagem de programação listados entre as opções apresentadas no questionário se baseia nas descrições exibidas nos artigos observados no levantamento bibliográfico conduzido ao longo do desenvolvimento desta teste.

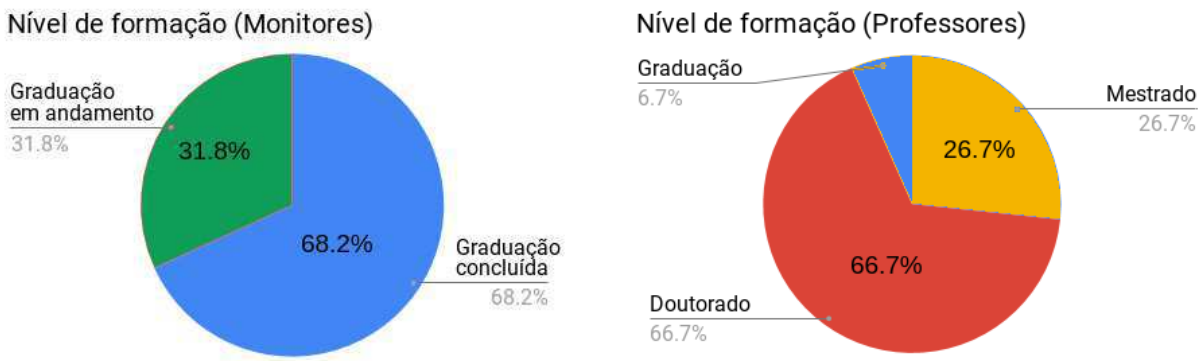
O questionário é composto pelas seguintes seções:

- **Cabeçalho** - um texto contendo a descrição da pesquisa e informações sobre confidencialidade e anonimato, apresentado no início da pesquisa;
- **Perfil** - onde são apresentadas perguntas relacionadas ao perfil geral do respondente;
- **Perfil - Professor** - onde são apresentadas perguntas relacionadas à experiência relacionada com a função de professor, apresentado somente aos participantes que são, ou já foram, professores;
- **Perfil - Monitor** - onde são apresentadas perguntas relacionadas à experiência relacionada com a função de monitor, apresentado somente aos participantes que são, ou já foram, monitores;
- **Identificação de problemas/dificuldades no contexto de ensino e aprendizagem de programação** - onde são apresentadas perguntas relacionadas aos possíveis fatores motivadores do cenário problemático relacionado ao ensino e aprendizagem de programação;
- **Processo de avaliação de códigos** - onde são apresentadas perguntas que buscam identificar os procedimentos e critérios adotados pelo avaliador, seja professor ou monitor, ao realizar avaliações de códigos;
- **Continuidade da pesquisa** - onde é solicitada a permissão do respondente para que o pesquisador entre em contato nas etapas posteriores da pesquisa.

Um total de 37 pessoas responderam o questionário, sendo 15 professores e 22 monitores/ex-monitores. Os sujeitos que responderam ao questionário estavam vinculados

às seguintes instituições: Universidade Federal de Alagoas(UFAL), Universidade Federal da Paraíba (UFPB), Universidade de São Paulo (USP), Universidade Federal de Minas Gerais (UFMG), Instituto Federal de Alagoas (IFAL), Instituto Federal de Sergipe (IFSE) e da Faculdade de Tecnologia (FATEC).

Em relação ao nível de formação, na Figura 4.5 são exibidos os gráficos relacionados ao nível de formação de cada categoria. Para os monitores, 68.2% possuíam graduação concluída e 31.8% estavam com a graduação em andamento. Para os professores, 66.7% possuíam doutorado, 26.7% possuíam mestrado e 6.7% possuíam graduação.



(a) Nível de formação dos monitores.

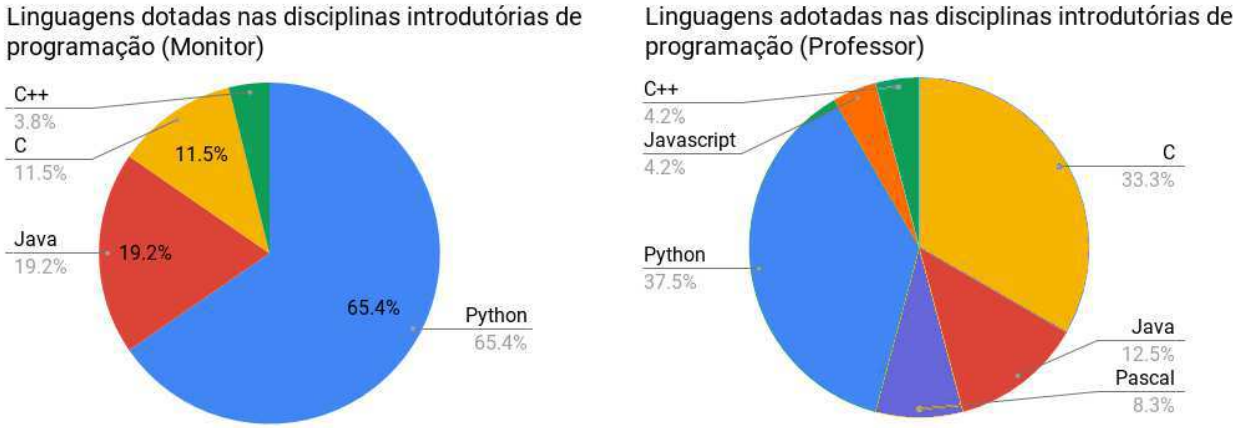
(b) Nível de formação dos professores.

Figura 4.5: Nível de formação das pessoas que responderam o questionário.

Considerando todas as pessoas, professores e monitores, elas desempenharam atividades em disciplinas introdutórias de programação pela última vez entre os anos 2013 e 2018. Tendo a maioria dos professores (46.7%) ministrado ao menos uma disciplina em 2017, e a maioria dos monitores (33.3%) tendo desempenhado monitoria em um disciplina no ano de 2018.

Entre as respostas fornecidas, a linguagem Python foi a mais utilizada, seja na condução da disciplina quando ministrada pelo professor, ou durante as atividades de monitoria. Entre os monitores, 65.4% desempenharam suas atividades em disciplinas onde a linguagem Python foi adotada. Entre os professores, 37.5% ministraram as disciplinas adotando a linguagem Python em suas turmas. Os dados são exibidos na Figura 4.6, onde é possível observar que as linguagens C, C++, Java, Javascript e Pascal também foram utilizadas.

Os dados do *survey* serviram como meio para determinar a linguagem e os avaliadores, com base no perfil desejado. Uma vez que a linguagem Python se mostrou como a mais



(a) Linguagens adotadas nas disciplinas quando desempenharam as atividades de monitoria.

(b) Linguagens adotadas na condução das disciplinas.

Figura 4.6: Linguagens adotadas nas disciplinas de programação introdutória.

utilizada entre as pessoas que responderam o questionário, este foi um dos fatos motivadores da adoção da linguagem nas etapas de pesquisa.

Todos os professores descreveram que foram adotadas atividades práticas nas disciplinas que ministraram, tendo a maioria adotado ferramentas de apoio (ex. Ides, juízes online, outras) na condução da disciplina. Somente um dos monitores descreveu que não foram utilizadas atividades práticas ou ferramentas de apoio na disciplina em que desempenhou as atividades de monitoria. Os demais monitores descreveram que foram adotadas atividades práticas e a maioria das disciplinas fez uso de ferramentas de apoio (ex. Ides, juízes online, outras).

Todos os professores consideram que existem problemas ou dificuldades relacionadas ao contexto de ensino e aprendizagem de programação. A opinião dos monitores é semelhante, tendo a grande maioria indicado que existem problemas ou dificuldades e somente três respondentes descreveram que talvez existam problemas ou dificuldades no contexto citado.

Na Figura 4.7, é exibido o gráfico listando as frequências com que os problemas apresentados no questionário foram indicados por parte dos professores e monitores. Além de indicar se considerava o item como um problema dentro do contexto era necessário fornecer o grau de importância relacionado ao problema.

Dentre os problemas ou dificuldades indicados pelos professores, o problema mais fre-

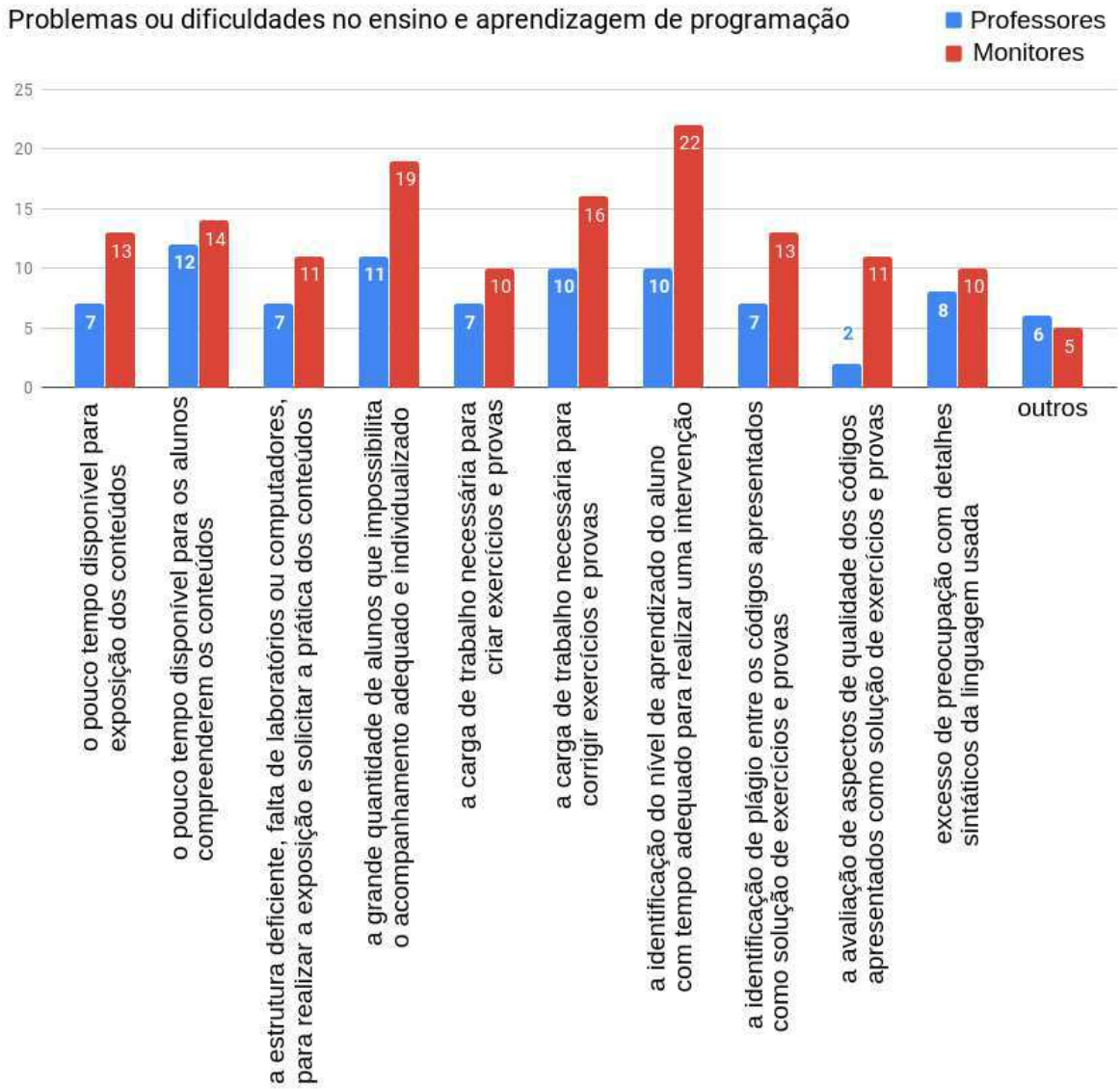


Figura 4.7: Problemas ou dificuldades existentes no ensino e aprendizagem de programação, sob a ótica dos professores e monitores.

quentemente indicado foi ‘o pouco tempo disponível para os alunos compreenderem os conteúdos’, este foi indicado por 12 professores. O problema ‘a grande quantidade de alunos que impossibilita o acompanhamento adequado e individualizado’ foi o segundo mais citado, sendo indicado por 11 professores. Observando o grau de importância relacionado ao problema, o problema ‘a grande quantidade de alunos que impossibilita o acompanhamento adequado e individualizado’ recebeu 6 indicações como o problema mais importante no contexto.

Outros problemas foram citados pelos professores de forma livre, alguns dos problemas adicionais descritos corroboram com um, ou mais, dos problemas listados e adicionam explicações. Outras descrições são parcialmente apresentadas a seguir:

“*A carga horária baixa da disciplina tende a deixar as aulas sempre no mesmo formato, variando apenas entre as exposições com datashow e quadro, e o uso do laboratório. Já tive experiências interessantes com gincanas de conhecimento e práticas de gamificação que nem sempre consigo repetir em turmas grandes.*”

(Professor 1)

“*A carga horária insuficiente para acompanhamento das dificuldades dos alunos ao fazerem os exercícios e para que os alunos façam os exercícios em sala, já que em horários extra-classe vários não se dedicam*”

(Professor 2)

“*Deixar mais amigáveis as informações de erro. Tentar indicar possíveis soluções usando IA. Muitos novatos se assustam a quantidade de erros e/ou com stacktrace do erro.*”

(Professor 3)

“*Problemas básicos de interpretação de texto, matemática básica, e português/inglês.*”

(Professor 4)

Um dos professores indicou que os problemas listados no questionário existem, mas não considera que estes sejam os mais relevantes. Este professor enumera as seguintes descrições como os problemas que ele acredita que sejam os mais relevantes no contexto de ensino e aprendizagem de programação.

"*(i) pouca identificação com a disciplina, achando-a desinteressante; (ii) pouco tempo que o aluno se dedica à prática; (iii) ansiedade demasiada, atrapalhando a solução de problemas que exigem um raciocínio mais encadeado. Ou seja, o aluno rapidamente fica impaciente; (iv) pouca resolução de problemas por parte dos docentes; (v) avaliações realizadas pelo docente com pouco nível de dificuldade, desestimulando o estudo de técnicas mais aprimoradas; (vi) ausência da adoção de material didático com um nível de simplicidade e de detalhes adequado para que o aluno possa aprender sozinho o que não entendeu durante a aula.*"

(Professor 5)

Dentre os problemas ou dificuldades indicados pelos monitores, o problema mais frequentemente indicado foi 'a identificação do nível de aprendizado do aluno com tempo adequado para realizar uma intervenção', tendo sido indicado por 22 monitores. O mesmo problema também foi apontado como o maior deste contexto, recebendo 8 indicações com maior grau de importância.

Outros problemas foram citados pelos monitores de forma livre, muitos destes descrevem o problema e sugerem abordagens. Algumas descrições são parcialmente apresentadas a seguir:

"*Falta de estratégias que sejam mais atrativas, do ponto de vista das atividades e projetos de programação. Poderiam ser criados projetos ou pequenos componentes de um software maior, e que esse software fosse realmente usado na prática, demonstrando assim aos futuros e atuais alunos da disciplina, que o aprendizado de programação é útil e não apenas para garantir nota ou curriculum.*"

(Monitor 1)

"*Alguns conceitos só são melhor assimilados ao final da disciplina, quando outros conceitos que são vistos separadamente ao longo do curso já foram apresentados. Isso torna difícil tanto a aprendizagem, quanto a avaliação do nível de compreensão do aluno ao longo da disciplina por parte do professor*"

(Monitor 2)

"*[...] métodos de avaliação inadequados, exemplos simples são apresentados nas aulas e no momento da avaliação são cobrados problemas complexos.*"

(Monitor 3)

Sendo os problemas 'o pouco tempo disponível para os alunos compreenderem os conteúdos', indicado pelos professores, e 'a identificação do nível de aprendizado do aluno com tempo adequado para realizar uma intervenção', indicado pelos monitores, aqueles que foram indicados com maior frequência. Sendo os problemas 'a grande quantidade de alunos que impossibilita o acompanhamento adequado e individualizado', indicado pelos professores, e 'a identificação do nível de aprendizado do aluno com tempo adequado para realizar uma intervenção', indicado pelos monitores, aqueles que foram interpretados como sendo os mais importantes no contexto. É possível relacionar estas indicações com a necessidade de prover *feedback* individualizado aos alunos, sem sobrecarregar o docente.

Uma vez que os professores consideram que a grande quantidade de alunos existente nas disciplinas de programação não permite realizar um acompanhamento de forma adequada e individualizada. Visto também que, os monitores consideram que a identificação do nível de aprendizado do aluno deve ocorrer de modo que seja possível realizar uma intervenção. A abordagem proposta nesta tese visa reduzir a quantidade de observações necessárias para realizar os processos de avaliação de códigos em exercícos práticos. Desta forma, a grande quantidade de alunos teria um menor impacto neste processo, o que permitiria ao professor/-monitor realizar observações mais detalhadas e consequentemente realizar uma intervenção de modo mais adequado (com mais informações ou detalhamento).

Para a realização das investigações, foi adotada uma categorização de problemas em três classes, sendo elas:

- **Problemas básicos**, são aqueles cujas soluções (códigos) necessitam apenas de um conjunto básico de construções (ex. cálculos matemáticos, entrada e saída, operações de comparação e atribuição);

- **Problemas de decisão**, são aqueles cujas soluções (códigos) necessitam de construções básicas e de estruturas de decisão/condicionais;

- **Problemas de repetição**, são aqueles cujas soluções (códigos) necessitam de construções básicas e de estruturas de repetição/iteração, podendo ou não necessitar de estruturas de decisão/condicionais.

Apresentando estas descrições, foi indagado quais, dentre os itens listados no questionário, são os critérios observados nas avaliações. O respondente poderia ainda citar outros critérios.

A seguir, na Figura 4.8, é apresentado o gráfico relacionado com a indicação dos critérios de avaliação para problemas classificados como básicos. São exibidos apenas os critérios citados com maior frequência por professores ou monitores.

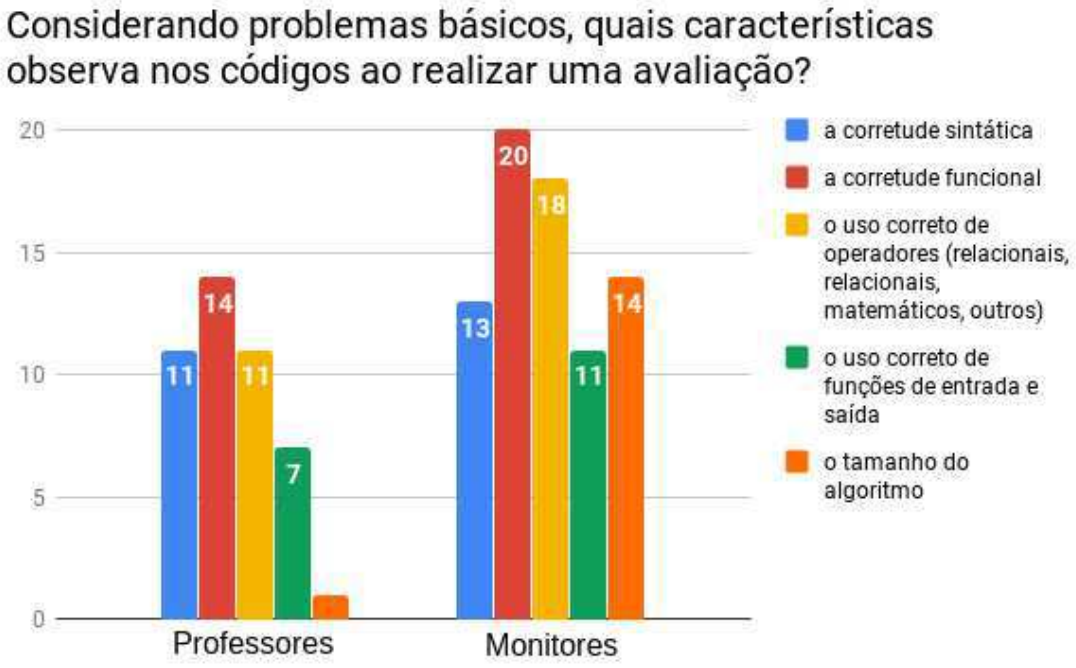


Figura 4.8: Características observadas nos códigos ao avaliar problemas 'Básicos'

Conforme é exibido no gráfico, para os professores 'a corretude funcional' representa o principal critério ao avaliar problemas do tipo 'Básico'. Sendo 'a corretude funcional' opção indicada por 14 professores. Os itens 'corretude sintática' e 'o uso correto de operadores (relacionais e matemáticos)' foram indicados como critérios utilizados por 11 professores, ficando em segundo lugar entre as indicações dos professores.

Do mesmo modo como ocorre com os professores, para os monitores 'a corretude funcional' representa o principal critério ao avaliar problemas do tipo 'Básico'. Sendo 'a corretude funcional' opção indicada por 20 monitores. O item 'o uso correto de operadores (relacionais e matemáticos)' foi indicado como critério utilizado por 18 monitores, ficando em segundo lugar entre as indicações dos monitores.

Outros critérios indicados pelos professores e monitores na avaliação de problemas 'Básicos' podem ser observados na Tabela A.1 exibida no Apêndice A.

A seguir, na Figura 4.9, é apresentado o gráfico relacionado com a indicação dos critérios de avaliação para problemas classificados como condicionais. São exibidos apenas os critérios citados com maior frequência por professores ou monitores.

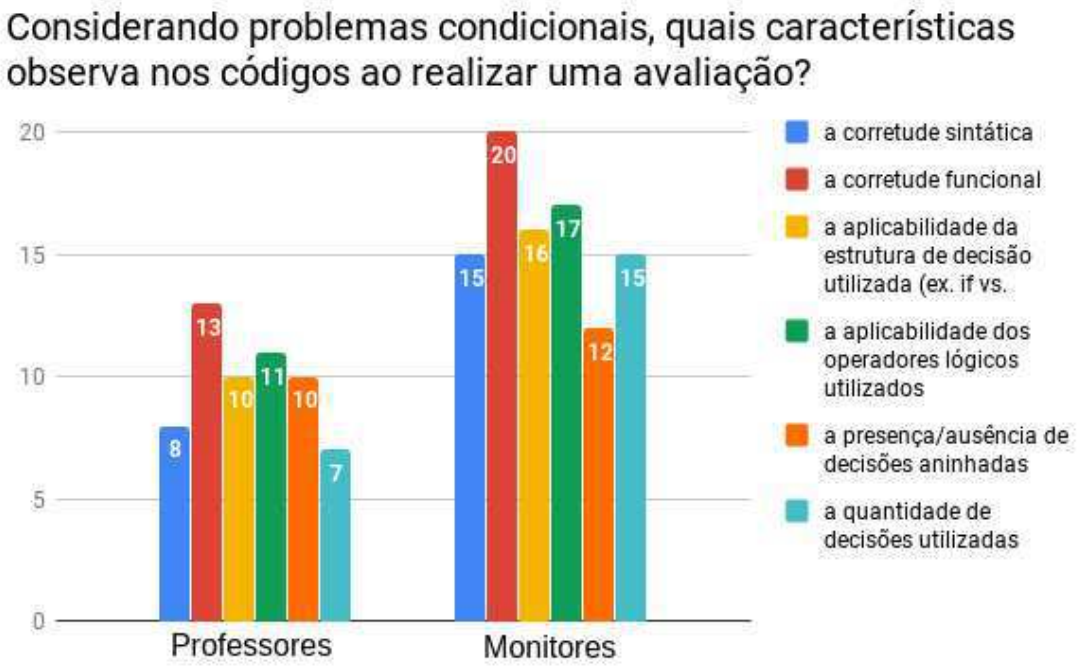


Figura 4.9: Características observadas nos códigos ao avaliar problemas 'Condicionais'

Conforme é exibido no gráfico, a maioria dos professores observa 'a corretude funcional' do código ao avaliar problemas do tipo 'Condicionais'. Sendo 'a corretude funcional' indicada como critério utilizado por 13 professores. O critério 'a aplicabilidade dos operadores lógicos utilizados', particular de problemas condicionais, foi indicado como critério utilizado por 11 professores. Sendo este, o segundo critério mais indicado entre os professores. Os critérios 'a aplicabilidade da estrutura de decisão utilizada (ex. if vs. switch)' e 'a presença/ausência de decisões aninhadas', particulares de problemas condicionais, foram indicados como critérios utilizados 10 professores cada um. Sendo estes, o terceiro critério mais indicado entre os professores.

Do mesmo modo como ocorre com os professores, para os monitores 'a corretude funcional' representa o principal critério ao avaliar problemas do tipo 'Condicionais'. Sendo

'a corretude funcional' indicada como critério utilizado por 20 monitores. O critério 'a aplicabilidade dos operadores lógicos utilizados', particular de problemas condicionais, foi indicado como critério utilizado por 17 monitores. Sendo este, o segundo critério mais indicado entre os monitores. O critério 'a aplicabilidade da estrutura de decisão utilizada (ex. if vs. switch)', particular de problemas condicionais, foi indicado como critério utilizado por 16 monitores. Sendo este, o terceiro critério mais indicado entre os monitores.

Outros critérios indicados pelos professores e monitores na avaliação de problemas 'Condicionais' podem ser observados na Tabela A.2 exibida no Apêndice A.

A seguir, na Figura 4.10, é apresentado o gráfico relacionado com a indicação dos critérios de avaliação para problemas classificados como repetição. São exibidos apenas os critérios citados com maior frequência por professores ou monitores.

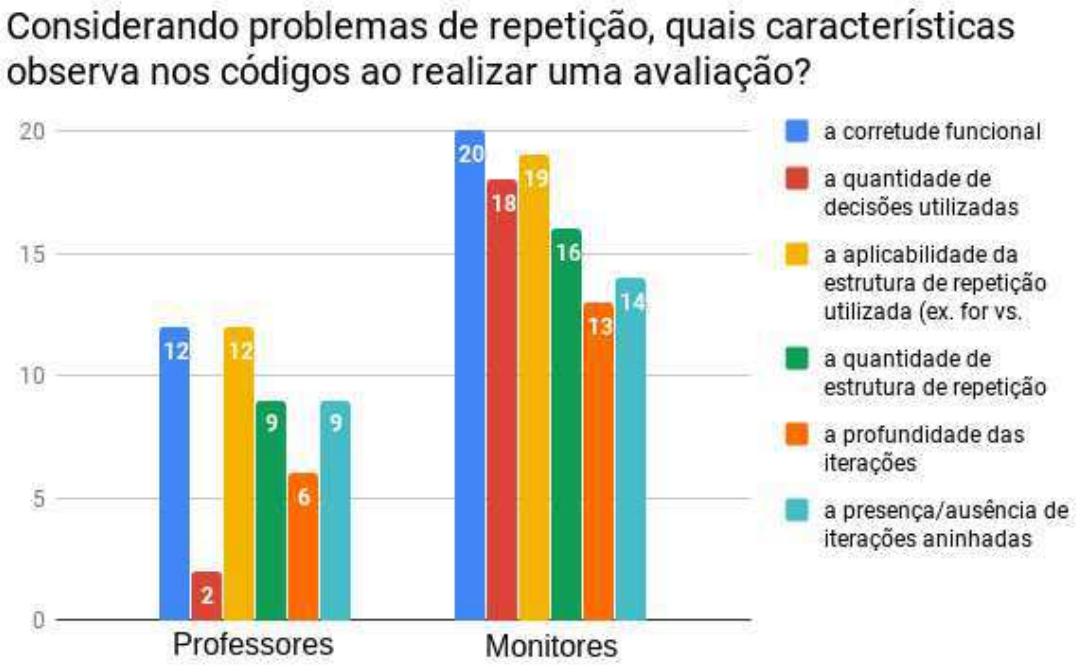


Figura 4.10: Características observadas nos códigos ao avaliar problemas 'Repetição'

Conforme é exibido no gráfico, a maioria dos professores observa os critérios 'a corretude funcional' e 'a aplicabilidade da estrutura de repetição (ex. for vs. while)' ao avaliar problemas do tipo 'Repetição'. Ambas as opções foram indicadas por 12 professores. Os critérios particulares de problemas de repetição, 'a quantidade de estrutura de repetição utilizadas' e 'a presença/ausência de iterações aninhadas', foram indicados como critérios utilizados por

9 professores.

Assim como para as classificações anteriores, conforme é exibido no gráfico a maioria dos monitores observa o critério 'a corretude funcional' ao avaliar problemas do tipo 'Repetição'. Esta opção foi indicada por 20 monitores. Os critérios particulares de problemas de repetição, 'a aplicabilidade da estrutura de repetição (ex. for vs. while)' e 'a quantidade de estrutura de repetição utilizadas', foram indicados por 19 e 18 monitores respectivamente.

Outros critérios indicados pelos professores e monitores na avaliação de problemas 'Repetição' podem ser observados na Tabela A.3 exibida no Apêndice A.

Observando os dados do levantamento, é possível concluir que o critério 'a corretude funcional' é o principal critério adotado por professores e monitores. Contudo, este não é o único critério que é levado em consideração quando um avaliador avalia um código. Desta forma, uma vez que a avaliação de um juiz online considera somente a corretude funcional das soluções, é necessário prover uma abordagem complementar que possibilite a avaliação de outros critérios adotados por um avaliador.

Os professores foram indagados sobre o processo de avaliação adotado. Observando as descrições, é possível notar que a execução do código, fornecendo entradas válidas e inválidas para observar as saídas produzidas é uma pratica comum. As descrições corroboram com a indicação do critério 'a corretude funcional' como o mais utilizado entre os docentes. Algumas descrições do processo de avaliação são apresentadas a seguir:

"*1) Avaliação do fluxo 'normal' da execução (o que é esperado está sendo executado);*
2) Avaliação do fluxo 'errado' da execução (o que não é esperado está sendo tratado);
3) Avaliação lógica do código (variáveis, condicionais, iterações, etc);
4) Avaliação geral do código (foi implementado com boas práticas de programação)."

(Professor 6)

"*Em geral, gosto de resolver o problema, definindo pesos para as partes do algoritmo e vou comparando as soluções dos alunos com a solução que fiz. Coloco observações sobre eficiência de uso de memória e processamento, mas não penalizo as soluções sob esses aspectos ou sobre o uso de outras estruturas desde que o problema proposto possa ser resolvido.*"

(Professor 7)

"*Tiro pouca pontuação (alguns décimos) de erros sintáticos. Se perceber que a ideia do algoritmo está compatível com uma solução correta, embora contenha erros mínimos, pontuo consideravelmente, penalizando apenas para o aluno perceber o que errou. Geralmente, penalizo a utilização incorreta de if-else, for e while.*"

(Professor 8)

"*Primeiro verifico se o código apresenta o comportamento (saída) esperado. Depois verifico a aplicabilidade das estruturas e comandos utilizados para solucionar o problema apresentado.*"

(Professor 9)

"*Basicamente, eu verifico se o estudante utilizou corretamente o construto esperado para aquela solução (e.g., operador aritmético, estrutura de seleção ou estrutura de repetição). Para mim, esse é o ponto mais importante. Após isso, eu verifico outras características como, por exemplo, indentação, nomes de variáveis, tamanho do código etc.*"

(Professor 10)

Os monitores foram indagados sobre o processo de avaliação adotado. Observando as descrições, é possível notar que, assim como ocorre com o processo de avaliações dos professores, a execução do código é bastante utilizada pelos monitores. Algumas respostas do processo de avaliação empregados pelos monitores são apresentadas a seguir:

"*Observo o código em busca de erros, depois executo o código utilizando alguns valores para testar os valores de saída.*"

(Monitor 4)

"*Identifico o problema e a solução de referência, logo verifico a solução, começando pela corretude sintática, depois a funcional, e então verifico os detalhes relacionados.*"

(Monitor 5)

"*Verifico a corretude funcional, utilizando diferentes entradas e verificando se a saída é a esperada, seguido pela leitura do código, atendando para a estrutura e buscando compreender a lógica utilizada para resolver o problema.*"

(Monitor 6)

"*Primeiramente eu respondo a questão, criando um código funcional com os conhecimentos utilizados durante aula. Depois, verifico se os resultados correspondem ao esperado. Posteriormente seriam feitas sugestões (podendo ou não descontar na pontuação).*"

(Monitor 7)

"*Verifico se para meus testes está dando a saída correta, depois verifico se há partes desnecessárias e ou se não há maneiras mais fáceis de fazer o mesmo.*"

(Monitor 8)

4.4 Um quasi experimento para avaliação da abordagem de agrupamentos adaptativos

Esta etapa de pesquisa consistiu na execução de um quasi experimento realizado com o uso de seis problemas aplicados em quatro turmas de cursos introdutórios de programação. Buscou-se investigar se as soluções apresentadas pelos alunos poderiam ser agrupadas de modo que um *feedback* objetivo (classificação/nota) pudesse ser generalizado. A generalização do *feedback* visava minimizar o esforço de avaliação dispendido pelo professor. Os *feedbacks* gerados com base no agrupamento foram comparados aos resultados fornecidos por avaliadores humanos. Esta comparação se baseou no grau de concordância obtido quando as listas de notas de dois avaliadores eram confrontadas.

4.4.1 Questões de pesquisa

As questões de pesquisa formuladas na pesquisa visam verificar de forma comparativa a concordância entre as avaliações fornecidas por um método baseado em agrupamento de código, que chamaremos de avaliador artificial, e a concordância entre as avaliações fornecidas por avaliadores humanos, que chamaremos de avaliadores.

A condução deste trabalho foi norteada para responder às seguintes questões de pesquisa: **QP1** - As avaliações fornecidas por um avaliador artificial são semelhantes às avaliações de um avaliador humano?

- H_0 **Hipótese nula**: A concordância entre as avaliações fornecidas ocorre ao acaso;
- H_1 **Hipótese alternativa**: A concordância entre as avaliações é superior àquela encontrada ao acaso.

QP2 - A concordância entre as avaliações fornecidas por um avaliador artificial e um avaliador é próxima à concordância entre as avaliações fornecidas por dois avaliadores?

- H_0 **Hipótese nula**: A concordância entre um avaliador artificial e um avaliador é igual à concordância entre dois avaliadores;
- H_{1A} **Hipótese alternativa A**: A concordância entre um avaliador artificial e um avaliador é inferior à concordância entre dois avaliadores;
- H_{1B} **Hipótese alternativa B**: A concordância entre um avaliador artificial e um avaliador é superior à concordância entre dois avaliadores.

As questões de pesquisa descritas se relacionam com os objetivos específicos onde se descreve que é necessário explorar se é possível agrupar códigos de modo que um *feedback* objetivo (classificação/nota) seja generalizado. Uma vez que o *feedback* objetivo é generalizado com a abordagem proposta, a avaliação do grau de concordância entre os avaliadores está relacionada com a adequação do que foi generalizado com o que é esperado por cada avaliador.

4.4.2 Descrição dos dados e cenários

A presente pesquisa foi realizada com base em dados de quatro turmas de cursos introdutórios de programação, localizadas em três instituições de ensino distintas, sendo elas a Universidade Federal de Alagoas - Campus Arapiraca (UFAL), a Universidade Federal da Paraíba - Campus IV (UFPB) e o Instituto Federal de Alagoas (IFAL). Os códigos observados foram submetidos durante disciplinas que ocorreram entre os anos de 2011 e 2014. Apenas códigos na linguagem *Python*, seja na versão 2 ou 3, foram alvo dos estudos conduzidos nesta pesquisa.

O conjunto de dados utilizado para a pesquisa é formado por um conjunto de problemas (exercícios) de programação, um conjunto de códigos submetidos por discentes como solu-

ções aos exercícios e um conjunto de avaliações fornecido por avaliadores, professores ou monitores/ex-monitores de disciplinas de programação.

Os exercícios utilizados correspondem a seis problemas contendo: um título, identificando o problema; um enunciado, descrevendo o que é solicitado para resolução do problema, e por vezes contextualizando o problema com uma história; um exemplo de entrada, descrevendo como foram fornecidos os valores de entrada nos testes; e um exemplo de saída, descrevendo como eram esperadas as saídas após a execução dos testes. A Tabela 4.4 ilustra a apresentação de um dos exercícios utilizados, o problema 'Aprovado', com os mesmos elementos que são apresentados no sistema *The Huxley*, e como foi exibido aos alunos durante as respectivas disciplinas e aos avaliadores durante o experimento. As descrições completas de todos os problemas utilizados podem ser encontradas no Apêndice B.

Tabela 4.4: Descrição do problema 'Aprovado'.

Problema: Aprovado
Enunciado: Faça um programa que leia três notas (valores reais) de um aluno, calcule sua média aritmética e imprima uma mensagem dizendo se o aluno foi aprovado, reprovado ou deverá fazer prova final. O critério de aprovação é o seguinte: Aprovado (média ≥ 7); Reprovado (média < 3); ou Prova final ($3 \leq$ média < 7).
Exemplo de entrada: 03 números reais separados por um final de linha.
Exemplo de saída: Uma mensagem que pode ser: "aprovado", "reprovado" ou "prova final"

Os códigos submetidos pelos discentes consistem em um total de 223 submissões de soluções propostas, ou de tentativas de soluções. Na Tabela 4.5, são exibidos os títulos, a quantidade de submissões por problema, a instituição onde a disciplina foi ofertada e uma classificação das questões de acordo com o conjunto de instruções utilizado nas submissões.

Para as questões classificadas com o tipo 'Básico', as instruções presentes em todas as submissões analisadas eram: $input$, $print$, $import$, funções matemáticas (ex. $sqrt$ e pow), operadores matemáticos (ex. $+, *, \%e/$) e operadores relacionais (ex. $=$ e $\leq$). Para as questões classificadas com o tipo 'Condicional', as instruções presentes em todas as submissões

Tabela 4.5: Lista de problemas com a respectiva quantidade de submissões, sua classificação e a instituição onde a disciplina foi ofertada.

	Título	**Total de submissões**	**Tipo**	**Instituição**
1	Salário com Bônus	32	Básico	UFAL
2	Distância entre 2 pontos	23	Básico	UFAL
3	Aprovado	43	Condicional	UFAL
4	Eleitor	40	Condicional	UFPB
5	*Loop* de ímpares	41	Repetição	IFAL
6	Divisível por 3	44	Repetição	UFAL

analisadas continham, além das instruções presentes no tipo 'Básico', estruturas de condição (ex. if e $else$) e operadores lógicos (ex. and e or). Para as questões classificadas com o tipo 'Repetição', as instruções presentes em todas as submissões analisadas continham, além das instruções presentes nos tipos 'Básico' e 'Condicional', estruturas de repetição (ex. for e $while$).

As avaliações foram fornecidas por 8 avaliadores, sendo 2 professores de uma instituição de ensino superior, 3 graduados que foram monitores de disciplinas de programação e 3 alunos de graduação que concluíram as disciplinas introdutórias de programação. Todos os avaliadores forneceram uma descrição de sua experiência em relação ao contexto de ensino e aprendizagem de programação, essa descrição é exibida na seção 4.4.3. Cada avaliador realizou os procedimentos de avaliação no tempo e ambiente em que julgou mais adequado. Para realizar uma avaliação era necessário fornecer:

- Uma descrição de quais critérios seriam utilizados para avaliar as submissões para aquele problema (como observado na Figura 4.11);
- Uma nota, no intervalo $[0; 10]$, para cada código submetido (como observado na Figura 4.12);
- Um *feedback* textual correspondendo ao que seria informado ao aluno autor do código (como observado na Figura 4.12);

- Uma observação, fornecida ao pesquisador, relatando dúvidas ou problemas relacionados com a avaliação de um código (como observado na Figura 4.12).

As avaliações foram realizadas através de um sistema implementado para facilitar a captura dos dados. Na Figura 4.11, é exibida a tela de descrição de um problema, nela o avaliador pode ver o enunciado, os exemplos de entrada e saída, a lista de códigos pendentes de avaliação e a lista dos código já avaliados. Os critérios que serão adotados nas avaliações dos códigos devem ser fornecidos nesta mesma tela.

Na Figura 4.12, é exibida a tela desenvolvida para coletar as avaliações, nela o avaliador deve fornecer uma nota, no intervalo $[0; 10]$, um *feedback* e uma observação, tal como descrito anteriormente.

4.4.3 Descrição dos avaliadores

Nesta seção, são exibidas descrições relacionadas ao perfil dos avaliadores. Para preservar a identidade de cada um dos avaliadores, a identificação deles será realizada através de valores inteiros, chamaremos estes de 'id'.

Algumas informações foram solicitadas aos avaliadores durante o cadastro no sistema de coleta de dados, estas solicitações são exibidas a seguir:

- Possui experiência como professor? Sim/Não;
- Nível de formação? Doutorado/Mestrado/Especialização/Graduação/Graduando;
- Possui experiência como monitor de disciplina? Sim/Não;
- Possui experiência como monitor de disciplina de programação? Sim/Não.

A seguir, são apresentados os perfis dos avaliadores, além da indicação do total de problemas avaliados de forma completa. Apenas avaliadores que concluíram a avaliação completa de ao menos um problema são apresentados.

O avaliador 'id:1' possui mestrado e experiência como professor, mas não desempenhou atividade de monitoria. Este avaliador realizou a avaliação completa das submissões para os 6 problemas solicitados.

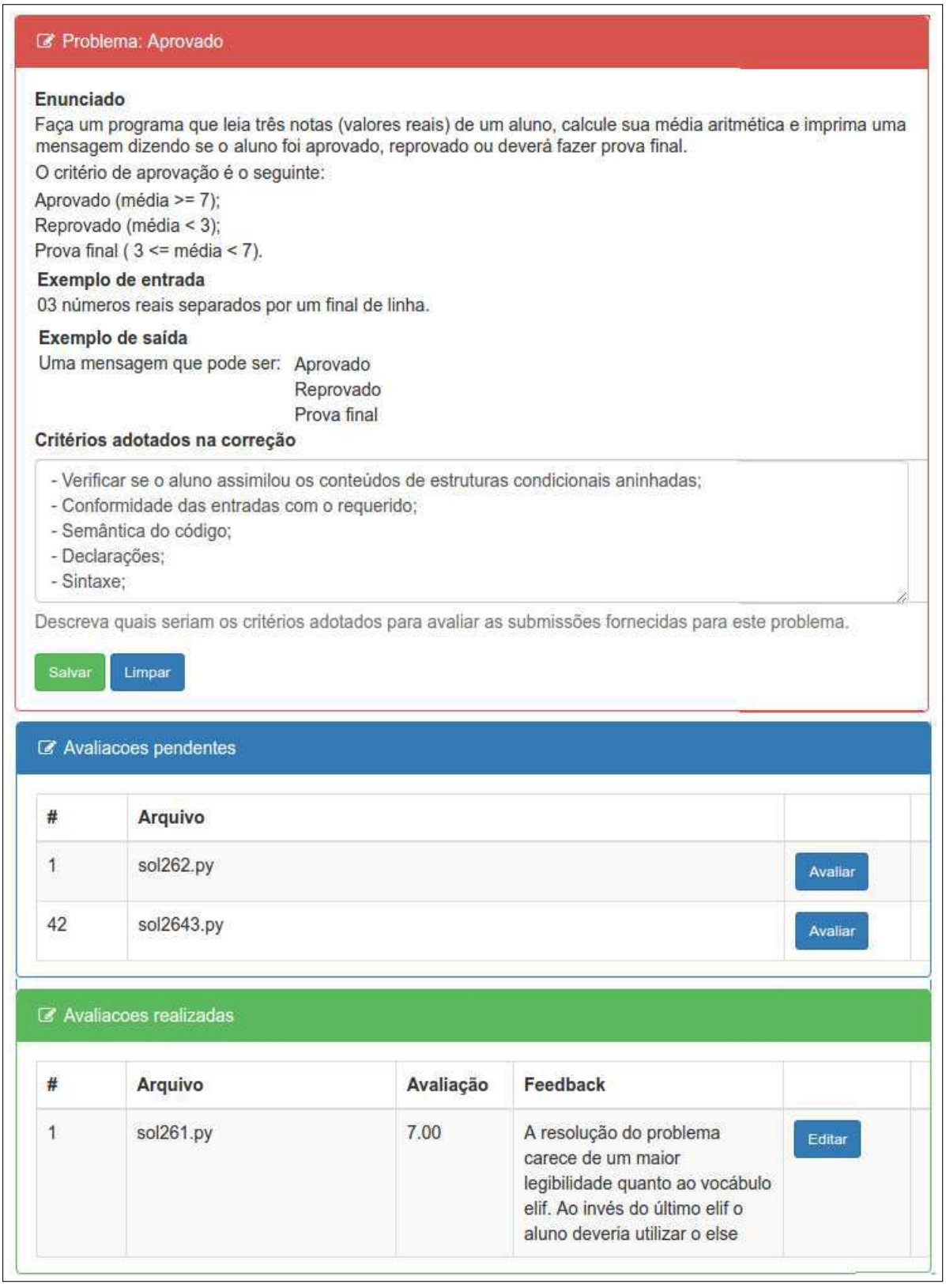


Figura 4.11: Tela de descrição de um problema como visualizado por um avaliador.

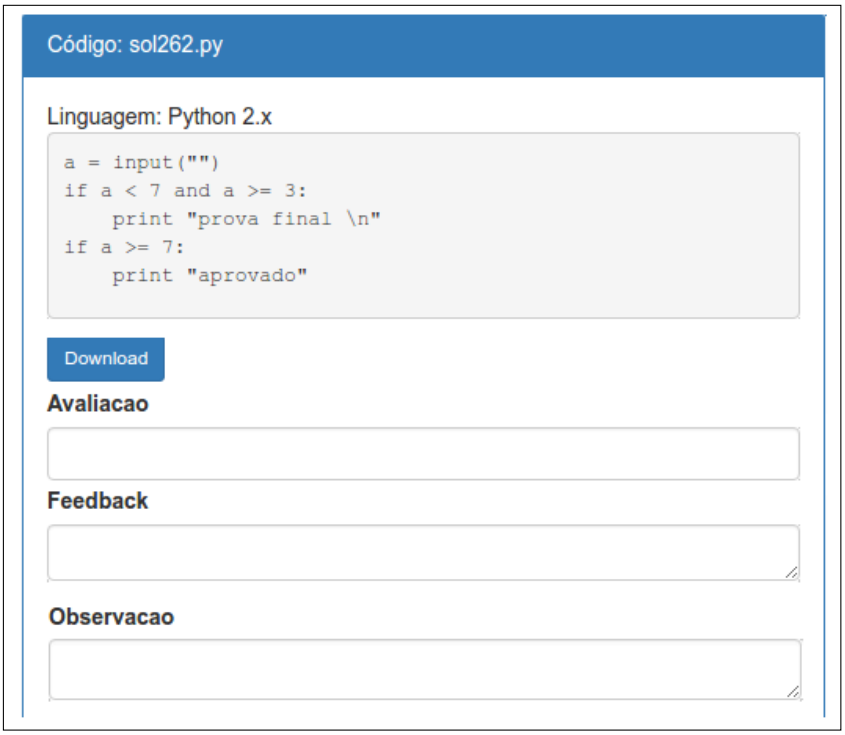


Figura 4.12: Tela de avaliação de um código.

O avaliador ‘id:5’ possui doutorado e experiência como professor, e desempenhou atividade de monitoria em ao menos uma disciplina de programação. Este avaliador realizou a avaliação completa das submissões para apenas 1 dos problemas solicitados.

Os avaliadores ‘id:6’,‘id:9 e’ ‘id:10’ possuem o mesmo perfil. Todos estes eram graduandos, portanto não possuem experiência como professor, e obtiveram uma boa média na disciplina introdutória de programação. Estes avaliadores realizaram a avaliação completa das submissões para os 6 problemas solicitados.

Os avaliadores ‘id:11’,‘id:12’ e ‘id:13’ também possuem o mesmo perfil. Todos estes eram graduados, sem experiência como professor e todos desempenharam atividade de monitoria em ao menos uma disciplina de programação. Os avaliadores ‘id:11’ e ‘id:12’ realizaram a avaliação completa das submissões para os 6 problemas solicitados. O avaliador ‘id:13’ realizou a avaliação completa das submissões para apenas 3 dos problemas solicitados.

4.4.4 Execução da abordagem

A condução do quasi experimento se restringiu a geração de um conjunto de notas (valores entre 0 e 10) e comparação do conjunto gerado com o fornecido por avaliadores (professores ou monitores/ex-monitores) de programação. O intuito desta investigação era verificar se uma avaliação fornecida por um método baseado em agrupamento de códigos forneceria resultados semelhantes aos de um avaliador humano.

Os avaliadores não foram alocados para avaliar as submissões dos problemas aleatoriamente, todas as submissões eram apresentadas na mesma ordem. As submissões foram selecionadas com base nas turmas para as quais um questionário, contendo o problema, tenha sido aplicado. A quantidade de submissões por problema é variável, dependendo de quantas submissões os alunos daquela turma realizaram, não foi aplicado nenhum critério de seleção para os códigos.

Inicialmente cada avaliador procedeu com a avaliação de dez códigos, sendo os mesmos códigos avaliados por todos e sendo estes apresentados na mesma ordem.

A execução do experimento foi realizada através da execução das etapas descritas na Seção 4.2.1. A extração das medidas sobre os códigos (Etapa 1) ocorreu logo após a seleção das turmas, o seguinte conjunto de métricas foi utilizado no quasi experimento:

- **Corretude funcional** - valor numérico no intervalo [0;100];
- **Complexidade ciclomática** - valor numérico inteiro positivo;
- **Número de operadores distintos** - valor numérico inteiro positivo ou nulo;
- **Número de operandos distintos** - valor numérico inteiro positivo ou nulo;
- **Similaridade baseada no coeficiente de Jaccard** - valor numérico no intervalo [0;1];
- **Similaridade baseada na distância de edição de texto** - valor numérico no intervalo [0;1];
- **Similaridade baseada na distância de edição de árvore** - valor numérico no intervalo [0;1].

Apenas o valor de similaridade em relação a uma solução de referência foi utilizado como entrada para o algoritmo de agrupamento, contudo foram computadas todas as similaridades entre todos os pares de código submetidos aos problemas. A determinação destas similaridades utilizou muitos recursos de processamento e memória.

A identificação das propriedades para agrupamento (Etapa 2), correspondendo a relação com os critérios adotados pelo avaliador, ocorreu sem o uso de nenhuma técnica de Inteligência Artificial. Desta forma, para identificar os critérios adotados por um avaliador, foram realizadas todas as combinações possíveis de medidas de software. O resultado destas combinações corresponde a 127 vetores de propriedades distintos para cada código.

A geração de grupos ou *clustering* (Etapa 3) foi realizada fazendo-se uso do algoritmo *Kmeans*. Foram realizadas execuções fornecendo $k = 5$ e outras utilizando $k = 10$. Os valores de k determinam a quantidade máxima de grupos que será criada pela execução do algoritmo. Desta forma, o valor de k também determina a quantidade de avaliações que serão realizadas antes da generalização dos *feedbacks*. Sendo assim, uma vez que a quantidade mínima de submissões para um problema correspondeu a 23 submissões, os valores $k = 5$ e $k = 10$ foram estabelecidos para que o esforço, quantidade de avaliações, fosse reduzido a metade.

As execuções do algoritmo para geração dos grupos tomaram cada um dos 127 vetores de propriedades gerados na etapa anterior. Dessa forma, foram criados 254 agrupametos diferentes, pois executou-se o algoritmo *kmeans* com $k = 5$ e $k = 10$ sobre os 127 vetores de propriedades gerados na etapa anterior.

Utilizando as avaliações realizadas no início do processo, buscou-se qual o agrupamento que melhor representava a distribuição de notas em relação aos grupos de cada agrupamento. O agrupamento que continha os grupos onde as notas possuiam as maiores concordâncias foi selecionado. Quando mais de um agrupamento demonstrava o mesmo grau de concordância, aquele que foi gerado com base na menor quantidade de métricas era selecionado. Uma vez que o agrupamento foi determinado e tendo as avaliações inciais sido realizadas, as notas para os códigos ainda não avaliados eram generalizadas com base no grupo em que este código pertencia. Essa generalização consistiu na atribuição de uma mesma nota para todos os elementos de um mesmo grupo.

4.4.5 Avaliação da abordagem

Para medir a concordância e o grau de proximidade existente entre as listas de avaliações fornecidas pelo avaliador artificial e pelos avaliadores, foram observadas as seguintes medidas:

- ***Kappa de Cohen*** - captura o grau de concordância entre duas listas de avaliações;
- ***Correlação de Pearson*** - captura o grau de correlação entre avaliações. Uma vez que uma concordância é computada apenas quando um valor exatamente igual é atribuído em ambas as listas, a correlação demonstra se as listas de avaliação possuem variações similares;
- **Distância entre notas** - considerando que as notas são fornecidas em um intervalo numérico, um somatório das diferenças de cada nota existente nas listas de avaliação foi computada.

4.4.6 Resultados e discussão

Os resultados obtidos são apresentados em um resumo geral, resultados obtidos para cada problema são apresentados no Apêndice C. Após exibir os resultados gerais, as perguntas de pesquisa serão respondidas.

Após computar os resultados de forma individual para os problemas, realizamos uma análise geral dos dados, utilizando as mesmas medidas estatísticas e de comparação, com o intuito de responder a questão de pesquisa 'QP1 - A avaliação fornecida por um método baseado em agrupamento de códigos é semelhante a avaliação de um avaliador?'. Os resultados gerais obtidos são apresentados na Tabela 4.6.

Tabela 4.6: Resultados obtidos para avaliação geral dos problemas

Kappa de Cohen			**Correl. de Pearson**			**Dist. total**		
Máx.	Mín.	Méd.	Máx.	Mín.	Méd.	Máx.	Min.	Méd.
1.00	0.39	0.76	1.00	0.70	0.92	57.50	0.00	28.88

Sendo assim, sobre a QP1, considerando a média, obteve-se uma concordância substancial entre a avaliação fornecida pelo avaliador artificial e a avaliação dos avaliadores. Este

resultado é fortalecido pelos valores obtidos em relação às outras medidas. Ocorre, considerando a média, uma correlação muito forte, ou seja, existe uma relação linear crescente entre as avaliações fornecidas. Na distância de notas, para as 223 avaliações, ocorreu diferença média de 28.88 pontos em relação às notas, essa diferença seria próxima a um décimo por avaliação. Observados estes resultados, pode ser sugerido que a avaliação baseada nos agrupamentos fornece uma avaliação semelhante a dos avaliadores. Ainda é possível descrever que, em média, a concordância de avaliação entre avaliador artificial e avaliadores humanos é superior a média de concordância de avaliação entre avaliadores humanos.

As concordâncias entre todos os pares de avaliadores, bem como a concordância de cada um destes em relação ao avaliador artificial, é apresentada na Tabela 4.7. O gráfico exibido na Figura 4.13 contém a representação em *boxplots* das concordâncias obtidas pelo avaliador artificial em relação aos avaliadores e das concordâncias gerais para cada avaliador em relação aos seus pares considerando todas as avaliações realizadas.

Tabela 4.7: Concordâncias entre todos os pares de avaliadores humanos, e de cada avaliador humano com o avaliador artificial considerando todas as avaliações.

Kappa de Cohen								
Avaliador	**id:1**	**id:5**	**id:6**	**id:9**	**id:10**	**id:11**	**id:12**	**id:13**
id:1	- -	0.08	0.17	0.25	0.30	0.27	0.04	0.25
id:5	0.08	- -	0.03	0.05	0.01	-0.07	0.05	-0.01
id:6	0.17	0.03	- -	0.15	0.09	0.12	-0.03	0.10
id:9	0.25	0.05	0.15	- -	0.45	0.36	0.08	0.39
id:10	0.30	0.00	0.09	0.45	- -	0.31	0.08	0.38
id:11	0.27	-0.07	0.12	0.36	0.31	- -	0.16	0.29
id:12	0.04	0.05	-0.03	0.08	0.08	0.16	- -	0.09
id:13	0.25	-0.01	0.10	0.39	0.38	0.29	0.09	- -
Média	**0.19**	**0.03**	**0.09**	**0.25**	**0.23**	**0.21**	**0.07**	**0.21**
Avaliador artificial	**0.74**	**0.76**	**0.70**	**0.83**	**0.78**	**0.70**	**0.79**	**0.78**

Observando os dados relacionados com as concordâncias entre avaliadores e seus pares, Tabela 4.7, é possível notar que o valor máximo de concordância obtido entre um par de

avaliadores corresponde apenas a uma concordância moderada ($kappa = 0.45$). A maior média de concordância obtida entre avaliadores corresponde apenas a uma concordância razoável ($kappa = 0.25$).

Observando os dados relacionados com as concordâncias entre cada avaliador e o avaliador artificial, Tabela 4.7, considerando a média geral por avaliador, no pior caso foi obtida uma concordância substancial ($kappa = 0.70$).

A visualização das informações pode ser facilitada pelo gráfico exibido na Figura 4.13, onde é possível identificar o valor máximo das concordâncias obtidas entre avaliadores e avaliador artificial em relação a qualquer uma das concordâncias entre um avaliador e seus pares. Contudo, uma vez que existe interseção entre os *boxplots*, não é possível afirmar que a concordância do avaliador artificial em relação aos avaliadores é superior, em comparação àquela existente entre cada avaliador em relação aos seus pares.

Sendo assim, sobre a QP2, considerando a média, obteve-se uma concordância substancial ($kappa = 0.76$) entre a avaliação baseada nos agrupamentos e a avaliação dos avaliadores, enquanto a concordância máxima obtida entre avaliadores foi apenas uma concordância moderada ($kappa = 0.45$). Assim, pode ser sugerido que a avaliação do avaliador artificial fornece uma concordância superior àquela obtida entre dois avaliadores.

Contudo, vale salientar que uma concordância só é computada quando exatamente o mesmo valor de nota é fornecido para uma avaliação. Então, para casos em que, por exemplo, um avaliador tenha fornecido uma nota 9.00 (nove inteiros) e outro avaliador tenha fornecido uma nota 9.50 (nove inteiros e meio), apesar das avaliações serem próximas, elas não são computadas como uma concordância. Para verificar a distância entre as avaliações dos avaliadores, faz-se necessária a observação de outras medidas como critério.

Sendo assim, foram computados e serão apresentados a seguir os resultados para as medidas, correlação de Pearson e distância total de notas. Estes dados são exibidos nas tabelas 4.8 e 4.9.

Na Tabela 4.8, são exibidos os valores de correlação entre as avaliações de cada avaliador e seus pares considerando todas as avaliações. É possível notar que, considerando a média geral por avaliador, no pior caso foi obtida uma correlação moderada (0.50) e no melhor caso, é obtida uma correlação forte (0.79) entre os avaliadores. Observando a média geral, foi obtida uma correlação forte (0.71), isso sugere que as variações nas avaliações seguem

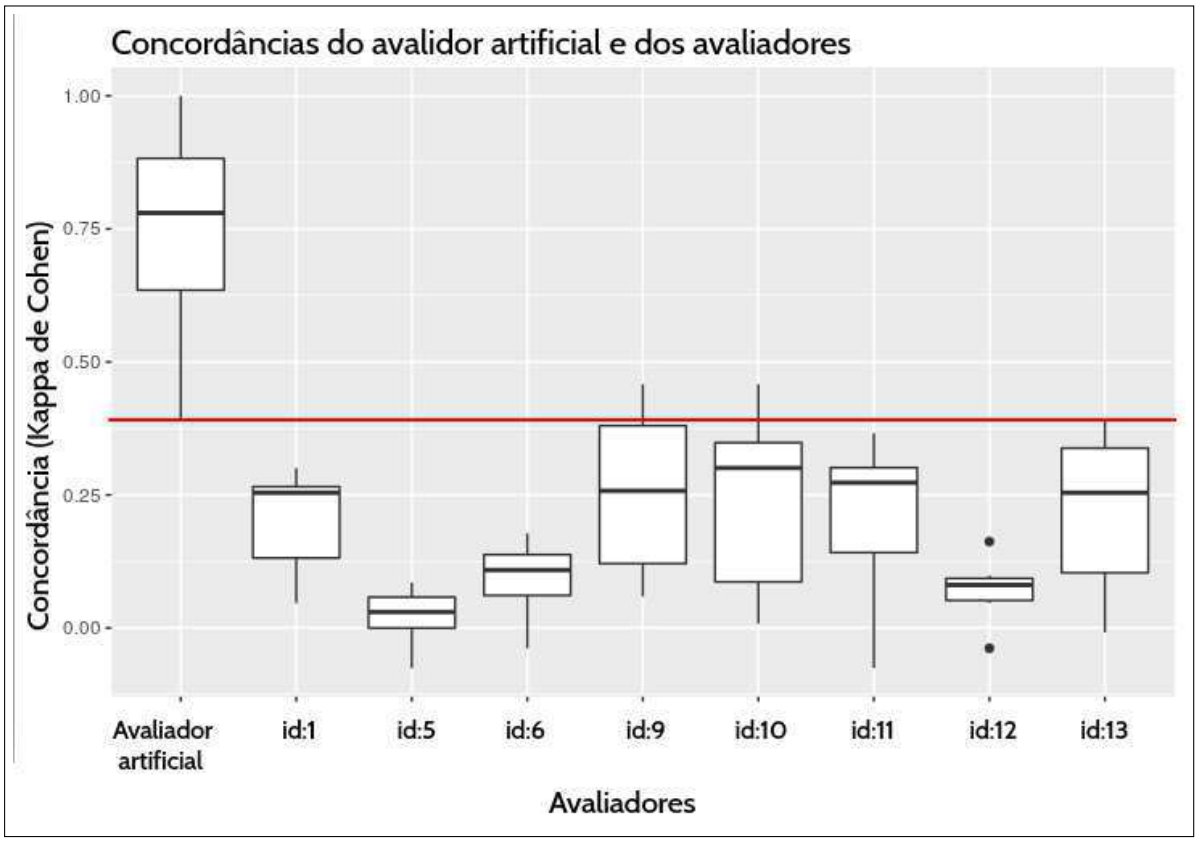


Figura 4.13: Concordâncias obtidas pelo avaliador artificial em relação aos avaliadores e das concordâncias gerais para cada avaliador em relação aos seus pares considerando todas as avaliações realizadas.

uma mesma tendência.

A visualização destas informações pode ser facilitada pelo gráfico exibido na Figura 4.14, onde é exibido o *boxplot* para as correlações entre as avaliações fornecidas.

Na Tabela 4.9, são exibidos os valores de distância de notas entre as avaliações de cada avaliador e seus pares considerando todas as avaliações. Vale observar que os avaliadores 'id:5' e 'id:12' não realizaram avaliações para todas as 223 submissões, tendo 'id:5' avaliado 23 submissões e 'id:12' avaliado 107 submissões, por isso as distâncias obtidas são mais baixas para estes avaliadores. Observando a média geral, foi obtida uma distância de 228.89 pontos, considerando este valor como o somatório das diferenças das notas, no universo de 223 submissões, tem-se em média uma distância de 1.29 pontos por avaliação. Desta forma, podemos descrever que a diferença entre as avaliações fornecidas entre os avaliadores

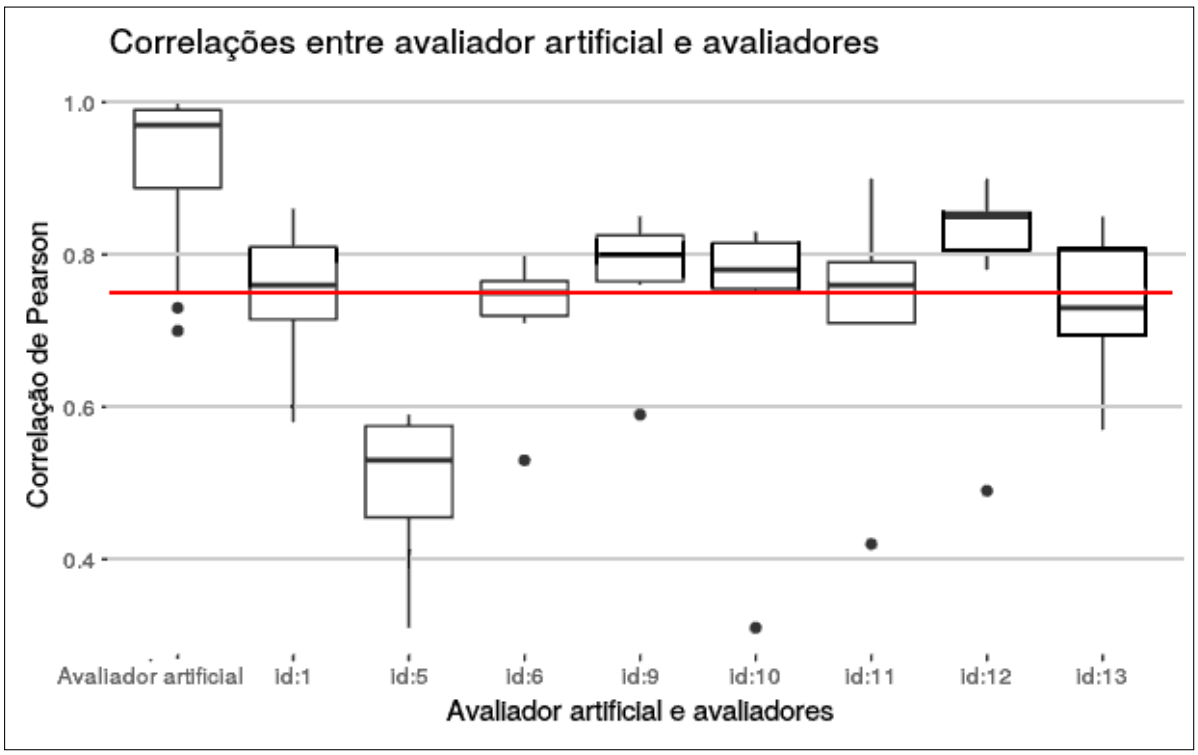


Figura 4.14: Correlações obtidas pelo avaliador artificial em relação aos avaliadores e das correlações gerais para cada avaliador em relação aos seus pares considerando todas as avaliações realizadas.

Tabela 4.8: Correlação entre as avaliações de cada avaliador e seus pares considerando todas as avaliações.

Correlação de Pearson								
Avaliador	**id:1**	**id:5**	**id:6**	**id:9**	**id:10**	**id:11**	**id:12**	**id:13**
id:1	- -	0.58	0.75	0.76	0.81	0.81	0.86	0.68
id:5	0.58	- -	0.53	0.59	0.31	0.42	0.49	0.57
id:6	0.75	0.53	- -	0.80	0.75	0.71	0.78	0.73
id:9	0.76	0.59	0.80	- -	0.82	0.77	0.85	0.83
id:10	0.81	0.31	0.75	0.82	- -	0.76	0.83	0.78
id:11	0.81	0.42	0.71	0.77	0.76	- -	0.90	0.71
id:12	0.86	0.49	0.78	0.85	0.83	0.90	- -	0.85
id:13	0.68	0.57	0.73	0.83	0.78	0.71	0.85	- -
Média	**0.75**	**0.50**	**0.72**	**0.77**	**0.72**	**0.72**	**0.79**	**0.73**

humanos não é muito grande.

A visualização destas informações pode ser facilitada pelo gráfico exibido na Figura 4.15, onde é exibido o *boxplot* para as distâncias, em relação a todas as avaliações realizadas.

4.4.7 Ameaças à validade

A condução do quasi experimento visa um processo de coleta e análise imparcial. As questões de pesquisa e as hipóteses foram estabelecidas antes do início da análise. As respostas apresentadas se baseiam nos resultados obtidos.

Os avaliadores que participaram do experimento possuem diferentes níveis de experiência em relação ao contexto de avaliação em disciplinas introdutórias de programação. Desta forma, se faz presente a ameaça à validade de constructo, relacionada com um constructo e seus níveis, neste caso a experiência dos sujeitos participantes do quasi experimento. Com isso, pode se descrever que os resultados apresentados podem ser diferentes se os sujeitos apresentarem níveis semelhantes. No contexto deste quasi experimento, pode ocorrer uma maior concordância entre os avaliadores humanos, por exemplo.

Ameaças à validade externa também se apresentam neste quasi experimento, ou seja,

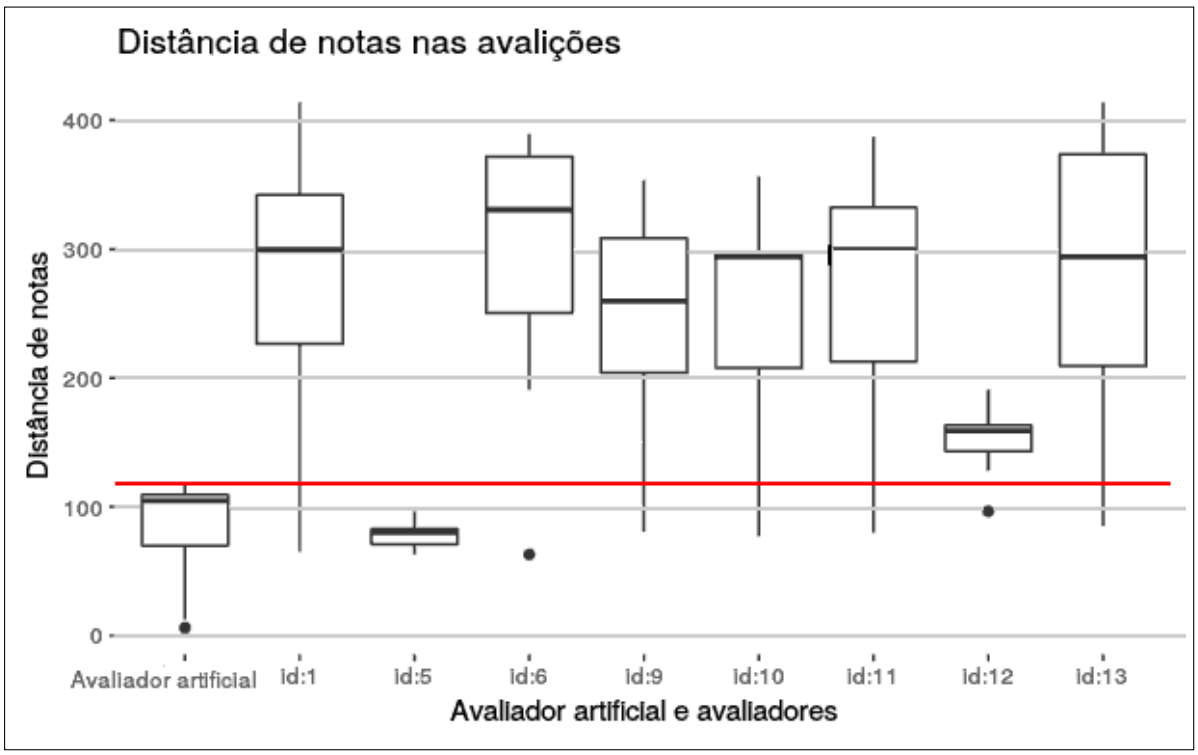


Figura 4.15: Distâncias de nota obtidas pelo avaliador artificial em relação aos avaliadores e das distâncias de nota para cada avaliador em relação aos seus pares considerando todas as avaliações realizadas.

Tabela 4.9: Distância de notas entre as avaliações de cada avaliador e seus pares considerando todas as avaliações.

Distância de notas								
Avaliador	**id:1**	**id:5**	**id:6**	**id:9**	**id:10**	**id:11**	**id:12**	**id:13**
id:1	- -	65.0	330.5	353.5	295.0	299.5	158.0	414.0
id:5	65.0	- -	63.0	80.5	77.0	80.0	96.5	85.0
id:6	330.5	63.0	- -	310.0	356.5	387.0	191.0	389.5
id:9	353.5	80.5	310.0	- -	248.5	307.0	159.5	259.5
id:10	295.0	77.0	356.5	248.5	- -	297.5	167.0	294.0
id:11	299.5	80.0	387.0	307.0	297.5	- -	128.0	357.5
id:12	158.0	96.5	191.0	159.5	167.0	128.0	- -	159.0
id:13	414.0	85.0	389.5	259.5	294.0	357.5	159.0	- -
Média	**273.64**	**78.14**	**289.64**	**245.5**	**247.92**	**265.21**	**151.28**	**279.78**

os resultados exibidos podem não ser válidos para uma população mais geral. Uma vez que apenas oito avaliadores participaram do experimento, e que o universo de avaliadores é bastante vasto para o contexto de programação, é possível que se a abordagem apresentada for avaliada para um conjunto diferente de avaliadores, os resultados não sejam semelhantes. Apenas códigos em *Python* foram observados, códigos escritos em outra linguagem podem alterar os valores das medidas e, consequentemente, afetar a forma de agrupamentos, sendo assim resultados distintos daqueles aqui exibidos podem ser gerados.

4.5 Um experimento para avaliação da abordagem de agrupamentos adaptativos

Esta etapa de pesquisa consistiu na execução de um experimento realizado com o uso de seis problemas aplicados em diversas turmas de cursos introdutórios de programação. As soluções foram selecionadas de maneira aleatória entre os códigos submetidos em todas as turmas existentes no repositório de dados.

Neste experimento, buscou-se investigar se as soluções apresentadas pelos alunos pode-

riam ser agrupadas de modo que um *feedback* objetivo (classificação/nota) e um *feedback* subjetivo (informação textual) pudessem ser generalizados. A generalização dos *feedbacks* visava minimizar o esforço de avaliação dispendido pelo professor.

Os *feedbacks* gerados com base no agrupamento foram comparados aos resultados fornecidos por avaliadores humanos. Esta comparação se baseou no grau de concordância obtido quando as listas de notas de dois avaliadores eram confrontadas, e na indicação do nível de adequação dos *feedbacks* textuais gerados. Os avaliadores foram selecionados com base nas informações coletadas no survey, descrito na Seção 4.5.3.

4.5.1 Questões de pesquisa

As questões de pesquisa formuladas no quasi experimento foram mantidas para o experimento. A investigação visa investigar de forma comparativa a concordância entre as avaliações fornecidas por um método baseado em agrupamento de código, que chamaremos de avaliador artificial, e a concordância entre as avaliações fornecidas por avaliadores humanos, que chamaremos de avaliadores.

A condução deste trabalho foi norteada para responder às seguintes questões de pesquisa: **QP1** - As avaliações (notas) fornecidas por um avaliador artificial são semelhantes as avaliações (notas) de um avaliador?

- H_0 **Hipótese nula**: A concordância entre as avaliações (notas) fornecidas ocorre ao acaso;
- H_1 **Hipótese alternativa**: A concordância entre as avaliações (notas) é superior àquela encontrada ao acaso.

QP2 - A concordância entre as avaliações (notas) fornecidas por um avaliador artificial e um avaliador é próxima à concordância entre as avaliações (notas) fornecidas por dois avaliadores?

- H_0 **Hipótese nula**: A concordância entre um avaliador artificial e um avaliador é igual à concordância entre dois avaliadores;
- H_{1A} **Hipótese alternativa A**: A concordância entre um avaliador artificial e um avaliador é inferior à concordância entre dois avaliadores;

- H_{1B} **Hipótese alternativa B**: A concordância entre um avaliador artificial e um avaliador é superior à concordância entre dois avaliadores.

QP3 - O *feedback* textual generalizado pelo avaliador artificial e um avaliador é adequado como avaliação para as submissões?

- H_A **Hipótese A**: O *feedback* textual generalizado NÃO é adequado;
- H_B **Hipótese B**: O *feedback* textual generalizado é PARCIALMENTE adequado;
- H_C **Hipótese C**: O *feedback* textual generalizado é TOTALMENTE adequado.

As questões de pesquisa descritas se relacionam com os objetivos específicos onde se descreve que é necessário explorar se é possível agrupar códigos de modo que um *feedback* objetivo (classificação/nota) e subjetivo (informação textual) sejam generalizados. Uma vez que os *feedbacks* são generalizados com base na abordagem proposta, a avaliação do grau de concordância entre os avaliadores está relacionada com a adequação do que foi generalizado com o que é esperado por cada avaliador.

4.5.2 Descrição dos dados e cenários

O conjunto de dados utilizado para a pesquisa é formado por um conjunto de problemas (exercícios) de programação, um conjunto de códigos submetidos por discentes como soluções aos exercícios, e um conjunto de avaliações fornecido por avaliadores, professores ou monitores/ex-monitores de disciplinas de programação.

Os exercícios utilizados correspondem a seis problemas contendo: um título, identificando o problema; um enunciado, descrevendo o que é solicitado para resolução do problema, e por vezes contextualizando o problema com uma história; um exemplo de entrada, descrevendo como foram fornecidos os valores de entrada nos testes; e um exemplo de saída, descrevendo como eram esperadas as saídas após a execução dos testes. A Tabela 4.10 ilustra a apresentação de um dos exercícios utilizados, o problema 'Consumo', com os mesmos elementos que são apresentados no sistema *The Huxley*, e como foi exibido aos alunos durante as respectivas disciplinas e aos avaliadores durante o experimento. As descrições completas de todos os problemas utilizados pode ser encontrada no Apêndice D.

Tabela 4.10: Descrição do problema 'Consumo'.

Problema: Consumo
Enunciado: Calcule o consumo médio de um automóvel sendo fornecidas a distância total percorrida (em Km) e o total de combustível gasto (em litros).
Exemplo de entrada: A entrada contém dois valores: um valor inteiro X representando a distância total percorrida (em Km) e um valor real Y representando o total de combustível gasto.
Exemplo de saída: Apresente o valor que representa o consumo médio do automóvel com 3 casas após a vírgula, seguido da mensagem "km/l".

Os seis problemas foram selecionados para o experimento dentre os 1800 problemas disponíveis no repositório. Alguns critérios de corte foram aplicados aos problemas existentes, sendo mantidos apenas problemas que tivessem:

- no mínimo 50 submissões em Python (2 ou 3), uma vez que é necessária uma quantidade mínima de submissões para a execução da abordagem;
- no máximo 2000 caracteres em seu enunciado, para não forçar o avaliador, voluntário no experimento, a realizar um processo de leitura e compressão muito demorado;
- tivessem sido utilizados em pelo menos 10 listas de exercícios, pois era desejado que um problema fosse bem utilizado no contexto investigado, sendo a quantidade de vezes que um problema foi utilizado em uma lista de exercícios um fator utilizado na seleção dos problemas.

Para todos os problemas a mesma quantidade de códigos foi selecionada. O experimento utilizou códigos selecionados aleatoriamente do repositório de dados do ambiente *The Huxley*. Desta forma, os códigos podem ter sido desenvolvidos por discentes matriculados em qualquer uma das mais de 300 turmas cadastradas no sistema. Apenas códigos na linguagem *Python*, seja na versão 2 ou 3, foram alvo dos estudos conduzidos nesta pesquisa.

Para cada avaliador, foram solicitadas avaliações para dez códigos de cada problema e para outros dez códigos, foram solicitadas confirmações de notas e confirmações de *feedbacks* textuais. Desta forma, os códigos submetidos pelos discentes consistem em um total

de 60 soluções, ou de tentativas de soluções propostas. Na Tabela 4.11, são exibidos os títulos, a quantidade de submissões por problema e a classificação das questões de acordo com o conjunto de instruções utilizado nas submissões.

Tabela 4.11: Lista de problemas, quantidade de submissões e classificação.

	Título	**Total de submissões**	**Tipo**
1	Fahrenheit para Celsius	20	Básico
2	Consumo	20	Básico
3	Classificação de Triângulos	20	Condicional
4	3 Números em Ordem Crescente	20	Condicional
5	Maior Múltiplo	20	Repetição
6	Fatorial	20	Repetição

A mesma forma de classificação adotada no quasi experimento foi mantida. Para as questões classificadas com o tipo 'Básico', as instruções presentes em todas as submissões analisadas eram: $input$, $print$, $import$, funções matemáticas (ex. $sqrt$ e pow), operadores matemáticos (ex. $+, *, \%e/$) e operadores relacionais (ex. $=$ e $\leq$). Para as questões classificadas com o tipo 'Condicional', as instruções presentes em todas as submissões analisadas continham, além das instruções presentes no tipo 'Básico', estruturas de condição (ex. if e $else$) e operadores lógicos (ex. and e or). Para as questões classificadas com o tipo 'Repetição', as instruções presentes em todas as submissões analisadas continham, além das instruções presentes nos tipos 'Básico' e 'Condicional', estruturas de repetição (ex. for e $while$).

As avaliações foram fornecidas por 12 avaliadores, sendo 3 professores de instituições de ensino superior ou técnico e 9 graduados, ou graduandos, que foram monitores de disciplinas introdutórias de programação.

Todos os avaliadores participaram do *survey*, a descrição do perfil de cada um destes é apresentada na seção 4.5.3. Somente avaliadores que ministraram disciplinas introdutórias de programação utilizando a linguagem Python (2 ou 3) na condução da disciplina foram selecionados. Somente monitores/ex-monitores que realizaram as atividades de monitoria em disciplinas introdutórias de programação que utilizaram a linguagem Python (2 ou 3) foram

selecionados. Os professores ou monitores/ex-monitores selecionados para o experimento serão chamados de avaliadores especialistas.

Cada avaliador especialista realizou os procedimentos de avaliação fornecendo:

- A seleção de critérios que seriam utilizados para avaliar as submissões para aquele problema (como observado na Figura 4.16);
- Uma nota, no intervalo $[0; 10]$, para cada código submetido (como observado na Figura 4.17);
- Um *feedback* textual correspondendo ao que seria informado ao aluno autor do código (como observado na Figura 4.17);
- Uma observação, fornecida ao pesquisador, relatando dúvidas ou problemas relacionados com a avaliação de um código (como observado na Figura 4.17).

O sistema implementado para facilitar a captura dos dados utilizado no quasi experimento foi atualizado para refletir as mudanças necessárias para a condução do experimento. Na Figura 4.16, é exibida a tela de descrição de um problema, nela o avaliador especialista pode ver o enunciado, os exemplos de entrada e saída e o conjunto de conteúdos que teria sido trabalhado com os alunos até o momento da aplicação dos exercícios.

Na Figura 4.17, é exibida a tela desenvolvida para coletar as avaliações, nela o avaliador deve fornecer uma nota, no intervalo $[0; 10]$, um *feedback* e uma observação, tal como descrito anteriormente.

O conjunto de conteúdos que teria sido trabalhado com os alunos até o momento da aplicação dos exercícios é determinado com base na classificação do problema. A seguir, é apresentada a descrição dos conteúdos para cada tipo de problema:

- Para problemas do tipo 'Básico'
  - Conceitos básicos sobre algoritmos;
  - Compilação e interpretação;
  - Variáveis e tipos de dados;
  - Operadores matemáticos;

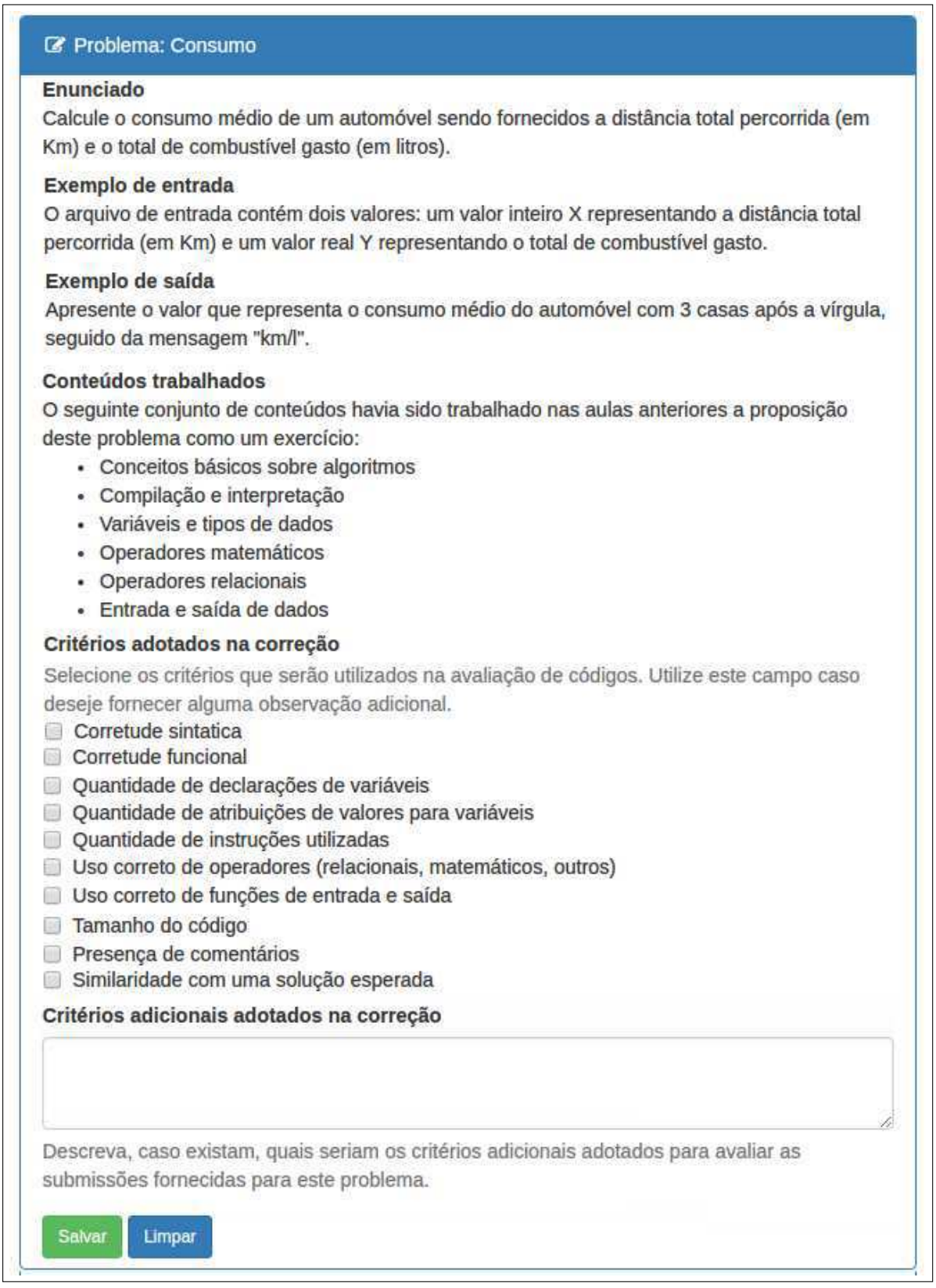


Figura 4.16: Tela de descrição de um problema como visualizado por um avaliador especialista.

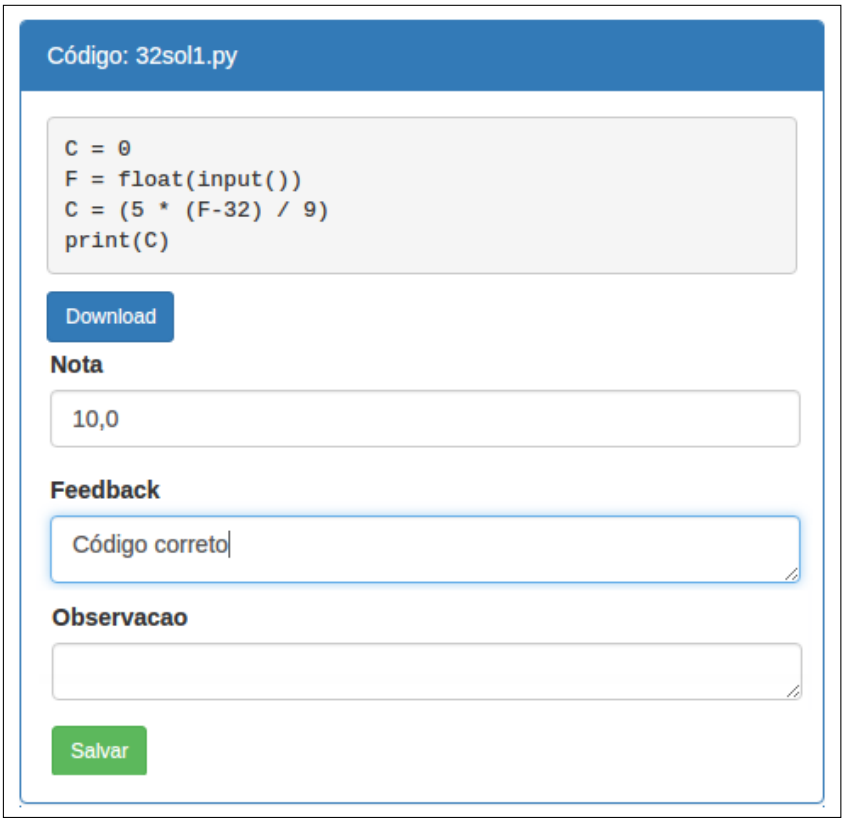


Figura 4.17: Tela de avaliação de um código.

  - Operadores relacionais;
  - Entrada e saída de dados.

- Para problemas do tipo ‘Decisão’
  - todos os conteúdos anteriores;
  - Operadores lógicos;
  - Estrutura condicional simples (if);
  - Estrutura condicional composta (if-elif-else);
  - Estruturas condicionais aninhadas (ou encadeadas).

- Para problemas do tipo ‘Repetição’
  - todos os conteúdos anteriores;
  - Estrutura de repetição (for);

  - Estrutura de repetição (while);
  - Estruturas de repetição aninhadas.

Os critérios que seriam adotados para as avaliações dos códigos deveriam ser selecionados na tela exibida na Figura 4.16. O avaliador especialista poderia ainda descrever critérios adicionais, caso necessário. Os critérios exibidos são relacionados com base na classificação do problema. A seguir, é apresentada a descrição dos critérios para cada tipo de problema:

- Para problemas do tipo 'Básico'
  - corretude sintática;
  - corretude funcional;
  - quantidade de declarações de variáveis;
  - quantidade de atribuições de valores para variáveis;
  - quantidade de instruções utilizadas;
  - uso correto de operadores (relacionais, matemáticos, outros);
  - uso correto de funções de entrada e saída;
  - tamanho do código;
  - presença de comentários;
  - similaridade com uma solução esperada.
- Para problemas do tipo 'Decisão'
  - todos os critérios anteriores;
  - aplicabilidade da estrutura de decisão utilizada (ex. if vs. switch);
  - quantidade de decisões utilizadas;
  - profundidade das decisões utilizadas;
  - presença/ausência de decisões em sequência;
  - presença/ausência de decisões compostas;
  - quantidade de condições especificadas em cada decisão;

  - aplicabilidade dos operadores lógicos utilizados;
  - quantidade de operadores lógicos utilizados.

- Para problemas do tipo 'Repetição'
  - todos os critérios anteriores;
  - aplicabilidade da estrutura de repetição utilizada (ex. for vs. while);
  - quantidade de estruturas de repetição utilizadas;
  - profundidade das repetições;
  - presença/ausência de repetições aninhadas;
  - eficiência em relação ao tempo de processamento;
  - eficiência em relação ao uso de memória.

Os avaliadores especialistas foram alocados para avaliar as submissões de todos os problemas, sendo estes apresentados em uma ordem aleatória. Do mesmo modo, as submissões eram apresentadas em uma ordem aleatória para cada avaliador especialista em relação a cada problema. Os códigos foram selecionados aleatoriamente entre todas as submissões realizadas para um problema, somente considerando submissões realizadas na linguagem *Python* (2 ou 3). Para todos os problemas, a mesma quantidade de códigos foi selecionada, sendo selecionados 20 submissões por problema.

Foram solicitadas dez avaliações de códigos para cada problema. A captura da avaliação ocorreu através do sistema de captura, cuja tela de avaliação é exibida na Figura 4.17. Após realizar todas as avaliações para um problema, os agrupamentos eram gerados e então, um avaliador especialista deveria confirmar as notas e *feedbacks* textuais generalizados. A captura das confirmações ocorreu através do sistema de captura, cujas telas de confirmação são exibidas nas Figuras 4.18 e 4.19.

Na Figura 4.18, é exibida a nota gerada, caso o avaliador julgue que a nota não é adequada ele pode fornecer um nova nota. De modo similar, na Figura 4.19, é exibido o *feedback* gerado, caso o avaliador julgue que o *feedback* não é adequado ele pode fornecer um novo *feedback*, adicionalmente o avaliador especialista deve indicar o nível de adequação do *feedback* gerado.

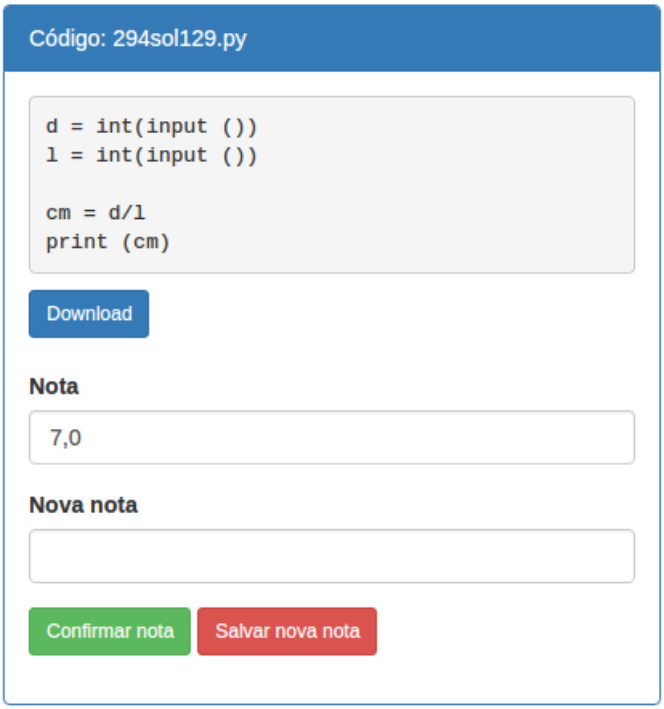


Figura 4.18: Tela de confirmação para nota.

4.5.3 Descrição dos avaliadores

Nesta seção, são exibidas descrições relacionadas ao perfil dos avaliadores especialistas. Para preservar a identidade de cada um dos avaliadores especialistas, a identificação deles será realizada através de valores inteiros (‘id’). A identificação ‘id’ do experimento não possui relação com a identificação ‘id’ do quasi experimento.

Os avaliadores especialistas participaram do *survey* conduzido em uma das etapas das pesquisas relacionadas com esta tese. Ao longo do questionário, foram solicitadas informações sobre o perfil do respondente, estas solicitações são exibidas a seguir:

- Possui experiência como professor? Sim/Não;
- Nível de formação? Doutorado/Mestrado/Especialização/Graduação/Graduando;
- Possui experiência como monitor de disciplina? Sim/Não;
- Possui experiência como monitor de disciplina de programação? Sim/Não;
- Quais das seguintes linguagens já adotou em uma disciplina introdutória de programação? C/Python/Java/Pascal/Outra.

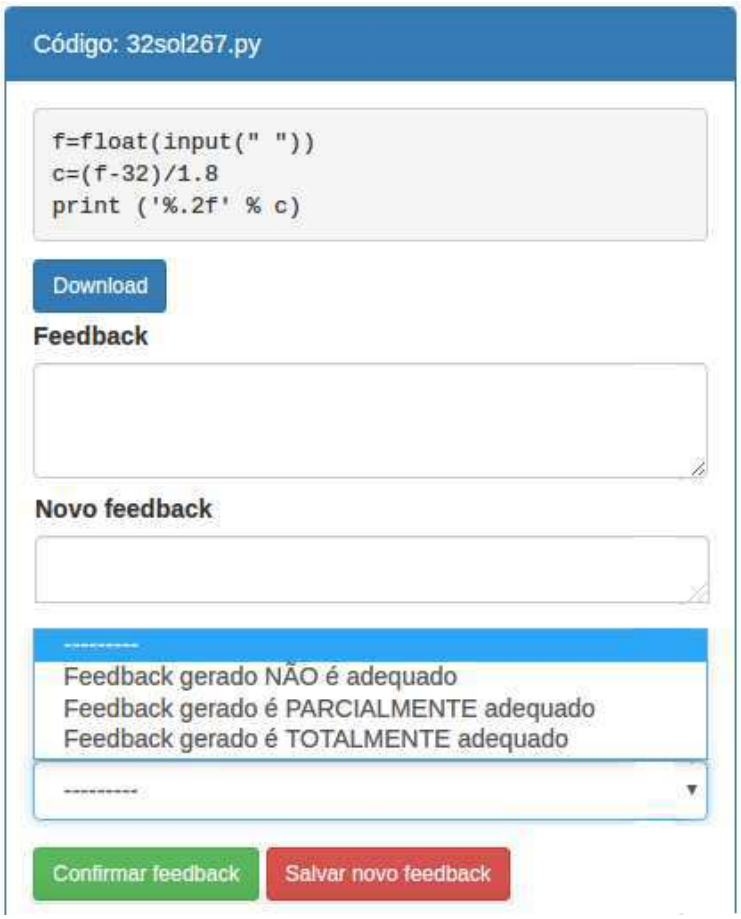


Figura 4.19: Tela de confirmação para *feedback* textual.

Foram selecionados apenas avaliadores que ministraram disciplinas introdutórias de programação utilizando a linguagem Python (2 ou 3) na condução da disciplina. Do mesmo modo, apenas monitores/ex-monitores que realizaram as atividades de monitoria em disciplinas introdutórias de programação e que utilizaram a linguagem Python (2 ou 3) foram selecionados.

Durante a etapa de validação dos dados foi possível observar que alguns dos avaliadores que realizaram a avaliação completa dos seis problemas apresentados, cometeram erros ou tiveram problemas em relação aos procedimentos de avaliação empregados. Desta forma, o conjunto de avaliações associado com estes avaliadores foi descartado.

A seguir são apresentados os perfis dos avaliadores que concluíram, sem erros ou problemas, a avaliação completa dos seis exercícios que foram apresentados.

Os avaliadores 'id:1' e 'id:5' possuem mestrado e experiência como professor, mas não desempenharam atividades de monitoria. Estes avaliadores realizaram a avaliação completa das submissões para os 6 problemas solicitados.

O avaliador 'id:19' possui doutorado e experiência como professor, mas não desempenhou atividade de monitoria. Este avaliador realizou a avaliação completa das submissões para os 6 problemas solicitados.

O avaliador 'id:6' é graduado em Ciência da Computação, não possui experiência como professor e desempenhou atividade de monitoria em ao menos uma disciplina de programação. Este avaliador realizou a avaliação completa das submissões para os 6 problemas solicitados.

Os avaliadores 'id:3', 'id:17', 'id:25', 'id:26' e 'id:27' são todos graduandos em Ciência da Computação, não possuem experiência como professor e desempenharam atividade de monitoria em ao menos uma disciplina de programação. Estes avaliadores realizaram a avaliação completa das submissões para os 6 problemas solicitados.

Todos os professores que participaram do experimento foram voluntários, nenhum incentivo às participações foi apresentado, estes realizaram as avaliações e confirmações no momento e ambiente em que julgaram mais adequados. Todos os monitores que participaram do experimento foram voluntários, incentivados a participar através de créditos em plataformas de serviços [1], estes realizaram as avaliações e confirmações no momento e am-

[1] créditos no Google Play e créditos no Uber

biente em que julgaram mais adequados.

4.5.4 Execução da abordagem

A condução do experimento investigou se um *feedback* objetivo (notas no intervalo [0;10]) e um *feedback* subjetivo (informação textual) poderiam ser gerados, e na comparação do conjunto de *feedbacks* gerado com o fornecido por avaliadores especialistas (professores ou monitores/ex-monitores) de programação. O intuito desta investigação era verificar se uma avaliação fornecida por um método baseado em agrupamento de códigos forneceria resultados semelhantes aos de um avaliador humano.

Do mesmo modo que no quasi experimento, inicialmente cada avaliador procedeu com a avaliação de dez códigos, randomização foi utilizada nesta etapa, tal como descrito anteriormente.

As etapas de execução da abordagem foram realizadas da seguinte maneira. A extração das medidas sobre os códigos (Etapa 1) ocorreu logo após a seleção dos códigos, todas as métricas descritas no Capítulo 2 foram utilizadas no experimento. Para as métricas de similaridade, apenas o valor de similaridade em relação a uma solução de referência foi computado.

A identificação das propriedades para agrupamento (Etapa 2), que visa representar os critérios adotados pelo avaliador especialista, foi realizada de duas maneiras distintas, correspondendo à diferentes execuções do experimento. Sendo a primeira execução realizada através de relacionamentos entre os critérios selecionados pelo avaliador e as métricas disponíveis. A segunda execução ocorreu de modo similar ao quasi experimento, onde todas as combinações possíveis de métricas de software foram geradas e utilizadas na etapa posterior. Dessa forma, foi possível selecionar automaticamente as métricas para representar os critérios de avaliação.

A relação entre os critérios selecionados pelo avaliador e as métricas disponíveis ocorreu tomando como base a definição da métrica. Sendo assim, alguns critérios tinham uma relação direta com uma ou mais métricas, enquanto outros não possuem uma relação direta ou total, e outros não possuíam métricas representativas. Desta forma, ao selecionar um critério, uma ou mais métricas eram utilizadas no vetor de propriedades.

Na Tabela 4.12, é exibida a relação de critérios e métricas para problemas classificados

na categoria 'Básico'.

A relação entre critério e métricas exibida para os problemas da categoria 'Básico' é mantida para os problemas da categoria 'Condicional'. Novas relações foram determinadas, na Tabela 4.13 é exibida a relação de critérios e métricas para problemas classificados na categoria 'Condicional'.

A relação entre critério e métricas exibida para os problemas das categorias 'Básico' e 'Condicional' é mantida para os problemas da categoria 'Repetição'. Novas relações foram determinadas, na Tabela 4.14 é exibida a relação de critérios e métricas para problemas classificados na categoria 'Repetição'.

A geração de grupos ou *clustering* (Etapa 3) foi realizada fazendo-se uso do algoritmo *Kmeans*, utilizando $k = 10$. O valor de k determina a quantidade máxima de grupos que será criada pela execução do algoritmo. Desta forma, o valor de k também determina a quantidade de avaliações que serão realizadas antes da generalização dos *feedbacks*. Sendo assim, uma vez que a quantidade de submissões para um problema correspondeu a 20 submissões, o valor de $k = 10$ foi estabelecido para que o esforço, quantidade de avaliações, fosse reduzido a metade.

Na primeira execução, a geração de grupos utilizou o vetor de propriedades determinado na etapa anterior através das relações de critérios e métricas. Uma vez que o agrupamento foi criado, utilizando as avaliações realizadas no início do processo, as notas para os códigos ainda não avaliados eram generalizadas com base no grupo em que este código pertencia. Essa generalização consistiu na atribuição de uma mesma nota para todos os elementos de um mesmo grupo.

Na segunda execução, a geração de grupos utilizou cada um dos vetores de propriedades criados na etapa anterior, através de todas as combinações possíveis entre as métricas. Desta forma, a seleção automática de métricas foi realizada com alto custo de processamento e memória. Assim como no quasi experimento, utilizando as avaliações realizadas no início do processo, buscou-se qual o agrupamento que melhor representava a distribuição de notas em relação aos grupos existentes. O agrupamento que continha os grupos onde as notas possuiam as maiores concordâncias foi selecionado. Quando mais de um agrupamento demonstrava o mesmo grau de concordância, aquele que foi gerado com base na menor quantidade de métricas era selecionado. Utilizando o agrupamento selecionado, os *feedbacks* (objetivo

Tabela 4.12: Relação entre critérios e métricas, para problemas classificados na categoria 'Básico'

Critério	**Métricas**
Corretude sintática	Corretude sintática
Corretude funcional	Corretude funcional
Quantidade de declarações de variáveis	Número de operandos distintos, Número total de operandos
Quantidade de atribuições	Número de operadores distintos, Número total de operadores
Quantidade de instruções utilizadas	LOC, SLOC, Vocabulário Número de operadores distintos, Número de operandos distintos, Número total de operadores, Número total de operandos
Uso correto de operadores (relacionais, matemáticos, outros)	Número de operadores distintos, Número total de operadores
Uso correto de funções de entrada e saída	nenhuma métrica
Tamanho do código	LOC, SLOC, Vocabulário, Tamanho, Volume
Presença de comentários	Linhas de comentário, *String* de múltiplas linhas
Similaridade com uma solução esperada	Similaridade baseada no coeficiente de Jaccard, similaridade baseada na distância de edição de texto, similaridade baseada na distância de edição de árvore

Tabela 4.13: Relação entre critérios e métricas, para problemas classificados na categoria 'Condicional'

Critério	**Métricas**
Aplicabilidade da estrutura de decisão (ex. if vs. switch)	Corretude funcional, Complexidade ciclomática
Quantidade de decisões	Complexidade ciclomática
Profundidade das decisões	Complexidade ciclomática
Presença/ausência de decisões em sequência	Complexidade ciclomática
Presença/ausência de decisões compostas	Complexidade ciclomática
Quantidade de condições especificadas em cada decisão	Número de operandos distintos, Número total de operadores, Número total de operandos, Vocabulário
Aplicabilidade dos operadores lógicos	Corretude funcional, Número de operadores distintos, Número total de operadores
Quantidade de operadores lógicos	Número de operadores distintos Número total de operadores

e subjetivo) foram generalizados, de modo que uma avaliação fornecida para um elemento de um grupo fosse replicada aos demais elementos do mesmo grupo. Para casos onde não existisse elemento no grupo com uma avaliação, foi utilizada a avaliação do elemento mais próximo.

4.5.5 Avaliação da abordagem

A avaliação dos *feedbacks* objetivos foi realizada adotando o mesmo processo utilizado no quasi experimento. Para medir a concordância e o grau de proximidade existente entre as listas de avaliações fornecidas pelo avaliador artificial e pelos avaliadores, foram observadas as seguintes medidas:

- ***Kappa de Cohen*** - captura o grau de concordância entre duas listas de avaliações;
- ***Correlação de Pearson*** - captura o grau de correlação entre avaliações. Uma vez que uma concordância é computada apenas quando um valor exatamente igual é atribuído em ambas as listas, a correlação demonstra se as listas de avaliação possuem variações similares;
- **Distância entre notas** - considerando que as notas são fornecidas em um intervalo numérico, um somatório das diferenças de cada nota existente nas listas de avaliação foi computada.

De modo complementar às medidas citadas, foram gerados gráficos de frequência de notas para cada avaliador na avaliação de cada problema. Caso um avaliador possuísse uma frequência muito alta em uma nota, por exemplo, a maioria das submissões é avaliada como totalmente correta (nota 10), essa característica favoreceria resultados que exibam alta concordância, alta correlação e baixa distância entre notas.

A avaliação dos *feedbacks* textuais ocorreu através da análise do indicador de adequação do *feedback* (‘*feedback NÃO é adequado*’, ‘*feedback é PARCIALMENTE adequado*’, ‘*feedback é TOTALMENTE adequado*’), informado pelo avaliador durante a etapa de confirmação. Além disso, de modo complementar, foi realizada uma análise comparativa entre o *feedback* textual gerado e o *feedback* textual indicado pelo avaliador especialista como o mais adequado. Estes resultados são discutidos na seção seguinte.

4.5.6 Resultados e discussão

Análise dos *feedbacks* objetivos

Os resultados obtidos são apresentados em um resumo geral, sendo separados para o conjunto de avaliadores especialistas formado por professores, seguido dos valores obtidos em relação ao conjunto de avaliadores especialistas formado por monitores. Em ambos os casos serão exibidos os resultados para as duas execuções da abordagem, sendo estas a identificação das propriedades baseada na relação critério e métrica, e a identificação das propriedades baseada na seleção automática de métricas. Os resultados obtidos para cada problema são apresentados e discutidos individualmente no Apêndice E, após exibir os resultados as perguntas de pesquisa serão respondidas.

Após computar os resultados de forma individual para os problemas, realizamos uma análise geral dos dados para cada perfil de avaliadores especialistas. Para a execução relacionada à identificação das propriedades baseada na relação critério e métrica, os resultados gerais obtidos para os avaliadores especialistas professores são apresentados na Tabela 4.15.

Observando os resultados obtidos para os avaliadores especialistas professores, utilizando execução relacionada à identificação das propriedades baseada na relação critério e métrica, foi obtida uma concordância quase perfeita ($kappa = 0.80$), considerando a média. Este resultado é fortalecido pelos valores obtidos em relação às outras medidas. Ocorre, considerando a média, uma correlação forte ($\rho = 0.80$), ou seja, existe uma relação linear crescente entre as avaliações fornecidas. Na distância de notas, para as 120 avaliações ocorreu diferença média de 11.53 pontos em relação às notas, essa diferença seria próxima a um décimo por avaliação.

Para a execução relacionada à identificação das propriedades baseada na seleção automática de métrica, os resultados gerais obtidos para os avaliadores especialistas professores são apresentados na Tabela 4.16.

Observando os resultados obtidos para os avaliadores especialistas professores, utilizando execução relacionada à identificação das propriedades baseada na seleção automática de métricas, foi obtida uma concordância quase perfeita ($kappa = 0.88$), considerando a média. Este resultado é fortalecido pelos valores obtidos em relação às outras medidas. Ocorre, considerando a média, uma correlação muito forte ($\rho = 0.92$), ou seja, existe uma relação

linear crescente entre as avaliações fornecidas. Na distância de notas, para as 120 avaliações ocorreu diferença média de 2.5 pontos em relação às notas, essa diferença seria inferior a um décimo por avaliação.

É importante notar que alguns dos resultados onde foi obtida uma concordância perfeita ($kappa = 1.00$) ocorreram em situações onde o avaliador julgou todas as soluções com a mesma nota. Todos estas ocorrências foram investigadas através de indagações ao avaliador, e foi possível observar que não ocorreu nenhum erro ou problema em relação aos procedimentos de avaliação empregados.

Comparando os resultados obtidos nas duas execuções do experimento, é possível observar que a seleção automática de métricas possibilitou a obtenção de melhores resultados. A descrição dos resultados detalhada por problema, exibida no Apêndice E, permite a visualização da variação de concordâncias e da quantidade de concordâncias perfeitas obtidas em cada execução. Na execução baseada na seleção automática de métricas, a variação entre as concordâncias foi menor e a quantidade de concordâncias perfeitas aumentou.

Os resultados gerais obtidos para os avaliadores especialistas monitores são apresentados na Tabela 4.17, para a execução relacionada à identificação das propriedades baseada na relação critérios e métricas.

Observando os resultados obtidos para os avaliadores especialistas monitores, utilizando execução relacionada à identificação das propriedades baseada na relação critérios e métricas, foi obtida uma concordância substancial ($kappa = 0.64$), considerando a média. Este resultado é fortalecido pelos valores obtidos em relação às outras medidas. Ocorre, considerando a média, uma correlação forte ($\rho = 0.74$), ou seja, existe uma relação linear crescente entre as avaliações fornecidas. Na distância de notas, para as 120 avaliações ocorreu diferença média de 18.66 pontos em relação às notas, essa diferença seria próxima a um décimo por avaliação.

Os resultados gerais obtidos para os avaliadores especialistas monitores são apresentados na Tabela 4.18, para a execução relacionada à identificação das propriedades baseada na seleção automática de métricas.

Observando os resultados obtidos para os avaliadores especialistas monitores, utilizando execução relacionada à identificação das propriedades baseada na seleção automática de métricas, foi obtida uma concordância substancial ($kappa = 0.79$), considerando a média. Este

resultado é fortalecido pelos valores obtidos em relação às outras medidas. Ocorre, considerando a média, uma correlação forte ($\rho = 0.86$), ou seja, existe uma relação linear crescente entre as avaliações fornecidas. Na distância de notas, para as 120 avaliações ocorreu diferença média de 8.7 pontos em relação às notas, essa diferença seria inferior a um décimo por avaliação.

Assim como ocorreu com os avaliadores especialistas com perfil de professor, a comparação dos resultados obtidos nas duas execuções do experimento permite observar que a seleção automática de métricas possibilitou a obtenção de melhores resultados. A descrição dos resultados detalhada por problema, exibida no Apêndice E, permite a visualização da variação de concordâncias e da quantidade de concordâncias perfeitas obtidas em cada execução. Na execução baseada na seleção automática de métricas, a variação entre as concordâncias foi menor e a quantidade de concordâncias perfeitas aumentou.

Sobre a QP1 para qualquer das execuções do experimento e em ambos os perfis analisados, considerando a média, obteve-se, no pior caso, uma concordância substancial e, no melhor caso, uma concordância quase perfeita. Desta forma, observados estes resultados, é possível afirmar que a avaliação baseada nos agrupamentos fornece uma avaliação semelhante a de avaliadores especialistas. Este resultado é fortalecido pelos os valores obtidos em relação às outras medidas. Ocorre, considerando a média, uma correlação forte, em qualquer dos cenários investigados, ou seja, existe relação linear crescente entre as avaliações fornecidas. Na distância de notas, para as 120 avaliações ocorreu diferença média inferior a um décimo em relação às notas.

As concordâncias entre todos os pares de avaliadores professores e o avaliador artificial são apresentadas na Tabela 4.19. Já as concordâncias entre todos os pares de avaliadores monitores e o avaliador artificial são apresentadas na Tabela 4.20.

Observando os dados apresentados, considerando professores e monitores, a maior concordância obtida correspondeu a uma concordância leve ($kappa = 0.15$, para os professores e $kappa = 0.22$, para os monitores). Considerando a média por avaliador, ou seja, o quanto um avaliador concorda com seus pares, o valor máximo de concordância obtido corresponde a uma concordância leve ($kappa = 0.13$, para os professores e $kappa = 0.16$, para os monitores).

Comparando os valores de concordâncias entre humanos com os valores obtidos em re-

lação ao avaliador artificial, exibidos nas Tabelas 4.19 e 4.20, independente do modo como a abordagem foi executada, as concordâncias do avaliador artificial em relação aos humanos é superior àquelas exibidas entre pares de avaliadores humanos. A menor concordância obtida corresponde a uma concordância razoável ($kappa = 0.34$), exibida na Tabela E.1. Considerando a média geral por avaliador, no pior caso foi obtida uma concordância moderada ($kappa = 0.55$).

A visualização das informações pode ser facilitada pelos gráficos exibidos nas Figuras 4.20 e 4.21, onde é possível comparar visualmente os valores das concordâncias obtidas entre avaliadores e avaliador artificial em relação a qualquer uma das concordâncias entre um avaliador e seus pares.

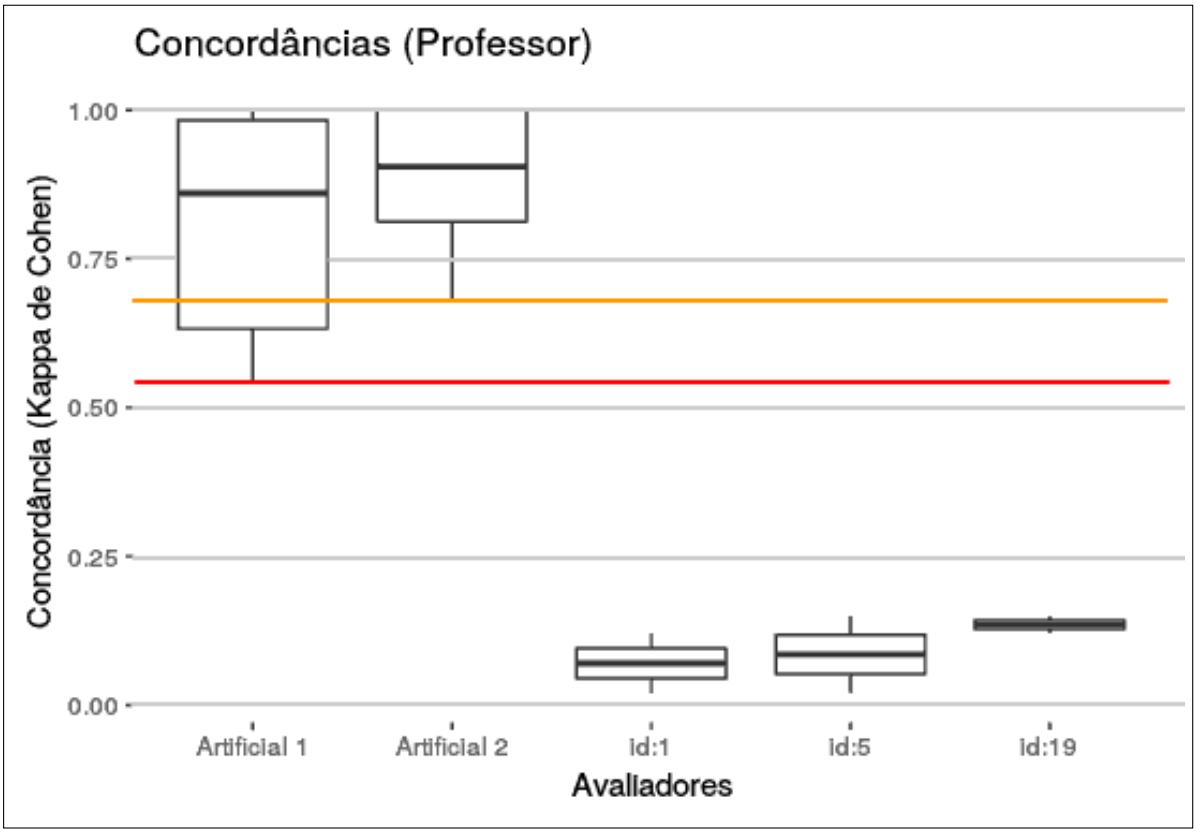


Figura 4.20: Concordâncias obtidas pelo avaliador artificial em relação aos avaliadores especialistas professores e das concordâncias gerais para cada avaliador em relação aos seus pares, considerando todas as avaliações realizadas.

Dessa forma, uma vez que não existe interseção entre os *boxplots*, é possível afirmar que

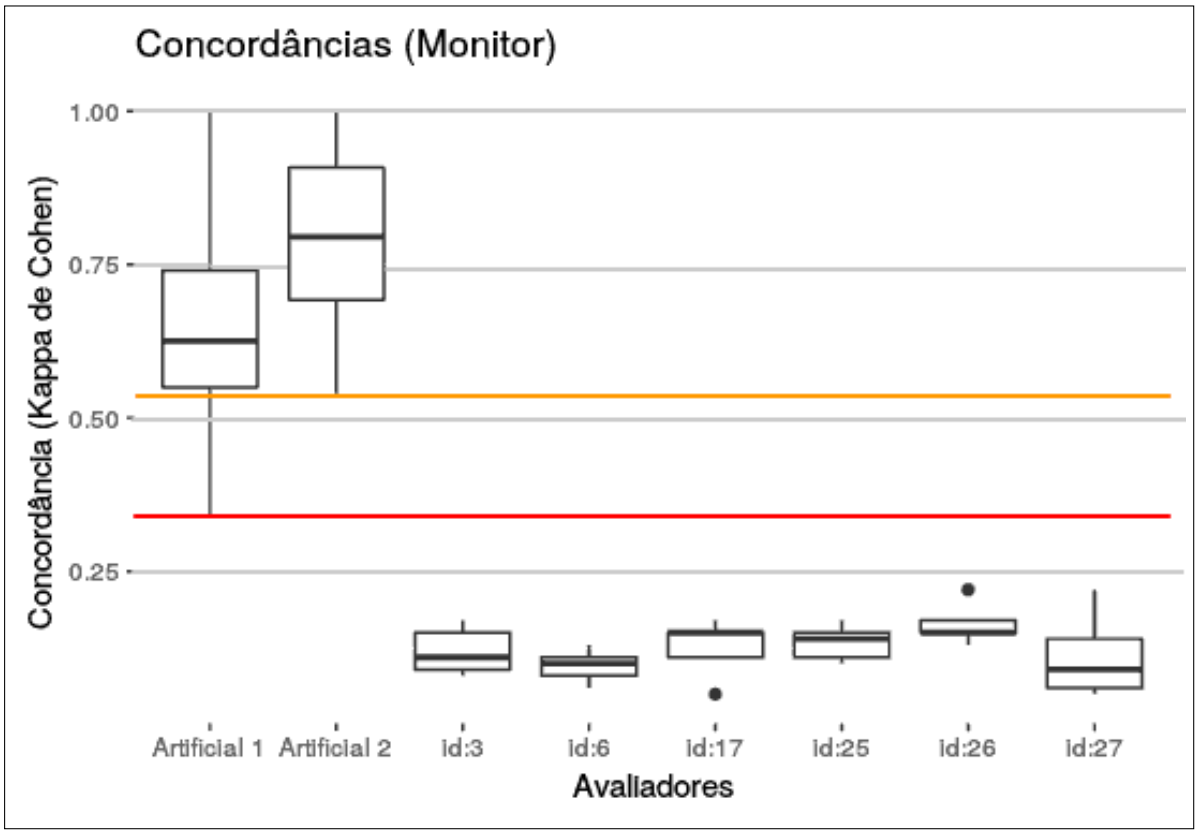


Figura 4.21: Concordâncias obtidas pelo avaliador artificial em relação aos avaliadores especialistas monitores e das concordâncias gerais para cada avaliador em relação aos seus pares, considerando todas as avaliações realizadas.

a concordância do avaliador artificial em relação aos avaliadores é superior àquela existente entre cada avaliador em relação aos seus pares.

Sobre a QP2, considerando a média, no pior caso obteve-se uma concordância moderada ($kappa = 0.55$) entre avaliador artificial e avaliador humano e, no melhor caso, uma concordância leve ($kappa = 0.16$) entre avaliador artificial e avaliador humano. Sendo assim, podemos afirmar que a avaliação do avaliador artificial fornece uma concordância superior àquela obtida entre dois avaliadores humanos.

Assim como ocorreu na análise exibida no quasi experimento, uma vez que uma concordância só é computada quando exatamente o mesmo valor de nota é fornecido para uma avaliação, faz-se necessária a observação de outras medidas como critério para verificar a distância entre as avaliações dos avaliadores.

Para isso, foram computados e são apresentados a seguir os resultados para as medidas, correlação de Pearson e distância total de notas.

Na Tabela 4.21, são exibidos os valores de correlação entre as avaliações de cada avaliador professor e seus pares considerando todas as avaliações. É possível notar, que considerando a média geral por avaliador, no pior caso foi obtida uma correlação fraca (0.46) e, no melhor caso, foi obtida uma correlação moderada (0.59) entre os avaliadores.

Na Tabela 4.22, são exibidos os valores de distâncias de notas entre as avaliações de cada avaliador professor e seus pares considerando todas as avaliações. É possível notar, que considerando a média geral por avaliador, no pior caso foi obtida uma distância de notas de 220 pontos, correspondendo a aproximadamente dois pontos por avaliação e, no melhor caso, foi obtida uma distância de notas de 182.5 pontos, correspondendo a menos de dois pontos por avaliação.

Na Tabela 4.23, são exibidos os valores de correlação entre as avaliações de cada avaliador monitor e seus pares considerando todas as avaliações. É possível notar, que considerando a média geral por avaliador, no pior caso foi obtida uma correlação fraca (0.48) e, no melhor caso foi obtida uma correlação moderada (0.66) entre os avaliadores.

Na Tabela 4.24, são exibidos os valores de distâncias de notas entre as avaliações de cada avaliador professor e seus pares considerando todas as avaliações. É possível notar, que considerando a média geral por avaliador, no pior caso foi obtida uma distância de notas de 352.9 pontos, correspondendo a aproximadamente três pontos por avaliação e, no melhor caso, foi obtida uma distância de notas de 227.1 pontos, correspondendo a menos de dois pontos por avaliação.

A visualização destas informações pode ser facilitada pelos gráficos das Figuras 4.22 e 4.23, onde são exibidos os *boxplot* para as correlações entre professores e entre monitores, respectivamente. Do mesmo modo, nos gráficos das Figuras 4.24 e 4.25, são exibidos os *boxplot* para as distâncias de notas das avaliações fornecidas por professores e por monitores, respectivamente.

Na Tabela 4.25 é possível observar o resumo de dados em relação a métrica de distância de notas na primeira execução do experimento, para cada um dos perfis de avaliadores.

Na Tabela 4.26 é possível observar o resumo de dados em relação a métrica de distância de notas na segunda execução do experimento, para cada um dos perfis de avaliadores.

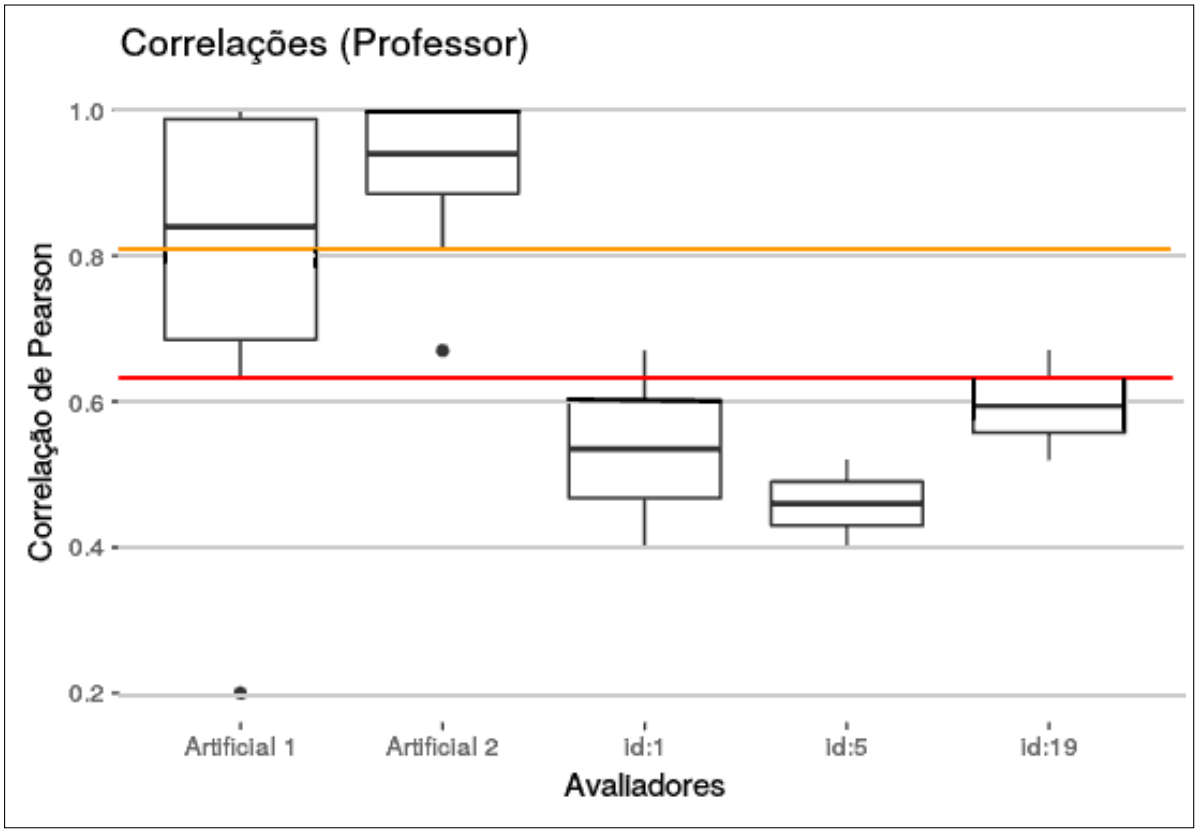


Figura 4.22: Correlações obtidas pelo avaliador artificial em relação aos avaliadores e das correlações gerais para cada avaliador especialista professor em relação aos seus pares, considerando todas as avaliações realizadas.

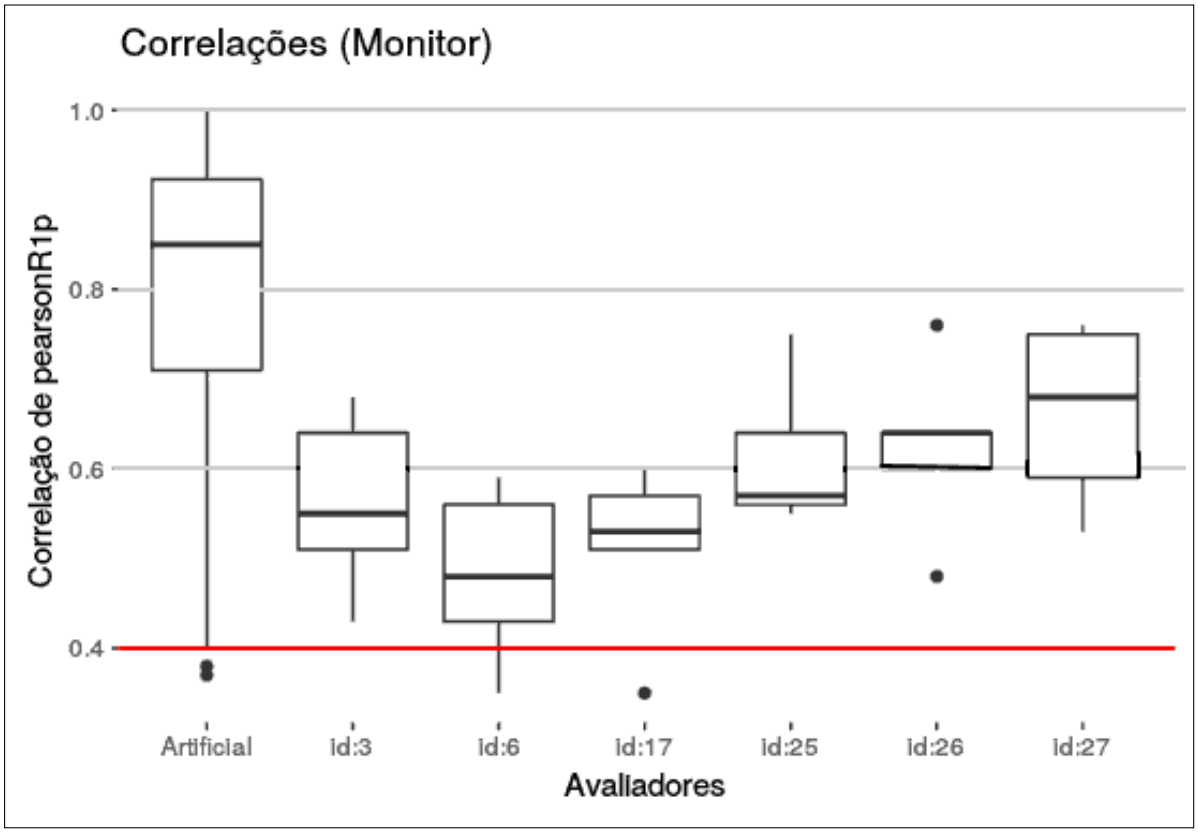


Figura 4.23: Correlações obtidas pelo avaliador artificial em relação aos avaliadores e das correlações gerais para cada avaliador especialista monitor em relação aos seus pares, considerando todas as avaliações realizadas.

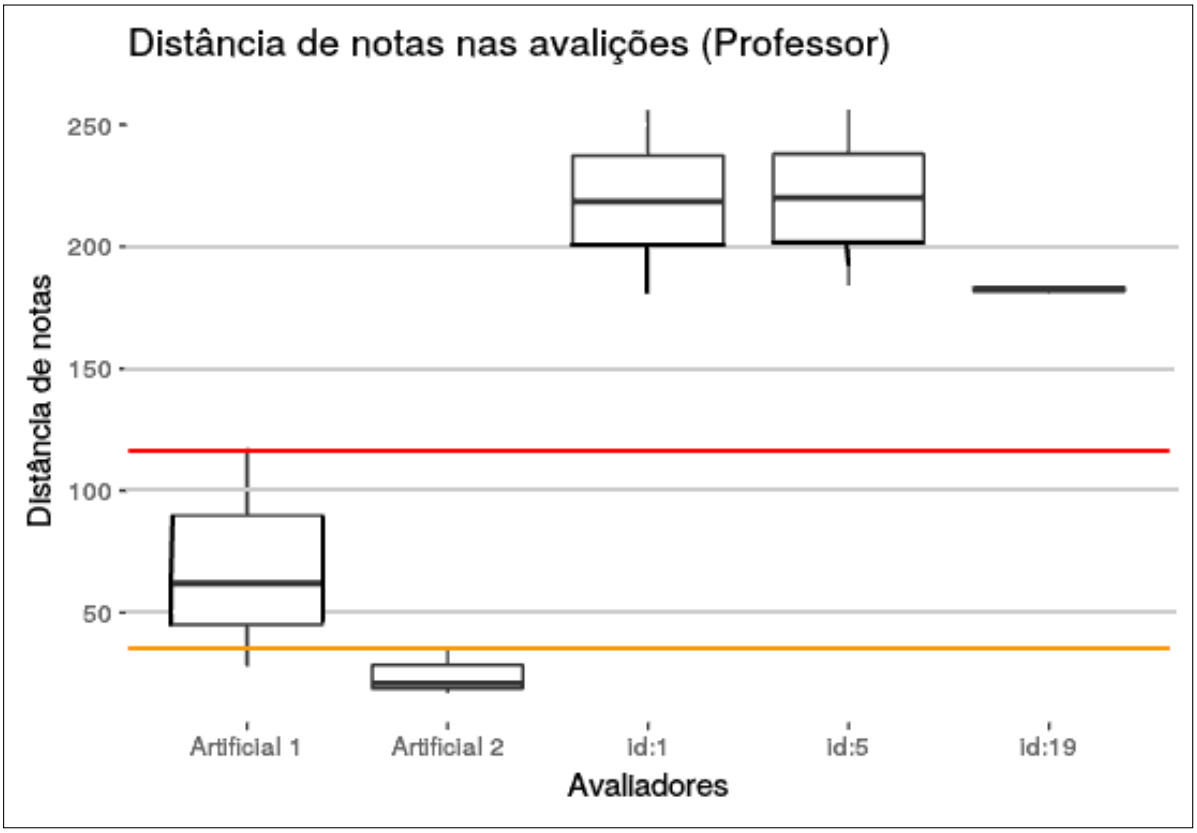


Figura 4.24: Distâncias de nota obtidas pelo avaliador artificial em relação aos avaliadores e das distâncias de nota para cada avaliador especialista professor em relação aos seus pares, considerando todas as avaliações realizadas.

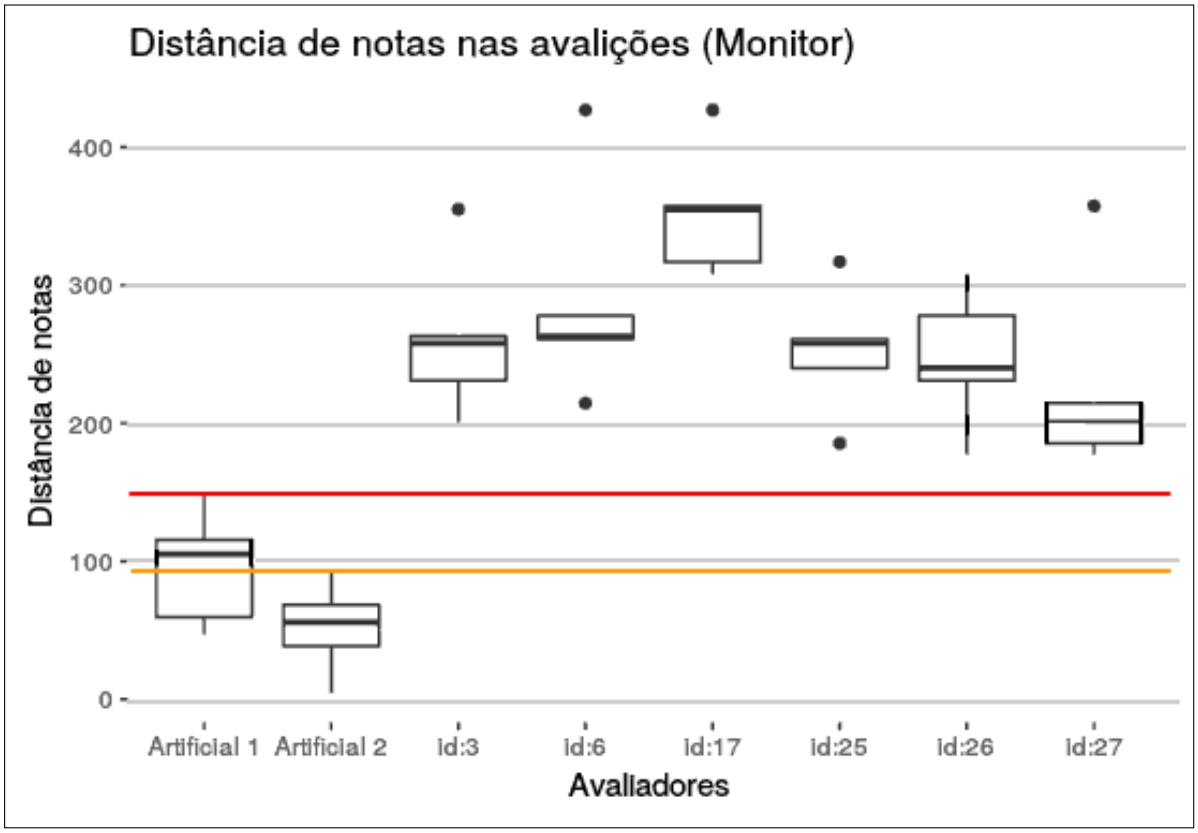


Figura 4.25: Distâncias de nota obtidas pelo avaliador artificial em relação aos avaliadores e das distâncias de nota para cada avaliador especialista monitor em relação aos seus pares, considerando todas as avaliações realizadas.

Análise dos *feedbacks* subjetivos

Os *feedbacks* subjetivos foram analisados através das classificações fornecidas pelos avaliadores no momento em que realizavam as confirmações. Um *feedback* subjetivo poderia ser classificado como: 'NÃO aplicável'; 'PARCIALMENTE aplicável' ou 'TOTALMENTE aplicável'. Alguns avaliadores não informaram a classificação do *feedback* subjetivo gerado, estes casos não serão contabilizados em nenhuma das classes anteriores.

Os resultados das classificações, em ambos os cenários de execução da abordagem, são apresentados na Tabela 4.27, para as classificações fornecidas pelos avaliadores professores e, na Tabela 4.28, para as classificações fornecidas pelos avaliadores monitores.

Para o cenário onde as avaliações foram fornecidas pelos professores, utilizando a identificação de propriedades baseada na relação de critérios e métricas, a maior parte dos *feedbacks* subjetivos foi classificado como 'TOTALMENTE aplicável'. A classificação 'NÃO aplicável' foi indicada na menor parte das avaliações. Para os casos em que não foi informada a adequação do *feedback*, duas situações são as mais comuns. A primeira situação se refere a quando a solução foi julgada como correta, e o *feedback* original contém mensagens parabenizando o aluno. A segunda situação se refere a quando nenhum *feedback* é fornecido originalmente, neste caso o *feedback* original e o *feedback* gerado se encontram vazios.

Para o cenário onde as avaliações foram fornecidas pelos professores, utilizando a identificação de propriedades baseada na seleção automática de métricas, era esperado uma melhoria em relação a classificação dos *feedbacks* textuais, uma vez que os *feedbacks* objetivos melhoraram. Contudo, a maior parte dos *feedbacks* subjetivos foi classificado como 'PARCIALMENTE aplicável'. A classificação 'NÃO aplicável' foi indicada na menor parte das avaliações. A quantidade de classificações 'TOTALMENTE aplicável', 'NÃO aplicável' e 'não informada' foram menores. Esse resultado é justificado pela característica subjetiva existente nesta classificação, ou seja, cada avaliador julgou de acordo com suas motivações quando classificar um *feedback* como 'NÃO aplicável', 'PARCIALMENTE aplicável' ou 'TOTALMENTE aplicável'. Para os casos em que não foi informado a adequação do *feedback*, os mesmos cenários anteriores se mantêm.

Desta forma, não é possível descrever que a execução da identificação de propriedades baseada na seleção automática de métricas resultou na obtenção de melhores resultados em relação aos *feedbacks* subjetivos.

Para o cenário onde as avaliações foram fornecidas pelos monitores, utilizando a identificação de propriedades baseada na relação de critérios e métricas, a maior parte dos *feedbacks* subjetivos foi classificado como 'TOTALMENTE aplicável'. A classificação 'PARCIALMENTE aplicável' foi indicada na menor parte das avaliações. Do mesmo modo que para o perfil de professores, esse resultado é justificado pela característica subjetiva existente nesta classificação, ou seja, cada avaliador julgou de acordo com suas motivações quando classificar um *feedback* como 'NÃO aplicável', 'PARCIALMENTE aplicável' ou 'TOTALMENTE aplicável'.

Para os casos em que não foi informado a adequação do *feedback*, os mesmos cenários anteriores se mantêm, sendo para os monitores mais comum a ocorrência de *feedbacks* originais e gerados vazios.

Para o cenário onde as avaliações foram fornecidas pelos monitores, utilizando a identificação de propriedades baseada na seleção automática de métricas, do mesmo modo que para o perfil de professor, era esperado uma melhoria em relação a classificação dos *feedbacks* textuais, uma vez que os *feedbacks* objetivos melhoraram. Contudo, para os monitores, a maior parte dos *feedbacks* subjetivos foi classificado como 'TOTALMENTE aplicável'. A classificação 'NÃO aplicável' foi indicada na menor parte das avaliações. A quantidade de classificações 'TOTALMENTE aplicável' e 'NÃO aplicável' foram menores. Para os casos em que não foi informada a adequação do *feedback*, os mesmos cenários anteriores se mantêm.

Desta forma, não é possível descrever que a execução da identificação de propriedades baseada na seleção automática de métricas resultou na obtenção de melhores resultados em relação aos *feedbacks* subjetivos.

Sendo assim, observados os resultados, é possível descrever que não existem diferenças significativas entre as classificações quando são comparados as diferentes execuções da abordagem.

A seguir, são exibidos exemplos de soluções e os *feedbacks* associados a eles. Uma vez que as diferentes execuções da abordagem não possuem diferenças significativas, as descrições a seguir não identificam em qual execução o *feedback* foi produzido.

Código Fonte 4.7: Solução apresentada para o problema 'Classificação de triângulos', onde o *feedback* textual foi classificado como 'NÃO aplicável'

```
1  a = float(input())
2  b = float(input())
3  c = float(input())
4
5  if (a == b) & (b == c):
6      print('equilatero')
7  elif (a != b) & (b != c) & (a != c):
8      print('escaleno')
9  else:
10     print('isosceles')
```

"*As possíveis saídas estão corretas, entretanto as entradas são do tipo inteiro, e o problema quer como entrada números reais. No primeiro if, não há necessidade da última verificação, por exemplo, quando afirma-se que a == b e a == c, logo b == c por transitividade. Então a verificação b == c se torna desnecessária.*"

(*Feedback* gerado para o Código 4.7)

"*O programa está correto. Entretanto em Python no lugar do operador lógico &, costuma-se usar o 'and.*"'

(*Feedback* correto para o Código 4.7)

Para o Código 4.7, é possível observar que o *feedback* gerado não possui nenhuma aplicabilidade em relação à solução apresentada. O avaliador critica no *feedback* gerado a entrada de dados do tipo inteiro, quando a solução utilizou a entrada de dados correta (*float*). Além disso, também é criticada a não utilização do conceito de transitividade na solução, embora este conceito esteja sendo utilizado.

Código Fonte 4.8: Solução apresentada para o problema 'Fatorial', onde o *feedback* textual foi classificado como 'NÃO aplicável'

```
1  vetor = list(range(100))
2  aux=0
3  while True:
4          vetor[aux]=int(input())
5          if vetor[aux] == -1:
6                  break
```

```
7           aux=aux+1
8   aux1=0
9   while True:
10          num=vetor[aux1]
11          if vetor[aux1] == -1:
12                  break
13          while num != 1:
14                  num=num-1
15                  vetor[aux1]=vetor[aux1]*num
16          print(vetor[aux1])
17          aux1=aux1+1
```

"*Os valores dos fatorias são calculados corretamente. Contudo, a solicitação pede que sejam calculados fatoriais até uma determinada condição, o código executa os cálculos uma vez e finaliza.*"

(*Feedback* gerado para o Código 4.8)

"*Programa não gera saída, apenas permanece em execução. Além disso, a questão era simples de ser implementada... não necessitava do uso de listas*"

(*Feedback* correto para o Código 4.8)

Para o Código 4.8, o avaliador classificou o *feedback* como 'NÃO aplicável', para este caso devem ser apresentadas algumas considerações. A primeira observação ocorre devido ao fato de que nenhuma solução semelhante foi apresentada para o avaliador, assim o algoritmo de agrupamento inseriu esta em um *cluster* onde não ocorria uma boa similaridade com os outros elementos. Essa situação possibilitou identificar uma melhoria para a abordagem, ou seja, quando uma solução diferente daquelas que foram inicialmente avaliadas surgir, é mais interessante solicitar uma nova avaliação. A segunda observação se refere a um erro cometido pelo avaliador. O avaliador descreve "*Programa não gera saída, apenas permanece em execução [...]*", o enunciado solicita 'O programa encerra a sua execução quando o número n dado for -1', a solução apresentada exibe as saídas apenas após a entrada '-1' ser fornecida. Com isso, compreendemos que a classificação poderia ser indicada como 'PARCIALMENTE aplicável', uma vez que o trecho "*Os valores dos fatorias são calculados corretamente.*" pode se referir a solução apresentada.

Código Fonte 4.9: Solução apresentada para o problema 'Consumo', onde o *feedback* textual foi classificado como 'PARCIALMENTE aplicável'

```
1 d = int(input())
2 l = int(input())
3
4 cm = d/l
5 print(cm)
```

"*Faltou ajustar as casas decimais da saída do programa.*"

(*Feedback* gerado para o Código 4.9)

"*Faltou ajustar as casas decimais da saída do programa. A segunda entrada está recebendo um número inteiro (int) e não um real (float).*"

(*Feedback* correto para o Código 4.9)

O *feedback* textual gerado para o Código 4.9 ilustra uma das situações mais comuns onde ocorre a classificação 'PARCIALMENTE aplicável'. Nesta situação, o *feedback* gerado contém a informação que o avaliador deseja fornecer ao discente. Contudo, não são exibidas informações adicionais desejadas.

Código Fonte 4.10: Solução apresentada para o problema '3 números em ordem crescente', onde o *feedback* textual foi classificado como 'PARCIALMENTE aplicável'

```
1  a = int(input())
2  b = int(input())
3  c = int(input())
4  if(a >= b)and(a >= c):
5      if(b >= c):
6          print(c)
7          print(b)
8          print(a)
9      else:
10         print(b)
11         print(c)
12         print(a)
13 elif(b >= a)and(b >= c):
14     if(a >= c):
```

```
15              print(c)
16              print(a)
17              print(b)
18          else:
19              print(a)
20              print(c)
21              print(b)
22  elif(c >= a)and(c >= b):
23          if(a >= b):
24              print(b)
25              print(a)
26              print(c)
27          else:
28              print(a)
29              print(b)
30              print(c)
31  elif(a==b==c):
32          print(a)
33          print(b)
34          print(c)
```

"*Excesso de condições lógicas, e ainda assim não resolve o problema.*"

(*Feedback* gerado para o Código 4.10)

"*Excesso de condições lógicas.*"

(*Feedback* correto para o Código 4.10)

O *feedback* textual gerado para o Código 4.10 ilustra outra situação comum onde ocorre a classificação 'PARCIALMENTE aplicável'. Neste caso, o *feedback* gerado contém a informação que o avaliador deseja fornecer ao discente. Contudo, também são apresentadas informações adicionais não aplicáveis.

Código Fonte 4.11: Solução apresentada para o problema '3 números em ordem crescente', onde o *feedback* textual foi classificado como 'PARCIALMENTE aplicável'

```
1  a = int(input())
2  b = int(input())
```

```
3  c = int(input())
4  if a < b and a < c and b < c:
5      print(a, b, c)
6  elif a < c and a < b and c < b:
7      print(a, c, b)
8  elif b < a and b < c and a < c:
9      print(b, a, c)
10 elif b < c and b < a and c < a:
11     print(b, c, a)
12 elif c < a and c < b and a < b:
13     print(c, a, b)
14 else:
15     if c < b and c < a and b < a:
16         print(c, b, a)
```

"*Quando a entrada tem dois números iguais, não gera nenhuma saída, o ideal é verificar usando os operadores >= ou <=. Quando o programa gera apenas uma saída, o ideal é usar apenas um if seguido de elif's e por último um else. Visto que vários if's em sequência tornam o código menos eficiente, pois mesmo que alguma saída já tenha sido gerada, todos os outros if's serão verificados.*"

(*Feedback* gerado para o Código 4.11)

"*Quando a entrada tem dois números iguais, não gera nenhuma saída, o ideal é verificar usando os operadores >= ou <=. A solução está printando os números na mesma linha, e o problema pede cada número em uma linha seguido de uma quebra de linha.*"

(*Feedback* correto para o Código 4.11)

O *feedback* textual gerado para o Código 4.11 corresponde à ocorrência das duas situações anteriores, onde ocorre a classificação 'PARCIALMENTE aplicável'. Neste caso, o *feedback* gerado contém a informação que o avaliador deseja fornecer ao discente. Contudo, também são apresentadas informações adicionais não aplicáveis e não são exibidas informações desejadas.

Grande parte dos *feedbacks* textuais classificados como 'TOTALMENTE aplicável' se referem a soluções avaliadas como totalmente corretas. Desta forma, em muitos casos, a

informação fornecida corresponde à felicitação ao aluno, por exemplo 'Parabéns', a indicação da corretude 'Solução correta', ou ao não fornecimento de esclarecimento, texto vazio. Outras situações são apresentadas a seguir.

Código Fonte 4.12: Solução apresentada para o problema 'Classificação de Triângulos', onde o *feedback* textual foi classificado como 'TOTALMENTE aplicável'

```
1 a = int(input())
2 b = int(input())
3 c = int(input())
4 if a == b == c:
5         print ("equilatero")
6 elif a == b or a == c or c == b:
7         print ("isosceles")
8 elif a != b != c:
9         print ("escaleno")
```

"*Parabéns! Mas poderia ter usado menos condições para resolver o problema*"

(*Feedback* gerado para o Código 4.12 e classificado como 'TOTALMENTE aplicável'.)

Para o Código 4.12, apesar do avaliador ter julgado o código como correto, são fornecidas indicações sobre como o aluno poderia melhorar sua solução.

Código Fonte 4.13: Solução apresentada para o problema 'Consumo', onde o *feedback* textual foi classificado como 'TOTALMENTE aplicável'

```
1 d = int(input ())
2 l = int(input ())
3
4 cm = d/l
5 print (cm)
```

"*A precisão de três casas decimais não está sendo respeitada na saída. Sugiro nomear as variáveis de modo mais descritivo. Utilize nomes que descrevam o que a variável armazena, por exemplo, 'distancia' ou 'km' no lugar de 'd'*"

(*Feedback* gerado para o Código 4.13 e classificado como 'TOTALMENTE aplicável'.)

Para o Código 4.13, o avaliador fornece a indicação sobre como o aluno poderia corrigir sua solução. Além disso, são fornecidas sugestões para legibilidade do código.

Código Fonte 4.14: Solução apresentada para o problema 'Maior Multiplo', onde o *feedback* textual foi classificado como 'TOTALMENTE aplicável'

```
1 m = int(input())
2 n = int(input())
3 maior = 0
4 for x in range(0,n):
5     if (x%m == 0) & (x > maior):
6         maior = x
7 print(maior)
```

"*Código parcialmente correto. Observe os limites de sua instrução de repetição.*"

(*Feedback* gerado para o Código 4.14 e classificado como 'TOTALMENTE aplicável'.)

Para o Código 4.14, o avaliador observou que o código falhava para algumas entradas, foi fornecida então a indicação sobre como o aluno poderia corrigir sua solução.

Código Fonte 4.15: Solução apresentada para o problema 'Fatorial', onde o *feedback* textual foi classificado como 'TOTALMENTE aplicável'

```
1 a = int(input())
2 c = int(input())
3 d = input()
4 b = a
5 fatorial = 1
6 while b > 0:
7     fatorial = fatorial*b
8     b = b - 1
9 print (fatorial)
10 b = c
11 fatorial = 1
12 while b > 0:
13     fatorial = fatorial*b
14     b = b - 1
15 print  (fatorial)
16 b = d
17 fatorial = 1
18 while b==-1:
19     break
```

"*Os valores dos fatorias são calculados corretamente. Contudo, a solicitação pede que sejam calculados fatoriais até uma determinada condição, o código executa os cálculos três vezes e finaliza. Código com muitas linhas, solução pode ser simplificada.*"

(*Feedback* gerado para o Código 4.15 e classificado como 'TOTALMENTE aplicável'.)

Para o Código 4.15, o avaliador observou que o código falhava em relação à condição de parada, foram fornecidas a indicação do problema em relação a solicitação do enunciado e descrições para melhoria do código.

Conclusões sobre os resultados e análise dos *feedbacks*

Os resultados obtidos ao longo desta investigação demonstram que a abordagem proposta é promissora. Os critérios de correção de um especialista podem ser capturados, com um bom grau de proximidade, por um conjunto de medidas da Engenharia de Software. Desta forma, as avaliações das soluções podem ser generalizadas com um bom grau de corretude, minimizando o esforço necessário para proceder com esta atividade.

Os códigos utilizados ao longo da investigação apresentavam diferentes níveis de corretude funcional e poderiam ser sintaticamente corretos ou incorretos. Embora códigos incorretos sintaticamente prejudiquem a execução da abordagem, mesmo com estes códigos entre as soluções utilizadas, a abordagem demonstrou um bom grau de corretude.

Apesar de serem necessárias melhorias para a abordagem proposta, as concordâncias obtidas entre notas geradas e as fornecidas por avaliadores possuem um bom nível de concordância, e os *feedbacks* textuais gerados foram, em sua maioria, classificados como 'PARCIALMENTE aplicável' ou 'TOTALMENTE aplicável'. Acreditamos que a pesquisa possui relevância e contribui para a área de informática na educação.

4.5.7 Ameaças à validade

A condução do experimento adotou restrições mais rigorosas em comparação ao quasi experimento descrito anteriormente. As mesmas questões de pesquisa foram investigadas, sendo mantidas também as hipóteses relacionadas, tendo estas sido estabelecidas antes do início da análise. As respostas apresentadas se baseiam nos resultados obtidos.

Neste experimento, se apresentam ameaças à validade externa e ameaças à validade de constructo. Sendo as ameaças à validade externa similares àquelas existentes no quasi experimento, o quantitativo de avaliadores especialistas e a utilização de códigos apenas na linguagem *Python*, não permitem que os resultados apresentados ao longo do trabalho possam ser generalizados para todos os outros contextos. O quantitativo de avaliadores especialistas que participaram do experimento é pequeno, se comparado ao universo de avaliadores no contexto de ensino e aprendizagem de programação introdutória. Sendo assim, os resultados exibidos podem não ser válidos para uma população mais geral. A utilização de códigos na linguagem *Python* foi mantida, apenas códigos escritos nesta linguagem foram observados. Sendo assim, uma vez que códigos escritos em outra linguagem podem alterar os valores das medidas e, consequentemente, afetar a forma como os agrupamentos são gerados, os resultados obtidos nesta pesquisa podem ser distintos com a mudança da linguagem utilizada na implementação das soluções dos discentes.

No que se refere as ameaças à validade de constructo, tem-se que a execução do experimento com a identificação das propriedades baseada na relação critérios e métricas depende do correto mapeamento entre os critérios selecionados pelos avaliadores e as métricas adotadas na geração dos agrupamentos. Desta forma, os resultados obtidos nesta pesquisa podem ser distintos com a mudança dos relacionamentos citados.

Tabela 4.14: Relação entre critérios e métricas, para problemas classificados na categoria 'Repetição'

Critério	**Métricas**
Aplicabilidade da estrutura de repetição (ex. for vs. while)	Corretude funcional, Complexidade ciclomática
Quantidade de estruturas de repetição	Complexidade ciclomática
Profundidade das repetições	Complexidade ciclomática
Presença/ausência de repetições aninhadas	Complexidade ciclomática
Eficiência em relação ao tempo de processamento	nenhuma métrica
Eficiência em relação ao uso de memória	Número de operandos distintos

Tabela 4.15: Resultados obtidos para avaliação geral dos problemas, para os avaliadores especialistas professores, utilizando execução relacionada a identificação das propriedades baseada na relação critério e métrica.

Kappa de Cohen			**Correl. de Pearson**			**Dist. total**		
Máx.	Mín.	Méd.	Máx.	Mín.	Méd.	Máx.	Min.	Méd.
1.00	0.54	0.80	1.00	0.20	0.80	40	0	11.53

Tabela 4.16: Resultados obtidos para avaliação geral dos problemas, para os avaliadores especialistas professores, utilizando execução baseada na seleção automática de métricas.

Kappa de Cohen			**Correl. de Pearson**			**Dist. total**		
Máx.	Mín.	Méd.	Máx.	Mín.	Méd.	Máx.	Min.	Méd.
1.00	0.68	0.88	1.00	0.67	0.92	13	0	2.5

Tabela 4.17: Resultados obtidos para avaliação geral dos problemas, para os avaliadores especialistas monitores, utilizando execução relacionada a identificação das propriedades baseada na relação critério e métrica.

Kappa de Cohen			**Correl. de Pearson**			**Dist. total**		
Máx.	Mín.	Méd.	Máx.	Mín.	Méd.	Máx.	Min.	Méd.
1.00	0.34	0.64	1.00	0.38	0.74	62	0	18.66

Tabela 4.18: Resultados obtidos para avaliação geral dos problemas, para os avaliadores especialistas monitores, utilizando execução baseada na seleção automática de métricas.

Kappa de Cohen			**Correl. de Pearson**			**Dist. total**		
Máx.	Mín.	Méd.	Máx.	Mín.	Méd.	Máx.	Min.	Méd.
1.00	0.54	0.79	1.00	0.37	0.86	25	0.00	8.7

Tabela 4.19: Concordâncias entre avaliadores professores, considerando todas as avaliações.

Kappa de Cohen			
Avaliador	**id:1**	**id:5**	**id:19**
id:1	- -	0.02	0.12
id:5	0.02	- -	0.15
id:19	0.12	0.15	- -
Média	**0.07**	**0.08**	**0.13**
Avaliador artificial critério-métrica	**0.65**	**0.88**	**0.87**
Avaliador artificial seleção automática	**0.80**	**0.93**	**0.95**

Tabela 4.20: Concordâncias entre avaliadores monitores, considerando todas as avaliações.

Kappa de Cohen						
Avaliador	**id:3**	**id:6**	**id:17**	**id:25**	**id:26**	**id:27**
id:3	- -	0.08	0.15	0.11	0.17	0.09
id:6	0.08	- -	0.11	0.10	0.13	0.06
id:17	0.15	0.11	- -	0.17	0.15	0.05
id:25	0.11	0.10	0.17	- -	0.15	0.14
id:26	0.17	0.13	0.15	0.15	- -	0.22
id:27	0.09	0.06	0.05	0.14	0.22	- -
Média	**0.12**	**0.09**	**0.14**	**0.13**	**0.16**	**0.11**
Avaliador artificial critério-métrica	**0.81**	**0.62**	**0.67**	**0.55**	**0.63**	**0.61**
Avaliador artificial seleção automática	**0.87**	**0.72**	**0.79**	**0.70**	**0.99**	**0.71**

Tabela 4.21: Correlação entre as avaliações de professores e seus pares, considerando todas as avaliações.

Correlação de Pearson			
Avaliador	**id:1**	**id:5**	**id:19**
id:1	- -	0.40	0.67
id:5	0.40	- -	0.52
id:19	0.67	0.52	- -
Média	**0.53**	**0.46**	**0.59**

Tabela 4.22: Distância de notas entre as avaliações de cada avaliador especialista professor e seus pares considerando todas as avaliações.

Distância de notas			
Avaliador	**id:1**	**id:5**	**id:19**
id:1	- -	256	181
id:5	256	- -	184
id:19	181	184	- -
Média	**218.5**	**220**	**182.5**

Tabela 4.23: Correlação entre as avaliações de cada avaliador especialista monitor e seus pares considerando todas as avaliações.

Correlação de Pearson						
Avaliador	**id:3**	**id:6**	**id:17**	**id:25**	**id:26**	**id:27**
id:3	- -	0.43	0.51	0.55	0.64	0.68
id:6	0.43	- -	0.35	0.56	0.48	0.59
id:17	0.51	0.35	- -	0.57	0.60	0.53
id:25	0.55	0.56	0.57	- -	0.64	0.75
id:26	0.64	0.48	0.60	0.64	- -	0.76
id:27	0.68	0.59	0.53	0.75	0.76	- -
Média	**0.56**	**0.48**	**0.51**	**0.61**	**0.62**	**0.66**

Tabela 4.24: Distância de notas entre as avaliações de cada avaliador especialista monitor e seus pares considerando todas as avaliações.

Distância de notas						
Avaliador	**id:3**	**id:6**	**id:17**	**id:25**	**id:26**	**id:27**
id:3	- -	263	355	258	231	200.5
id:6	263	- -	427	261	278	214.5
id:17	355	427	- -	317	308	357.5
id:25	258	261	317	- -	240	185.5
id:26	231	278	308	240	- -	177.5
id:27	200.5	214.5	357.5	185.5	177.5	- -
Média	**261.5**	**288.7**	**352.9**	**252.3**	**246.9**	**227.1**

Tabela 4.25: Resumo de dados em relação a métrica de distância de notas na primeira execução do experimento

	Rodada 1 - Distância entre notas						
	Mínimo	1^o Quartil.	Mediana	Média.	3^o Quartil	Máximo	Desvio padrão
Monitores	0.00	7.37	16.00	18.66	24.62	62.00	13.89
Professores	0.00	0.50	7.25	11.53	17.00	40.00	14.01

Tabela 4.26: Resumo de dados em relação a métrica de distância de notas na segunda execução do experimento

	Rodada 2 - Distância entre notas						
	Mínimo	1^o Quartil.	Mediana	Média.	3^o Quartil	Máximo	Desvio padrão
Monitores	0.00	2.87	7.50	8.76	13.25	25.00	7.42
Professores	0.00	0.00	2.50	4.11	5.75	13.00	4.49

Tabela 4.27: Classificação de *feedbacks* subjetivos realizada pelos avaliadores professores

Classificação de *feedbacks* subjetivos (Professor)				
	NÃO aplicável	**PARCIALMENTE aplicável**	**TOTALMENTE aplicável**	**Não informada**
Avaliador artificial critério-métrica	32	42	59	47
Avaliador artificial seleção automática	29	56	55	40

Tabela 4.28: Classificação de *feedbacks* subjetivos realizada pelos avaliadores monitores

Classificação de *feedbacks* subjetivos (Monitor)				
	NÃO aplicável	**PARCIALMENTE aplicável**	**TOTALMENTE aplicável**	**Não informada**
Avaliador artificial critério-métrica	118	86	128	28
Avaliador artificial seleção automática	103	110	117	30

Capítulo 5

Considerações Finais

5.1 Contribuições

As contribuições relacionadas ao desenvolvimento desta tese são sumarizadas nesta seção. São apresentadas as contribuições diretamente derivadas da pesquisa descrita, os trabalhos científicos criados ao longo da pesquisa e as possíveis aplicações onde a abordagem pode ser utilizada após adequações aos respectivos contextos.

A principal contribuição derivada da presente pesquisa consiste na possibilidade de generalizar os *feedbacks* (objetivo e subjetivo) utilizando *clustering* de códigos. Apesar de ainda serem necessários ajustes, com este resultado é possível prover o *feedback* desejado sem atrasos, de modo que este processo não onere os educadores.

A possibilidade de utilizar métricas de software como um meio de representar critérios de avaliação, adotados por avaliadores (professores e monitores/ex-monitores), também se mostra como uma importante contribuição. Utilizando este resultado, é possível propor novas abordagens e ferramentas que explorem essa característica para apoiar o ensino e aprendizagem de programação.

Os critérios adotados por avaliadores (professores e monitores/ex-monitores), em relação ao processo de avaliação de códigos, foram comparados e se observou que estes são distintos para cada avaliador e podem ser modificados ao longo do tempo considerando um mesmo avaliador. Com esta observação, derivada desta pesquisa, sugere-se que é importante considerar estas variações ao propor e desenvolver soluções para o contexto de avaliação em ambientes educacionais de programação.

Os trabalhos científicos, citados na Seção 5.1.1, correspondem aos artigos produzidos pelo autor da tese, sejam estes durante a definição do problema que seria estudado, ou sejam produções relacionadas às etapas da pesquisa. Também são citados os trabalhos de conclusão de curso e projetos de extensão desenvolvidos sob orientação do pesquisador, os quais possuem relação com desenvolvimento dos temas abordados na tese.

Na Seção 5.1.2, são brevemente descritas as possíveis aplicações do método proposto. São listadas propostas onde a aplicação poderia auxiliar, ou agilizar, os processos relacionados à avaliação em ambientes educacionais de programação.

5.1.1 Trabalhos científicos

Os seguintes artigos científicos completos foram produzidos pelo autor da tese, ao longo do desenvolvimento da pesquisa:

- BARBOSA, A. A.; COSTA, E. B.; BRITO, P. H., **Adaptive clustering of codes for assessment in introductory programming courses** In: 14th International Conference on Intelligent Tutoring Systems (ITS 2018), 2018, Montreal/Canada. Proceedings of the 14th International Conference on Intelligent Tutoring Systems.

- BARBOSA, A. A.; BRITO, P. H.; COSTA, E. B.. **Uma abordagem adaptativa para gerar agrupamento de códigos em disciplinas de programação introdutória** In: XXVIII Simpósio Brasileiro de Informática na Educação (SBIE 2017), 2017, Recife/PE. Anais do Simpósio Brasileiro de Informática na Educação, 2017.

- BARBOSA, A. A.; COSTA, E. B.. **Simulated learners in peers assessment for introductory programming courses** In: 2nd Workshop on Simulated Learners (WSL), 2015, Madrid. Proceedings of the Workshops at the 17th International Conference on Artificial Intelligence in Education AIED 2015. v.5.

- BARBOSA, A. A.; COSTA, E. B.; FERREIRA, D. I. S. **Influência da linguagem no ensino introdutório de programação** In: Simpósio Brasileiro de Informática na Educação (SBIE), 2014, Dourados. Anais do 25o Simpósio Brasileiro de Informática na Educação (SBIE), 2014.

Os seguintes trabalhos de conclusão de curso foram desenvolvidos sob orientação do pesquisador autor desta tese e possuem relação com o tema abordado na pesquisa:

- Roberto Bartolomeu Fragoso Neto, **Interface para um ambiente virtual de aprendizagem focado em programação**, 2017
- Igor Simões de Oliveira Lima, **Sentinela: Uma ferramenta para monitorar sessões de estudo em disciplinas introdutórias de programação**, 2016
- João Vitor Silva Oliveira, **Investigando o uso de múltiplas soluções de referência para classificar códigos usando algoritmos de similaridade**, 2016
- José Arnóbio de Oliveira Júnior, **Perfis de jogadores em contextos de ensino/aprendizagem**, 2016
- Allan Lucio Correia, **Safira: Um sistema para apoiar a abordagem de avaliação por pares em disciplinas introdutórias de programação**, 2016
- Danilo Victor Barbosa da Costa, **Uso de algoritmos de similaridades para classificar códigos de acordo com a taxonomia SOLO em disciplinas de programação introdutória.**, 2016

Os seguintes artigos foram produzidos com base nos trabalhos de conclusão de curso desenvolvidos sob orientação do pesquisador autor desta tese e possuem relação com o tema abordado na pesquisa:

- BARBOSA, A. A.; OLIVEIRA, J. V. S. **Investigando o uso de múltiplas soluções de referência para classificar códigos usando algoritmos de similaridade** In: II Congresso sobre Tecnologias na Educação (Ctrl+E 2017), 2017, Mamanguape/PB. Anais do II Congresso sobre Tecnologias na Educação (Ctrl + E 2017), 2017.
- OLIVEIRA JUNIOR, J. A.; BARBOSA, A. A.. **Perfis de jogadores em contextos de ensino/aprendizagem em disciplinas de programação** In: II Workshop de Ensino em Pensamento Computacional, Algoritmos e Programação, 2016, Uberlândia/MG. Anais dos Workshops do Congresso Brasileiro de Informática na Educação, 2016.

- BARBOSA, A. A.; COSTA, D. V. B.; CORREIA, A. L.; MOURA, D. L. L.; COSTA, E. B. Uso de algoritmos de similaridade para classificar códigos de acordo com a taxonomia SOLO em disciplinas de programação introdutória In: I Workshop de Ensino em Pensamento Computacional, Algoritmos e Programação, 2016, Uberlândia/MG. Anais dos Workshops do Congresso Brasileiro de Informática na Educação, 2016. (Prêmio de 2º melhor artigo do evento).

- CORREIA, A. L.; COSTA, D. V. B.; BARBOSA, A. A.; **Uso de avaliação por pares em disciplinas introdutórias de programação** In: Workshop sobre Educação em Informática (WEI), 2015, Recife. XXXV CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO: A internet de tudo toda observada. Porto Alegre, RS: Sociedade Brasileira de Computação (SBC), 2015.

Os seguintes projetos foram coordenados pelo pesquisador autor desta tese e possuem relação com o tema abordado nesta tese:

- 2016 - 2017. **Ensino de programação: divulgação da atividade de programação e avaliação de ferramentas de apoio ao ensino de programação.** 3 alunos de graduação (Bolsistas). Apoio financeiro da Universidade Federal de Alagoas, através de bolsas de extensão.

- 2014 - 2015. **Ensino de programação: Identificando problemas e propondo soluções**. 3 alunos de graduação (Bolsistas). Apoio financeiro da Universidade Federal de Alagoas, através de bolsas de extensão.

5.1.2 Possíveis aplicações do método proposto

A abordagem de agrupamento proposta nesta tese pode ser útil em diversos contextos, algumas possibilidades são descritas a seguir.

Nas teses de doutorado de Matheus Gaudêncio [22], Elena Glassman [29] e Eliane Araújo [16], pesquisas com objetivos relacionados à presente tese foram desenvolvidas. Todas estas pesquisas focam no contexto de ensino e aprendizagem de programação. O desenvolvimento de soluções que empreguem os produtos oriundos de cada uma destas teses pode ser realizado

através de projetos de pesquisa desenvolvidos como colaboração entre os pesquisadores em suas respectivas universidades.

O processo de tentativas de solução realizado por um aluno pode ser avaliado utilizando a abordagem proposta. A cada nova submissão, a solução proposta pelo aluno pode ser classificada em um dos grupos determinados em execuções anteriores da abordagem e, com isso, obter o *feedback* generalizado sem nenhum atraso. O conjunto de soluções pode ainda ser ordenado cronologicamente, para que relações entre as soluções sejam criadas com base nos grupos aos quais foram associadas, auxiliando o professor na compreensão do raciocínio utilizado pelo discente.

Ambientes *Massive Open Online Courses* (*MOOC*s) possuem como um de seus principais problemas as atividades de avaliação, uma vez que existem milhares de alunos e uma pequena equipe de professores e assistentes. Sendo assim, em tais ambientes, é inviável proceder com meios tradicionais de avaliação. Desta forma, para cursos onde o foco é a programação, geralmente são utilizados testes automatizados ou a abordagem de avaliação por pares (*peer assessment*) [19; 68; 61; 52; 37]. Contudo, o uso de avaliação por pares possui algumas limitações e dificuldades, uma destas ocorre com a verificação da validade de uma avaliação. Algumas propostas, tais como os trabalhos [52; 37], visam criar avaliações confiáveis para servir como base para classificar avaliações realizadas pelos alunos do curso. Abordagem de agrupamento proposta nesta tese pode apoiar a atividade de avaliação por pares e servir como meio complementar as abordagens exibidas em [52; 37], de modo que as avaliações confiáveis possam ser generalizadas, reduzindo a carga de trabalho da equipe de educadores.

Em [28], é descrito que a performance de um estudante é influenciada por seus pares. A proposta apresentada em [14] descreve competências necessárias para criação de um aluno simulado, o qual pode servir como um par influenciador de aprendizado em ambientes virtuais de aprendizagem. Uma das competências apresentadas é relacionada à avaliação de soluções, sendo assim, o método proposto pode servir como meio para prover o *feedback* necessário na proposta de aluno simulado discutida.

Uma vez que a avaliação de códigos é uma atividade demorada, sujeita ao viés e aos erros do avaliador, o método proposto pode servir como um assistente de avaliação. Utilizando técnicas de inteligência artificial, o assistente pode, por exemplo, apresentar casos

anteriores e similares que já foram avaliados, dessa forma, o professor pode simplesmente acatar a avaliação proposta e agilizar o processo. Através do uso da abordagem, após um período de tempo, acreditamos que a maior parte das avaliações só precise ser confirmada pelo professor, somente soluções inovadoras necessitariam de novas avaliações.

Diversas abordagens de apoio aos processos de ensino e aprendizagem de programação já foram propostas, uma das mais utilizadas são os juízes online. O método proposto pode ser adotado como um complemento a estas abordagens, possibilitando que algumas correções sejam efetuadas pelo juiz, enquanto outras sejam realizadas por um professor. A avaliação fornecida pelo juiz online pode ainda ser complementada com o *feedback* de avaliações anteriores, criado pelo professor, ou ter um grau de qualidade avaliado em relação ao grupo em que a solução foi classificada.

As descrições apresentadas são algumas das possibilidades onde o método proposto pode ser aplicado, outros usos poderão surgir à medida em que o trabalho evoluir.

5.2 Trabalhos futuros

Neste capítulo, são apresentados os trabalhos futuros, estes podem dar continuidade à pesquisa apresentada nesta tese, de modo a melhorar a abordagem proposta.

A adoção de novas métricas de software se mostra necessária, pois é necessário ampliar as métricas aplicáveis aos problemas classificados como 'Condicionais' e 'Repetição'. A métrica utilizada com mais frequência na análise destes problemas é a complexidade ciclomática. Contudo, essa métrica não permite, por exemplo, diferenciar as estruturas condicionais ou de repetição utilizadas.

O método proposto se baseia na extração de medidas a partir de códigos. O uso de códigos escritos em outras linguagens, diferentes de *Python*, pode alterar as propriedades extraídas dos códigos. Dessa forma, é desejável observar a sensibilidade do método para, ao menos, uma outra linguagem.

Vale ainda observar que, como o método se baseia na extração de medidas, ele é aplicável a códigos sintaticamente corretos, códigos sintaticamente incorretos podem ser analisados de forma a reduzir a qualidade dos resultados. Essa característica torna a utilidade da abordagem limitada a um grupo de códigos e discentes nas disciplinas alvo. Isto posto, é necessário

pesquisar métricas, ou uma solução complementar a abordagem, que sejam efetivas para uso em códigos com sintaxe incorreta.

O algoritmo *Kmeans* foi o único algoritmo utilizado nesta tese. Soluções que apresentem características distintas daquelas analisadas inicialmente ao longo da execução da abordagem serão inseridas em um grupo com elementos não muito similares. Outros algoritmos de agrupamento tratam esta situação de outras maneiras. Dessa forma, deseja-se investigar o uso de outras técnicas de agrupamento. Algumas técnicas de classificação e agrupamento já estão disponíveis para condução da pesquisa, citamos o algoritmo KNN como um exemplo, mas seu uso ainda não foi investigado. Além disso, pode ser necessário pesquisar técnicas de aprendizagem de máquina adicionais, pois outras técnicas desconhecidas pelo autor podem ser mais adequadas ao método.

A execução da abordagem onde ocorre a identificação automática das métricas demonstraram um alto custo de computação (memória). Sendo assim, pode-se afirmar que é necessário minimizar custo computacional do método, buscando alternativas para implementação do oráculo de identificação de propriedades, uma das etapas de execução do método.

5.3 Conclusões

Nesta teste de doutorado, foi proposta uma abordagem baseada em agrupamento de códigos como meio para minimizar o esforço dispendido na avaliação em disciplinas de programação introdutória. Compreendendo avaliação como o processo de fornecer um *feedback* composto por uma classificação/nota e uma informação textual/explicação para o aluno.

O ensino e aprendizagem de programação possuem fundamental importância para área de computação. Como descrito anteriormente, tais conhecimentos são a base para compreensão de diversos outros conceitos. O cenário problemático encontrado em relação a este contexto justifica a existência de tantas pesquisas sobre o tema. Propostas de abordagens, ferramentas ou técnicas que minimizem as dificuldades existentes nos ambientes de ensino e aprendizagem de programação são facilmente encontradas em eventos ou periódicos da área.

Uma das principais dificuldades em ambientes de ensino está relacionada com as atividades de avaliação. No contexto de programação, a avaliação das soluções propostas tem dificuldades adicionais, como já discutido anteriormente. Pesquisas que tenham como foco

melhorar, facilitar ou agilizar atividades avaliativas são necessárias. A presente pesquisa possui este foco, e se mostra relevante pois propõe um método com características não exploradas por pesquisas anteriores.

Todas as avaliações da proposta foram realizadas utilizando um conjunto de notas (valores entre 0 e 10) e *feedbacks* textuais, fornecidos por avaliadores humanos. Estes avaliadores possuíam experiência na área de ensino e aprendizagem de programação, desempenhando a função de professor ou monitor de disciplinas introdutórias.

Nesta insvestigação, a concordância obtida entre notas geradas e as fornecidas por um avaliador, adotando-se a medida estatística *Kappa de Cohen*, variou de uma concordância razoável, no pior caso analisado, para uma concordância perfeita, no melhor caso analisado. Os critérios de correção de um especialista podem ser capturados, com um bom grau de proximidade, por um conjunto de medidas da Engenharia de Software e as avaliações de soluções pertencentes a um mesmo grupo se mostraram bastante próximas. As concordâncias obtidas entre o avaliador artificial e avaliadores humanos é superior àquela obtida entre pares de avaliadores humanos.

Os *feedbacks* textuais gerados foram, em sua maioria, classificados como 'PARCIALMENTE aplicável' ou 'TOTALMENTE aplicável', independente do código ser avaliado como correto, parcialmente correto ou incorreto. Acreditamos que ajustes possibilitarão uma melhoria dos *feedbacks* gerados e, consequentemente, de sua classificação por parte dos avaliadores.

Embora sejam necessários ajustes em relação a abordagem proposta, os resultados obtidos ao longo desta investigação demonstram que é possível utilizar o método baseado em agrupamento de códigos como meio para minimizar o esforço de avaliação.

Bibliografia

[1] Leônidas S Barbosa, Teresa CB Fernandes, and André MC Campos. Takkou: Uma ferramenta proposta ao ensino de algoritmos. In *Anais do CSBC/WEI*, 2011.

[2] Ricardo Barcelos, Liane Tarouco, and Magda Bercht. O uso de mobile learning no ensino de algoritmos. *Revista Novas Tecnologias na Educação (RENOTE)*, 7(3):327–337, Dezembro 2009.

[3] Jacob Benesty, Jingdong Chen, Yiteng Huang, and Israel Cohen. *Pearson Correlation Coefficient*, pages 1–4. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[4] S D Benford, E K Burke, E Foxley, and C A Higgins. The ceilidh system for the automatic grading of students on programming courses. In *Proceedings of the 33rd Annual on Southeast Regional Conference*, ACM-SE 33, pages 176–182, New York, NY, USA, 1995. Association for Computing Machinery (ACM).

[5] Lauren R. Biggers and Nicholas A. Kraft. Quantifying the similiarities between source code lexicons. In *Proceedings of the 49th Annual Southeast Regional Conference*, ACM-SE '11, pages 80–85, New York, NY, USA, 2011. Association for Computing Machinery (ACM).

[6] Rodrigo. L. B. L. Campos. Metodologia erm2c: Para melhoria do processo de ensino aprendizagem de lógica de programação. In *Anais do XXX Congresso da Sociedade Brasileira de Computação, Workshop de Educação em Informática (CSBC 2010 - WEI 2010)*, Minas Gerais, Brasil, Julho 2010.

[7] J. Castro, R. Yera Toledo, and L. MartÃnez. A fuzzy approach for natural noise management in group recommender systems. *Expert Systems with Applications*, pages 237–249, 2018.

[8] José Osvaldo M Chaves, Angélica F Castro, Rommel W Lima, Marcos Vinicius a Lima, and Karl H a Ferreira. Integrando Moodle e Juízes Online no Apoio a Atividades de Programação. In *Anais do Simpósio Brasileiro de Informática na Educação*, pages 244–254, 2013.

[9] Rohan Roy Choudhury, Hezheng Yin, Joseph Moghadam, Antares Chen, and Armando Fox. Autostyle: Scale-driven hint generation for coding style. In *Proceedings of the 13th International Conference on Intelligent Tutoring Systems*, ITS 201, pages 122–132, 2016.

[10] J. Cohen. Weighted kappa: Nominal scale agreement with provision for scaled disagreement or partial credit. *Psychological Bulletin*, 70:213–220, 1968.

[11] Sébastien Combéfis and Arnaud Schils. Automatic programming error class identification with code plagiarism-based clustering. In *Proceedings of the 2Nd International Code Hunt Workshop on Educational Software Engineering*, CHESE 2016, pages 1–6, New York, NY, USA, 2016. Association for Computing Machinery (ACM).

[12] Daryl D Souza, Margaret Hamilton, and Michael C. Harris. Software development marketplaces: Implications for plagiarism. In *Procceedings of the 9th Australasian Conf. on Computing Education*, pages 27–33, Darlinghurst, Australia, 2007.

[13] Alexandre de Andrade Barbosa, Allan Lucio Correia, and Evandro de Barros Costa. Um mapeamento sistemático sobre analisadores de código em disciplinas de programação. In *Anais do Simpósio Brasileiro de Informática na Educação*, 2015.

[14] Alexandre de Andrade Barbosa and Evandro de Barros Costa. Simulated learners in peers assessment for introductory programming courses. In *Proceedings of the Workshops at the 17th International Conference on Artificial Intelligence in Education, AIED 2015, Madrid, Spain, June 22 + 26, 2015.*, 2015.

[15] Alexandre de Andrade Barbosa, Dyego Ítalo Ferreira, and Evandro de Barros Costa. Influência da linguagem no ensino introdutório de programação. In *Anais do Congresso Brasileiro de Informática na Educação/Simpósio Brasileiro de Informática na Educação*, pages 612–621, 2014.

[16] Eliane Cristina de Araujo. *Automatização de Feedback para Apoiar o Aprendizado no Processo de Resolução de Problemas de Programação*. Tese de doutorado, Universidade Federal de Campina Grande, set. 2017.

[17] Eliane Cristina de Araujo, Dalton Serey, and Jorge Figueiredo. Qualitative aspects of students programs: Can we make them measurable? In *2016 IEEE Frontiers in Education Conference (FIE)*, pages 1–8, Oct 2016.

[18] Eliane Cristina de Araujo, Dalton Serey, Matheus Gaudencio, and Jorge Figueiredo. Applying spectrum based fault localization on novices programs. In *2016 IEEE Frontiers in Education Conference (FIE)*, pages 1–8, Oct 2016.

[19] Michael de Raadt, David Lai, and Richard Watson. An evaluation of electronic individual peer assessment in an introductory programming course. In *Procceedings of the Seventh Baltic Sea Conference on Computing Education Research - Volume 88*, Koli Calling, pages 53–64, Darlinghurst, Australia, 2007.

[20] M. Delamaro, M. Jino, and J. Maldonado. *Introdução Ao Teste De Software*. Elsevier Brasil, 2013.

[21] Matheus Gaudencio do Rego. Repositório do pycomparecode. `https://github.com/matheusgr/pycomparecode`. Último acesso em fevereiro de 2017.

[22] Matheus Gaudencio do Rego. *Feedback através da Comparação de Códigos no Apoio ao Processo de Ensino Aprendizagem de Introdução à Programação*. Tese de doutorado, Universidade Federal de Campina Grande, nov. 2015.

[23] Alexandre N. Duarte, Hugo Moreira, and Thiago Silva Mello. Competitividade como fator motivacional para o estudo de computação. In *Anais Simpósio Brasileiro de Informática na Educação*, João Pessoa, PB, Brasil, Novembro 2010.

[24] Stephen H. Edwards and Manuel A. Perez-Quinones. Web-cat: Automatically grading programming assignments. *SIGCSE Bull.*, 40(3):328–328, June 2008.

[25] Katti Faceli, Ana C Lorena, João Gama, and ACPLF Carvalho. *Inteligência Artificial: Uma Abordagem de Aprendizado de Máquina*. Livros Técnicos e Científicos, 2011.

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

[27] George E. Forsythe and Niklaus Wirth. Automatic grading programs. *Communications of the Association for Computing Machinery (ACM)*, 8(5):275–278, May 1965.

[28] Stephanie Frost and Gordon I. McCalla. Exploring through simulation the effects of peer impact on learning. In *Procceedings of the Workshops at the 16th International Conference on Artificial Intelligence in Education AIED 2013, Memphis, USA, July 9-13, 2013*, 2013.

[29] Elena L. Glassman. *Clustering and Visualizing Solution Variation in Massive Programming Classes*. Tese de doutorado, Massachusetts Institute of Technology, set. 2016.

[30] Elena L. Glassman, Aaron Lin, Carrie J. Cai, and Robert C. Miller. Learnersourcing personalized hints. In *Proceedings of the 19th Association for Computing Machinery Conference on Computer-Supported Cooperative Work & Social Computing*, CSCW '16, pages 1626–1636, New York, NY, USA, 2016. Association for Computing Machinery (ACM).

[31] Robert Goldblatt and Marcel Jackson. Well-structured program equivalence is highly undecidable. *Association for Computing Machinery Transactions of Computer Logic*, 13(3):26:1–26:8, August 2012.

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

[33] Andrew Head, Elena Glassman, Gustavo Soares, Ryo Suzuki, Lucas Figueredo, Loris D'Antoni, and Björn Hartmann. Writing reusable code feedback at scale with mixed-initiative program synthesis. In *Proceedings of the Fourth (2017) Association for Computing Machinery Conference on Learning @ Scale*, L@S '17, pages 89–98, New York, NY, USA, 2017. Association for Computing Machinery (ACm).

[34] J. B. Hext and J. W. Winings. An automatic grading scheme for simple programming exercises. *Communications of the Association for Computing Machinery (ACM)*, 12(5):272–275, May 1969.

[35] Michael J. Hull, Daniel Powell, and Ewan Klein. Infandango: Automated grading for student programming. In *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education*, ITiCSE '11, pages 330–330, New York, NY, USA, 2011. Association for Computing Machinery (ACM).

[36] Stavros Konstantinidis. Computing the edit distance of a regular language. *Information and Computation*, 205(9):1307–1316, 2007.

[37] Chinmay Kulkarni, Koh Pang Wei, Huy Le, Daniel Chia, Kathryn Papadopoulos, Justin Cheng, Daphne Koller, and Scott R. Klemmer. Peer and self assessment in massive online classes. *Association for Computing Machinery Transactions of Computer Human Interaction*, 20(6):33:1–33:31, December 2013.

[38] Vladimir Iosifovich Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10, 1966.

[39] Sihan Li, Xusheng Xiao, Blake Bassett, Tao Xie, and Nikolai Tillmann. Measuring code behavioral similarity for programming and software engineering education. In *Proceedings of the 38th International Conference on Software Engineering Companion*, ICSE '16, pages 501–510, New York, NY, USA, 2016. Association for Computing Machinery (ACM).

[40] T.J. MacCabe, McCabe, Associates, and IEEE Computer Society. *Structured testing*. Tutorial Texts Series. IEEE Computer Society Press, 1983.

[41] Raina Mason and Graham Cooper. Introductory programming courses in australia and new zealand in 2013 - trends and reasons. In *Procceedings of the Sixteenth Australasian Computing Education Conference - Volume 148*, ACE, pages 139–147, Darlinghurst, Australia, 2014.

[42] Raina Mason, Graham Cooper, and Michael de Raadt. Trends in introductory programming courses in australian universities: Languages, environments and pedagogy. In *Procceedings of the Fourteenth Australasian Computing Education Conference - Volume 123*, ACE, pages 33–42, Darlinghurst, Australia, 2012.

[43] Michael McCracken, Vicki Almstrum, Danny Diaz, Mark Guzdial, Dianne Hagan, Yifat Ben-David Kolikant, Cary Laxer, Lynda Thomas, Ian Utting, and Tadeusz Wilusz. A multi-national, multi-institutional study of assessment of programming skills of first-year cs students. In *Working Group Reports from ITiCSE*, pages 125–180, New York, USA, 2001.

[44] Phatludi Modiba, Vreda Pieterse, and Bertram Haskins. Evaluating plagiarism detection software for introductory programming assignments. In *Proceedings of the Computer Science Education Research Conference 2016*, CSERC '16, pages 37–46, New York, NY, USA, 2016. Association for Computing Machinery (ACM).

[45] Joseph Bahman Moghadam, Rohan Roy Choudhury, HeZheng Yin, and Armando Fox. Autostyle: Toward coding style feedback at scale. In *Proceedings of the Second (2015) Association for Computing Machinery Conference on Learning @ Scale*, L@S '15, pages 261–266, New York, NY, USA, 2015. Association for Computing Machinery (ACM).

[46] Marcelle Pereira Mota, Silvana R. de Brito, Mireille Pinheiro Moreira, and Eloi Luiz Favero. Ambiente integrado a plataforma moodle para apoio ao desenvolvimento das habilidades iniciais de programação. In *Anais do XX Simpósio Brasileiro de Informática na Educação (Simpósio Brasileiro de Informática na Educação 2009)*, Florianópolis, SC, Brasil, Novembro 2009.

[47] Andrzej S. Murawski. About the undecidability of program equivalence in finitary languages with state. *Association for Computing Machinery Transactions of Computer Logic*, 6(4):701–726, October 2005.

[48] Luiz F Noschang, Elieser A Fillipi Pelz, and André LA Raabe. Portugol studio: Uma ide para iniciantes em programação. In *Anais do CSBC/WEI*, pages 535–545, 2014.

[49] Rodrigo B. Paes, Romero Malaquias, Marcio Guimaraes, and Hyggo Almeida. Ferramenta para a avaliação de aprendizado de alunos em programação de computadores. In *Anais do II Congresso Brasileiro de Informática na Educação (Congresso Brasileiro de Informática na Educação 2013) Workshops (Workshops do Congresso Brasileiro de Informática na Educação 2013)*, Campinas, SP, Novembro 2013.

[50] Fillipi D Pelz, Elieser A de Jesus, and André LA Raabe. Um mecanismo para correção automática de exercícios práticos de programação introdutória. In *Anais do Simpósio Brasileiro de Informática na Educação*, 2012.

[51] Chris Piech. Kmeans. `http://stanford.edu/~cpiech/cs221/handouts/kmeans.html`, 2013. Último acesso em fevereiro de 2017.

[52] Chris Piech, Jonathan Huang, Zhenghao Chen, Chuong B. Do, Andrew Y. Ng, and Daphne Koller. Tuned models of peer assessment in moocs. *CoRR*, abs/1307.2579, 2013.

[53] Roger S. Pressman. *Software Engineering: A Practitioner's Approach*. McGraw-Hill Higher Education, 5th edition, 2001.

[54] André Raabe, Elieser Ademir de Jesus, Andrei Hodecker, and Fillipi Pelz. Avaliação do feedback gerado por um corretor automático de algoritmos. In *Anais do Simpósio Brasileiro de Informática na Educação*, pages 358–366, 2015.

[55] Vinicius Ramos, Mateus Freitas, Mauricio Galimbert, Antonio Carlos Mariani, and Raul Wazlawick. A comparação da realidade mundial do ensino de programação para iniciantes com a realidade nacional: Revisão sistemática da literatura em eventos brasileiros. In *Anais do Simpósio Brasileiro de Informática na Educação*, 2015.

[56] Matheus Gaudencio Rego, Ayla Dantas, and Dalton Serey Guerrero. Can computers compare student code solutions as well as teachers? In *Proceedings of the 45th Association for Computing Machinery Technical Symposium on Computer Science Education*, SIGCSE '14, pages 21–26, New York, NY, USA, 2014. Association for Computing Machinery (ACM).

[57] Joanna C. S. Santos and Admilson R. L. Ribeiro. JOnline: proposta preliminar de um juiz online didático para o ensino de programação. In *Anais do Simpósio Brasileiro de Informática na Educação*, pages 964–967, 2011.

[58] SciPy developers. Scientific computing tools for python. `https://www.scipy.org/about.html`, 2017.

[59] Marlos Tacio Silva, Evandro De Barros Costa, Paulo Henrique Barbosa, and Juliana De Carvalho Cavalcante. Um Arcabouço para Construção de Mecanismos de Análise de Códigos de Programação Introdutória. In *Anais do Simpósio Brasileiro de Informática na Educação*, pages 1083–1092, 2014.

[60] Rishabh Singh, Sumit Gulwani, and Armando Solar-Lezama. Automated feedback generation for introductory programming assignments. In *Procceedings of the 34th Association for Computing Machinery SIGPLAN Conference on Programming Language Design and Implementation*, PLDI, pages 15–26, New York, USA, 2013.

[61] Jirarat Sitthiworachart and Mike Joy. Computer support of effective peer assessment in an undergraduate programming class. *Journal of Computer Assisted Learning*, 24(3):217–231, 2008.

[62] Ian Sommerville. *Software Engineering*. Addison-Wesley Publishing Company, USA, 9th edition, 2010.

[63] Shashank Srikant and Varun Aggarwal. A system to grade computer programming skills using machine learning. In *Proceedings of the 20th Association for Computing Machinery SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 1887–1896, New York, NY, USA, 2014. Association for Computing Machinery (ACM).

[64] Martijn Stegeman, Erik Barendsen, and Sjaak Smetsers. Towards an empirically validated model for assessment of code quality. In *Procceedings of the 14th Koli Calling Int. Conf. on Computing Education Research*, Koli Calling, pages 99–108, New York, USA, 2014.

[65] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining, (First Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005.

[66] The Huxley Team. The huxley. `www.thehuxley.com`.

[67] D. Tsichritzis. The equivalence problem of simple programs. *Journal of the Association for Computing Machinery (ACM)*, 17(4):729–738, October 1970.

[68] Joe Warren, Scott Rixner, John Greiner, and Stephen Wong. Facilitating human interaction in an online programming course. In *Procceedings of the 45th Association for Computing Machinery Technical Symposium on Computer Science Education*, SIGCSE, pages 665–670, New York, USA, 2014.

[69] Szymon Wasik, Maciej Antczak, Jan Badura, Artur Laskowski, and Tomasz Sternal. A survey on online judge systems and their applications. *Association for Computing Machinery Computer Survey*, 51(1):3:1–3:34, January 2018.

[70] Aharon Yadin. Reducing the dropout rate in an introductory programming course. *Association for Computing Machinery Inroads*, 2(4):71–76, 2011.

[71] Hezheng Yin, Joseph Moghadam, and Armando Fox. Clustering student programming assignments to multiply instructor leverage. In *Proceedings of the Second (2015) Association for Computing Machinery Conference on Learning @ Scale*, L@S '15, pages 367–372, New York, NY, USA, 2015. Association for Computing Machinery (ACM).

[72] Susilo Veri Yulianto and Inggriani Liem. Automatic grader for programming assignment using source code analyzer. In *Data and Software Engineering (ICODSE), 2014 International Conference on*, pages 1–4. IEEE, 2014.

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

Apêndice A

Questionário

A.1 Apresentação do questionário

Neste capítulo é apresentado o questionário aplicado na condução da pesquisa Survey, descrita no Capítulo 4. O questionário foi aplicado para dois grupos: professores e monitores/ex-monitores de disciplinas de programação.

A coleta dos dados foi disponibilizada através de formulários criado na plataforma *Google Forms*, os formulários estão disponíveis nos links: 'Formulário - Professor' [1] e 'Formulário - Monitor' [2]

O questionário é composto pelas seguintes seções:

Cabeçalho texto contendo a descrição da pesquisa e informações sobre confidencialidade e anonimato, apresentado no início da pesquisa;

Perfil contém perguntas relacionadas ao perfil geral do respondente;

Perfil - Professor contém perguntas relacionadas a experiência relacionada com a função de professor, apresentado somente aos participantes que já são/foram professores;

Perfil - Monitor contém perguntas relacionadas a experiência relacionada com a função de monitor, apresentado somente aos participantes que já são/foram monitores;

[1] https://docs.google.com/forms/d/1ohzHVz-ovj0WQCzgEUSHep7ohppLp6HPauRbrCQCIQI

[2] https://docs.google.com/forms/d/1VGNVcuYjIgCpkTw_EUt32mJdI7yPw7cjwPBuHAzelPo

Identificação de problemas/dificuldades no contexto de ensino e aprendizagem de programação contém perguntas relacionadas aos possíveis fatores motivadores do cenário problemático relacionado ao ensino e aprendizagem de programação;

Processo de avaliação de códigos contém perguntas que buscam identificar os procedimentos e critérios adotados pelo avaliador, seja professor ou monitor, ao realizar avaliações de códigos;

Continuidade da pesquisa contém perguntas relacionadas com a permissão de contato para etapas posteriores da pesquisa.

Avaliação em disciplinas introdutórias de programação

Obrigado por considerar participar deste survey.

Descrição da pesquisa

O objetivo desta pesquisa é compreender os processos relacionados com a avaliação de códigos em disciplinas introdutórias de programação. A investigação visa identificar quais os problemas, ou dificuldades, podem estar relacionados ao ensino e aprendizagem de programação, sob a ótica do professor/educador. O objetivo é compreender os processos adotados, não se trata de uma avaliação de seus métodos. Responder as perguntas deste questionário deve levar cerca de 10 minutos. Esta pesquisa está sendo conduzida por Alexandre Barbosa (UFCG e UFAL), aluno de doutorado da Universidade Federal de Campina Grande (UFCG), sob orientação do professor Evandro de Barros Costa (UFCG e UFAL).

Sobre confidencialidade e anonimato

Alguns dados de identificação (nome e email) serão solicitados com o objetivo possibilitar o contato do pesquisador para etapas posteriores da pesquisa. Este contato ocorrerá apenas sob sua permissão, ou seja, é possível responder o questionário e informar que não deseja ser contatado posteriormente. Os dados coletados serão utilizados apenas pelos pesquisadores envolvidos, e somente no contexto desta pesquisa. Todos os dados de identificação serão removidos, ou seja, não serão citados nas publicações resultantes da pesquisa. Sua participação neste survey é voluntária, a qualquer momento é possível desistir de enviar os

dados.

Quaisquer dúvidas e solicitações relacionadas com a pesquisa podem ser esclarecidas através do e-mail: [endereço]

Mais uma vez, agradeço por considerar participar deste survey.

Perfil

Nome: _____

Instituição: _____

Nível de formação:

( ) Pós doutorado

( ) Doutorado

( ) Mestrado

( ) Especialização

( ) Graduação

Perfil - Professor

Leciona desde (informe apenas o ano): _____

Quando lecionou uma disciplina introdutória de programação pela última vez (informe apenas o ano): _____

Quais das seguintes linguagens já adotou em disciplina(s) introdutórias de programação:

( ) C

( ) Python

( ) Java

( ) Pascal

( ) C++

Adota atividades práticas na condução da disciplina?

( ) Sim

( ) Não

Adota ferramenta de apoio na condução da disciplina? _____

Qual ou quais ferramenta(s)? _____

_____

Perfil - Monitor

Durante a graduação, foi monitor de alguma disciplina de programação:

( ) Sim

( ) Não

Possui conhecimento em quais das seguintes linguagens:

( ) C

( ) Python

( ) Java

( ) Pascal

( ) C++

Quando foi monitor de uma disciplina introdutória de programação pela última vez (informe apenas o ano): _____

Quais das seguintes linguagens foram adotadas na condução da disciplina introdutória de programação:

( ) C

( ) Python

( ) Java

( ) Pascal

( ) C++

Foram adotadas atividades práticas na condução da disciplina?

( ) Sim

( ) Não

Quais ferramentas de apoio foram utilizadas na condução da disciplina? _____

Identificação de problemas/dificuldades no contexto de ensino e aprendizagem de programação

Considera que existe algum problema ou dificuldade no contexto de ensino e aprendizagem de programação?

( ) Sim

( ) Não

( ) Talvez

Sob a ótica do educador (professor/monitor), quais aspectos da atividade de ensino de programação estão relacionados aos problemas ou dificuldades existentes neste contexto. Enumere os 5 maiores problemas ou dificuldades de acordo com o grau de importância, ou seja, 1-Maior problema/dificuldade e 5- Menor problema/dificuldade.

○1 ○ 2 ○ 3 ○ 4 ○ 5 o pouco tempo disponível para exposição dos conteúdos

○1 ○ 2 ○ 3 ○ 4 ○ 5 o pouco tempo disponível para os alunos compreenderem os conteúdos

○1 ○ 2 ○ 3 ○ 4 ○ 5 a estrutura deficiente, falta de laboratórios ou computadores, para realizar a exposição e solicitar a prática dos conteúdos

○1 ○ 2 ○ 3 ○ 4 ○ 5 a grande quantidade de alunos que impossibilita o acompanhamento adequado e individualizado

○1 ○ 2 ○ 3 ○ 4 ○ 5 a carga de trabalho necessária para criar exercícios e provas

○1 ○ 2 ○ 3 ○ 4 ○ 5 a carga de trabalho necessária para corrigir exercícios e provas

○1 ○ 2 ○ 3 ○ 4 ○ 5 a identificação do nível de aprendizado do aluno com tempo adequado para realizar uma intervenção

○1 ○ 2 ○ 3 ○ 4 ○ 5 a identificação de plágio entre os códigos apresentados como solução de exercícios e provas

○1 ○ 2 ○ 3 ○ 4 ○ 5 a avaliação de aspectos de qualidade dos códigos apresentados como solução de exercícios e provas

○1 ○ 2 ○ 3 ○ 4 ○ 5 o excesso de preocupação com detalhes sintáticos da linguagem usada

outros: _____

Processo de avaliação de códigos

Considerando uma categorização de problemas em três classes, sendo estas:

Problemas básicos, aqueles cujas soluções (códigos) necessitam apenas de um conjunto básico de construções (ex. cálculos matemáticos, entrada e saída, operações de comparação e atribuição);

Problemas de decisão, aqueles cujas soluções (códigos) necessitam de construções básicas e de estruturas de decisão/condicionais;

Problemas de repetição, aqueles cujas soluções (códigos) necessitam de construções básicas e de estruturas de repetição/iteração, podendo ou não necessitar de estruturas de decisão/condicionais.

Considerando problemas básicos (de acordo com a descrição anterior), quais características observa nos códigos ao realizar uma avaliação?

( ) corretude sintática;

( ) corretude funcional;

( ) quantidade de declarações de variáveis;

( ) quantidade de atribuições de valores para variáveis;

( ) quantidade de instruções utilizadas;

( ) uso correto de operadores (relacionais, matemáticos, outros);

( ) uso correto de funções de entrada e saída;

( ) tamanho do código;

( ) presença de comentários;

( ) similaridade com uma solução esperada.

( ) outros: _____

Considerando problemas de decisão (de acordo com a descrição anterior), quais características observa nos códigos ao realizar uma avaliação?

( ) corretude sintática;

( ) corretude funcional;

( ) quantidade de declarações de variáveis;

( ) quantidade de atribuições de valores para variáveis;

( ) quantidade de instruções utilizadas;

( ) uso correto de operadores (relacionais, matemáticos, outros);

( ) uso correto de funções de entrada e saída;

( ) tamanho do código;

( ) presença de comentários;

( ) similaridade com uma solução esperada.

( ) aplicabilidade da estrutura de decisão utilizada (ex. if vs. switch);

( ) quantidade de decisões utilizadas;

( ) profundidade das decisões utilizadas;

( ) presença/ausência de decisões em sequência;

( ) presença/ausência de decisões compostas;

( ) quantidade de condições especificadas em cada decisão;

( ) aplicabilidade dos operadores lógicos utilizados;

( ) quantidade de operadores lógicos utilizados.

outros: _____

Considerando problemas de repetição (de acordo com a descrição anterior), quais características observa nos códigos ao realizar uma avaliação?

( ) corretude sintática;

( ) corretude funcional;

( ) quantidade de declarações de variáveis;

( ) quantidade de atribuições de valores para variáveis;

( ) quantidade de instruções utilizadas;

( ) uso correto de operadores (relacionais, matemáticos, outros);

( ) uso correto de funções de entrada e saída;

( ) tamanho do código;

( ) presença de comentários;

( ) similaridade com uma solução esperada.

( ) aplicabilidade da estrutura de decisão utilizada (ex. if vs. switch);

( ) quantidade de decisões utilizadas;

( ) profundidade das decisões utilizadas;

( ) presença/ausência de decisões em sequência;

( ) presença/ausência de decisões compostas;

( ) quantidade de condições especificadas em cada decisão;

( ) aplicabilidade dos operadores lógicos utilizados;

( ) quantidade de operadores lógicos utilizados.

( ) aplicabilidade da estrutura de repetição utilizada (ex. for vs. while);

( ) quantidade de estrutura de repetição utilizadas;

( ) profundidade das repetições;

( ) presença/ausência de repetições aninhadas;

( ) eficiência em relação ao tempo de processamento;

( ) eficiência em relação ao uso de memória.

outros: _____

Descreva o processo que utiliza para realizar a avaliação de códigos? _____

_____

Continuidade da pesquisa

O pesquisador poderá entrar em contato, através de e-mail, para dar continuidade nas etapas posteriores da pesquisa?

( ) Sim

( ) Não

Email: _____

A.2 Dados obtidos atráves questionário

Tabela A.1: Características observadas nos códigos ao avaliar problemas ‘Básicos’

Considerando problemas básicos, quais características observa nos códigos ao realizar uma avaliação?

	Frequências	
Critério	Professores	Monitores
a corretude sintática	11	13
a corretude funcional	14	20
a quantidade de declarações de variáveis	2	6
a quantidade de atribuições de valores para variáveis	3	4
a quantidade total de instruções utilizadas	6	7
o uso correto de operadores (relacionais, matemáticos, outros)	11	18
o uso correto de funções de entrada e saída	7	11
a similaridade entre o código apresentado e uma solução esperada	6	5
a descrição da implementação (presença de comentários)	1	3
o tamanho do algoritmo	1	14

Tabela A.2: Características observadas nos códigos ao avaliar problemas 'Condicionais'

Considerando problemas de decisão, quais características observa nos códigos ao realizar uma avaliação?

	Frequências	
Critério	Professores	Monitores
a corretude sintática	8	13
a corretude funcional	13	20
a quantidade de declarações de variáveis	1	6
a quantidade de atribuições de valores para variáveis	1	4
a quantidade total de instruções utilizadas	5	7
o uso correto de operadores (relacionais, matemáticos, outros)	8	18
o uso correto de funções de entrada e saída	3	11
a similaridade entre o código apresentado e uma solução esperada	5	5
a descrição da implementação (presença de comentários)	2	3
o tamanho do algoritmo	1	14
a aplicabilidade da estrutura de decisão utilizada (ex. if vs. switch)	10	16
a profundidade das decisões utilizadas	4	10
a aplicabilidade dos operadores lógicos utilizados	11	17
a presença/ausência de decisões aninhadas	10	12
a presença/ausência de decisões compostas	8	12
a presença/ausência de decisões em sequência	8	10
a quantidade de condições especificadas em cada decisão	4	10
a quantidade de operadores lógicos utilizados	4	7
a quantidade de decisões utilizadas	7	15

Tabela A.3: Características observadas nos códigos ao avaliar problemas ‘Repetição’

Considerando problemas de repetição, quais características observa nos códigos ao realizar uma avaliação?

	Frequências	
Critério	Professores	Monitores
a corretude sintática	7	14
a corretude funcional	12	20
a quantidade de declarações de variáveis	4	6
a quantidade de atribuições de valores para variáveis	5	2
a quantidade total de instruções utilizadas	4	8
o uso correto de operadores (relacionais, matemáticos, outros)	8	14
o uso correto de funções de entrada e saída	4	9
a similaridade entre o código apresentado e uma solução esperada	4	9
a descrição da implementação (presença de comentários)	2	2
o tamanho do algoritmo	3	7
a aplicabilidade da estrutura de decisão utilizada (ex. if vs. switch)	3	12
a profundidade das decisões utilizadas	3	6
a aplicabilidade dos operadores lógicos utilizados	6	11
a presença/ausência de decisões aninhadas	1	10
a presença/ausência de decisões compostas	1	9
a presença/ausência de decisões em sequência	1	9
a quantidade de condições especificadas em cada decisão	2	6
a quantidade de operadores lógicos utilizados	5	7
a quantidade de decisões utilizadas	2	18
a aplicabilidade da estrutura de repetição utilizada (ex. for vs. while)	12	19
a quantidade de estrutura de repetição utilizadas	9	16
a profundidade das iterações	6	13
a presença/ausência de iterações aninhadas	9	14
a eficiência em relação ao tempo de processamento	2	13
a eficiência em relação ao uso de memória	5	5

Apêndice B

Descrição dos problemas adotados no quasi experimento

No presente capítulo são exibidos os enunciados e as descrições de entrada e saída esperadas para cada um dos problemas analisados na execução do quasi experimento exibido na Seção 4.4. Todos os problemas e as submissões foram realizadas através do sistema *The Huxley* [66] [49], descrito anteriormente.

Tabela B.1: Descrição do problema 'Distância entre dois pontos'.

Problema: Distância entre dois pontos
Enunciado: Leia os quatro valores correspondentes aos eixos x e y de dois pontos quaisquer no plano, p1(x1,y1) e p2(x2,y2) e calcule a distância entre eles, mostrando 4 casas decimais após a vírgula.
Exemplo de entrada: O arquivo de entrada contém duas linhas de dados. Na primeira linha contém os valores inteiros: x1, y1 e na segunda linha contém os valores inteiros x2, y2.
Exemplo de saída: Calcule e imprima o valor da distância segundo a fórmula fornecida, com 4 casas após o ponto decimal.

Tabela B.2: Descrição do problema ‘Salário com Bônus’.

Problema: Salário com Bônus
Enunciado: Faça um programa que leia o nome de um vendedor, o seu salário fixo e o total de vendas efetuadas por ele no mês (em dinheiro). Sabendo que este vendedor ganha 15% de comissão sobre suas vendas efetuadas, informar o total a receber no final do mês, com duas casas decimais.
Exemplo de entrada: O arquivo de entrada contém um texto (primeiro nome do vendedor) e 2 números com duas casas decimais, representando o salário fixo do vendedor e montante total das vendas efetuadas por este vendedor, respectivamente.
Exemplo de saída: Imprima o total que o funcionário deverá receber, conforme exemplo fornecido. Você deve sempre arredondar. Por exemplo, se o resultado for 57.347, você deve imprimir 57.35 e não 57.34

Tabela B.3: Descrição do problema ‘Aprovado’.

Problema: Aprovado
Enunciado: Faça um programa que leia três notas (valores reais) de um aluno, calcule sua média aritmética e imprima uma mensagem dizendo se o aluno foi aprovado, reprovado ou deverá fazer prova final. O critério de aprovação é o seguinte: • Aprovado (média ≥ 7); • Reprovado (média < 3); • Prova final ($3 \leq$ média < 7).
Exemplo de entrada: 03 números reais separados por um final de linha.
Exemplo de saída: Uma mensagem que pode ser: • aprovado • reprovado • prova final

Tabela B.4: Descrição do problema ‘Eleitor’.

Problema: Eleitor
Enunciado: Faça um programa que leia a idade (valor inteiro) de uma pessoa e informe sua classe eleitoral: • nao eleitor (abaixo de 16 anos) • eleitor obrigatorio (maior e igual a 18 ou menor e igual a 65 anos) • eleitor facultativo (entre 16 e 18 anos ou acima dos 65 anos)
Exemplo de entrada: Consiste de um número inteiro indicando a idade da pessoa.
Exemplo de saída: Uma linha escrito: “nao eleitor”, “eleitor obrigatorio” ou “eleitor facultativo” (sem as aspas) de acordo com o critério da descrição.

Tabela B.5: Descrição do problema ‘Loop de ímpares’.

Problema: Loop de ímpares
Enunciado: Faça um programa que imprima todos os números ímpares entre dois números dados.
Exemplo de entrada: Dois números inteiros, n e m, separados por um final de linha.
Exemplo de saída: Todos os números ímpares maiores ou iguais a n e menores ou iguais a m, separados por um final de linha.

Tabela B.6: Descrição do problema ‘Divisível por 3’.

Problema: Divisível por 3
Enunciado: O problema envolve achar a quantidade de divisores de um número que são divisíveis por 3.
Exemplo de entrada: Um inteiro N.
Exemplo de saída: Um inteiro R seguido de um final de linha, sendo R o número de divisores de N que são divisíveis por 3. Caso não tenha nenhum imprima “O numero nao possui divisores multiplos de 3!” sem as aspas e com um final de linha.

Apêndice C

Resultados obtidos para os problemas adotados no quasi experimento

No presente capítulo são exibidos separadamente os resultados obtidos para cada problema analisados na execução do quasi experimento exibido na Seção 4.4.

Na Tabela C.1 são exibidos os resultados para o problema 'Salário com Bônus'. Visto que nem todos os avaliadores forneceram avaliações completas para este problema, foram consideradas as avaliações de 6 avaliadores.

Tabela C.1: Resultados obtidos para o problema 'Salário com Bônus' (Tipo básico)

Kappa de Cohen			**Correl. de Pearson**			**Dist. de notas**		
Máx.	Min.	Méd.	Máx.	Min.	Méd.	Máx.	Min.	Méd.
1.00	0.74	0.82	1.00	0.94	0.97	9.00	0.00	5.16

Na Tabela C.2 são exibidos os resultados para o problema 'Distância entre dois pontos'. Todos os oito avaliadores forneceram avaliações completas para este problema.

Tabela C.2: Resultados obtidos para o problema 'Distância entre dois pontos' (Tipo básico)

Kappa de Cohen			**Correl. de Pearson**			**Dist. de notas**		
Máx.	Min.	Méd.	Máx.	Min.	Méd.	Máx.	Min.	Méd.
1.00	0.74	0.85	1.00	0.93	0.97	10.50	0.00	4.75

Na Tabela C.3 são exibidos os resultados para o problema 'Aprovado'. Visto que nem to-

dos os avaliadores forneceram avaliações completas para este problema, foram consideradas as avaliações de 7 avaliadores.

Kappa de Cohen			**Correl. de Pearson**			**Dist. total**		
Máx.	Min.	Méd.	Máx.	Min.	Méd.	Máx.	Min.	Méd.
0.95	0.64	0.77	0.99	0.92	0.96	24	2.00	12.57

Tabela C.3: Resultados obtidos para o problema ‘Aprovado’ (Tipo condicional)

Na Tabela C.4 são exibidos os resultados para o problema ‘Eleitor’. Visto que nem todos os avaliadores forneceram avaliações completas para este problema, foram consideradas as avaliações de 6 avaliadores.

Kappa de Cohen			**Correl. de Pearson**			**Dist. total**		
Máx.	Min.	Méd.	Máx.	Min.	Méd.	Máx.	Min.	Méd.
0.76	0.39	0.56	0.88	0.70	0.80	54	23	37.83

Tabela C.4: Resultados obtidos para o problema ‘Eleitor’ (Tipo condicional)

Na Tabela C.5 são exibidos os resultados para o problema ‘Divisivel por 3’. Visto que nem todos os avaliadores forneceram avaliações completas para este problema, foram consideradas as avaliações de 6 avaliadores.

Kappa de Cohen			**Correl. de Pearson**			**Dist. total**		
Máx.	Min.	Méd.	Máx.	Min.	Méd.	Máx.	Min.	Méd.
0.77	0.51	0.60	0.93	0.73	0.82	57.50	27.50	41.83

Tabela C.5: Resultados obtidos para o problema ‘Divisivel por 3’ (Tipo repetição)

Na Tabela C.6 são exibidos os resultados para o problema ‘Loop de impares’. Visto que nem todos os avaliadores forneceram avaliações completas para este problema, foram consideradas as avaliações de 6 avaliadores.

Na Tabela C.7 são exibidos os resultados das concordâncias entre cada avaliador especialista e o avaliador artificial, obtidos na execução do quasi experimento.

Kappa de Cohen			Correl. de Pearson			Dist. total		
Máx.	Min.	Méd.	Máx.	Min.	Méd.	Máx.	Min.	Méd.
0.95	0.86	0.90	0.99	0.98	0.98	6.00	1.00	4.50

Tabela C.6: Resultados obtidos para o problema 'Loop de impares' (Tipo repetição)

Tabela C.7: Concordâncias entre cada avaliador e avaliador artificial.

Kappa de Cohen								
	id:1	**id:5**	**id:6**	**id:9**	**id:10**	**id:11**	**id:12**	**id:13**
Salário com Bônus	0.79	- -	0.74	1.00	0.77	0.80	- -	0.85
Distância entre 2 pontos	0.88	0.76	0.74	0.86	1.00	0.94	0.86	0.80
Aprovado	0.67	- -	0.78	0.95	0.89	0.68	0.64	0.78
Eleitor	0.56	- -	0.53	0.76	0.58	0.39	- -	0.57
Loop de ímpares	0.95	- -	0.86	0.87	0.90	0.92	0.89	0.95
Divisível por 3	0.62	- -	0.60	0.55	0.55	0.51	- -	0.77
Média	**0.74**	**0.76**	**0.70**	**0.83**	**0.78**	**0.70**	**0.79**	**0.78**

A seguir são exibidos os resultados obtidos para cada um dos problemas analisados. Os resultados são descritos através das medidas estatísticas e de comparação apresentadas no Capítulo 2.3. Também é apresentada para cada problema a lista de métricas identificadas para cada um dos avaliadores.

Problema: 'Salário com bônus'

Problema: 'Distância entre dois pontos'

Problema: 'Aprovado'

Problema: 'Eleitor'

Problema: 'Divisível por 3'

Problema: 'Loop de ímpares'

Na Tabela E.3 são exibidas as contagens das métricas identificadas em relação aos critérios dos avaliadores para cada um dos problemas. Estas métricas foram identificadas através da seleção automática de métricas

Especialista	Kappa	Pearson	D. Euclidiana	Distância total
1	0.79	0.98	2.00	4.0
6	0.74	0.98	2.17	3.5
9	1.00	1.00	0.00	0.0
10	0.77	0.83	9.16	14.0
11	0.80	0.93	6.04	7.0
13	0.85	0.86	7.29	9.5

Tabela C.8: Resumo dos resultados obtidos para o problema ‘salário com bônus’ utilizando o algoritmo de agrupamento *kmeans*.

Especialista	Kappa	Pearson	D. Euclidiana	Distância total
1	0.88	0.98	2.82	4.0
5	0.76	0.80	8.36	12.0
6	0.74	0.98	2.17	3.5
9	0.86	0.87	11.31	16.0
10	1.00	1.00	0.00	0.0
11	0.94	0.98	4.00	4.0
12	0.86	0.99	1.11	1.5
13	0.80	0.98	3.67	6.0

Tabela C.9: Resumo dos resultados obtidos para o problema ‘Distância entre dois pontos’ utilizando o algoritmo de agrupamento *kmeans*.

Especialista	Kappa	Pearson	D. Euclidiana	Distância total
1	0.67	0.95	7.61	24.0
6	0.78	0.96	5.61	12.0
9	0.95	0.99	2.00	2.0
10	0.89	0.99	2.44	4.0
11	0.68	0.92	8.48	24.0
12	0.64	0.98	3.53	10.0
13	0.78	0.93	8.24	18.0

Tabela C.10: Resumo dos resultados obtidos para o problema 'Aprovado' utilizando o algoritmo de agrupamento *kmeans*.

Especialista	Kappa	Pearson	D. Euclidiana	Distância total
1	0.56	0.27	19.00	63.0
6	0.53	0.59	14.96	49.0
9	0.76	0.83	11.48	26.0
10	0.58	0.79	10.24	33.0
11	0.39	0.59	17.23	65.0
13	0.56	0.84	13.30	39.0

Tabela C.11: Resumo dos resultados obtidos para o problema 'Eleitor' utilizando o algoritmo de agrupamento *kmeans*.

Especialista	Kappa	Pearson	D. Euclidiana	Distância total
1	0.62	0.77	14.53	45.5
6	0.60	0.75	12.54	39.0
9	0.55	0.72	17.32	58.5
10	0.55	0.60	17.03	57.5
11	0.51	0.72	17.79	51.0
13	0.76	0.62	22.89	56.0

Tabela C.12: Resumo dos resultados obtidos para o problema 'Divisível por 3' utilizando o algoritmo de agrupamento *kmeans*.

Especialista	Kappa	Pearson	D. Euclidiana	Distância total
1	0.95	0.99	3.00	3.0
6	0.86	0.96	5.40	7.5
9	0.87	0.98	3.46	6.0
10	0.90	0.91	10.19	12.0
11	0.92	0.99	1.00	1.0
12	0.89	0.99	1.41	2.0
13	0.95	0.99	1.00	1.0

Tabela C.13: Resumo dos resultados obtidos para o problema 'Loop de ímpares' utilizando o algoritmo de agrupamento *kmeans*.

Tabela C.14: Contagem das métricas identificadas em relação aos critérios dos avaliadores para cada um dos problemas.

Métricas / **Problemas**	Salário com Bônus	Distância entre 2 pontos	Aprovado	Eleitor	Loop de ímpares	Divisível por 3	**Total**
Corretude funcional	7	4	6	2	6	6	31
Complexidade ciclomática			5	2	4	2	13
Número de operadores distintos ($n1$)	1	3	3	1	5	3	16
Número de operandos distintos ($n2$)	4	5	3	3	3	2	20
Sim. baseada no coef. de Jaccard	4	1	4	2		3	14
Sim. baseada na dist. de edição de texto	6	4	4	5	5	3	27
Sim. baseada na dist. de edição de árvore	3	3	5	3	5	3	21

Apêndice D

Descrição dos problemas adotados no experimento

No presente capítulo são exibidos os enunciados e as descrições de entrada e saída esperadas para cada um dos problemas analisados na execução do experimento exibido na Seção 4.5. Todos os problemas e as submissões foram realizadas através do sistema *The Huxley* [66] [49], descrito anteriormente.

Tabela D.1: Descrição do problema ‘Consumo’.

Problema: Consumo
Enunciado: Calcule o consumo médio de um automóvel sendo fornecidos a distância total percorrida (em Km) e o total de combustível gasto (em litros).
Exemplo de entrada: A entrada contem dois valores: um valor inteiro X representando a distancia total percorrida (em Km) e um valor real Y representando o total de combustível gasto.
Exemplo de saída: Apresente o valor que representa o consumo médio do automóvel com 3 casas apos a virgula, seguido da mensagem "km/l".

Tabela D.2: Descrição do problema ‘Fahrenheit para Celsius’.

Problema: Fahrenheit para Celsius
Enunciado: Converter uma temperatura dada em graus Fahrenheit para graus Celsius.
Exemplo de entrada: A temperatura em Fahrenheit e dada como um numero real.
Exemplo de saída: Um numero real, formatado com duas casas decimais, seguido de um final de linha.

Tabela D.3: Descrição do problema ‘Classificação de Triângulos’.

Problema: Classificação de Triângulos
Enunciado: Os triângulos mais simples são classificados de acordo com os limites das proporções relativas de seus lados: Um triangulo equilátero possui todos os lados congruentes ou seja iguais. Um triangulo equilátero e também equiângulo: todos os seus ângulos internos são congruentes (medem 60), sendo, portanto, classificado como um polígono regular. Um triangulo isósceles possui pelo menos dois lados de mesma medida e dois ângulos congruentes. O triangulo equilátero e, consequentemente, um caso especial de um triangulo isósceles, que apresenta não somente dois, mas todos os três lados iguais, assim como os ângulos, que medem todos 60o. Num triangulo isósceles, o angulo formado pelos lados congruentes e chamado angulo do vértice. Os demais ângulos denominam-se ângulos da base e são congruentes. Em um triângulo escaleno, as medidas dos três lados são diferentes. Os ângulos internos de um triângulo escaleno também possuem medidas diferentes. Sua missão e escrever um programa para classificar um triangulo de lados de comprimentos dados em: escaleno (os três lados de comprimentos diferentes), isósceles (dois lados de comprimentos iguais) ou equilátero (os três lados de comprimentos iguais).
Exemplo de entrada: A entrada consiste de 03 números reais maiores que zero correspondendo ao comprimento dos lados do triangulo.
Exemplo de saída: A saída deve ser: “escaleno”, “isosceles” ou “equilatero”’ seguido de um final de linha. Obs.: as aspas acima não devem ser impressas e a saída deve ser impressa sem acentos.

Tabela D.4: Descrição do problema ‘3 Números em Ordem Crescente’.

Problema: 3 Números em Ordem Crescente
Enunciado: Faca um programa que leia 3 números inteiros e os imprima em ordem crescente.
Exemplo de entrada: 03 números inteiros separados por um final de linha.
Exemplo de saída: Os 03 números lidos impressos em ordem crescente, cada um em uma linha.

Tabela D.5: Descrição do problema ‘Maior Múltiplo’.

Problema: Maior Múltiplo
Enunciado: Seu objetivo é determinar o maior múltiplo de um inteiro dado menor do que ou igual a um outro inteiro dado
Exemplo de entrada: Consiste de dois inteiros positivos M e N.
Exemplo de saída: A saída consiste do maior número que seja múltiplo de M e menor que N, se não houver um múltiplo de M menor que N você deve imprimir “sem múltiplos menores que N”, sem as aspas, onde N deve ser substituído pelo valor de entrada N.

Tabela D.6: Descrição do problema ‘Fatorial’.

Problema: Fatorial
Enunciado: Calcule os fatoriais de uma sequência de números dada.
Exemplo de entrada: O programa receberá uma sequência de inteiros n, onde $0 <= n <= 12$. O programa encerra a sua execução quando o número n dado for -1.
Exemplo de saída Para cada n, deve-se imprimir um inteiro k seguido de um final de linha, correspondendo ao fatorial.

Apêndice E

Resultados obtidos para os problemas adotados no experimento

No presente capítulo são exibidos separadamente os resultados obtidos para cada problema analisados na execução do experimento exibido na Seção 4.5.

Na Tabela E.1 são exibidos os resultados das concordâncias entre cada avaliador especialista e o avaliador artificial, utilizando execução baseada na relação critério e métrica.

Tabela E.1: Concordâncias entre cada avaliador especialista e o avaliador artificial, utilizando execução baseada na relação critério e métrica.

Kappa de Cohen									
	Professores			**Monitores**					
	id:1	**id:5**	**id:19**	**id:3**	**id:6**	**id:17**	**id:25**	**id:26**	**id:27**
Fahrenheit para Celsius	0.61	0.93	0.62	1.00	0.47	0.55	0.74	0.85	0.55
Consumo	0.90	1.00	0.91	0.86	0.70	0.67	0.51	0.40	0.63
Class. de Triângulos	0.59	1.00	1.00	1.00	0.70	0.82	0.63	0.74	0.54
3 Núm. em Ord. Cres.	0.55	1.00	0.90	0.62	0.50	0.49	0.34	0.55	0.73
Maior Múltiplo	0.73	0.68	1.00	0.55	0.76	0.86	0.61	0.62	0.59
Fatorial	0.54	0.67	0.82	0.87	0.56	0.62	0.46	0.63	0.64
Média geral	**0.65**	**0.88**	**0.87**	**0.81**	**0.62**	**0.67**	**0.55**	**0.63**	**0.61**

Na Tabela E.2 são exibidos os resultados das concordâncias entre cada avaliador especialista e o avaliador artificial, utilizando execução baseada na seleção automática de métricas.

Tabela E.2: Concordâncias entre cada avaliador especialista e o avaliador artificial, utilizando execução baseada na seleção automática de métricas.

Kappa de Cohen									
	Professores			**Monitores**					
	id:1	**id:5**	**id:19**	**id:3**	**id:6**	**id:17**	**id:25**	**id:26**	**id:27**
Fahrenheit para Celsius	0.80	0.93	1.00	1.00	0.54	0.67	0.81	1.00	0.81
Consumo	0.90	1.00	0.91	0.87	0.77	0.90	0.71	1.00	0.63
Class. de Triângulos	0.85	1.00	1.00	1.00	0.81	0.88	0.78	0.93	0.64
3 Núm. em Ord. Cres.	0.73	1.00	0.90	0.77	0.62	0.57	0.58	1.00	0.73
Maior Múltiplo	0.86	0.79	1.00	0.66	0.84	0.86	0.74	1.00	0.76
Fatorial	0.68	0.74	0.91	0.93	0.75	0.84	0.60	1.00	0.70
Média geral	**0.80**	**0.93**	**0.95**	**0.87**	**0.72**	**0.79**	**0.70**	**0.99**	**0.71**

Na Tabela E.3 são exibidas as contagens das métricas identificadas em relação aos critérios dos avaliadores para cada um dos problemas. Estas métricas foram identificadas através da seleção automática de métricas

Tabela E.3: Contagem das métricas identificadas em relação aos critérios dos avaliadores para cada um dos problemas.

Métricas / **Problemas**	Fahrenheit para Celsius	Consumo	3 Núm. em Ord. Cres.	Class. de Triângulos	Maior Múltiplo	Fatorial	**Total**
Corretude funcional	6	8	6	5	5	5	35
Complexidade ciclomática	1	1	5	3	4	5	19
Número de operadores distintos ($n1$)	5	2	6	7	4	6	30
Número de operandos distintos ($n2$)	1	2	5	5	3	3	19
Número total de operadores ($N1$)	3		2	3	3	6	17
Número total de operandos ($N2$)	4	2	3	4	4	2	19
Vocabulário ($n = n1 + n2$)	4	1	4	2	4	5	20
Tamanho ($N = N1 + N2$)	2	1	1	1	1	1	7
Volume ($V = N \times \log_2 n$)							0
LLOC							0
SLOC	6	7	4	4	5	7	33
Linhas em branco							0
Sim. baseada no coef. de Jaccard	1		2	3	3	2	11
Sim. baseada na dist. de edição de texto							0
Sim. baseada na dist. de edição de árvore							0

Na Figura E.1, E.2 e E.3 são exibidas as frequências de notas fornecidas pelos avaliadores professores em cada um dos problemas apresentados.

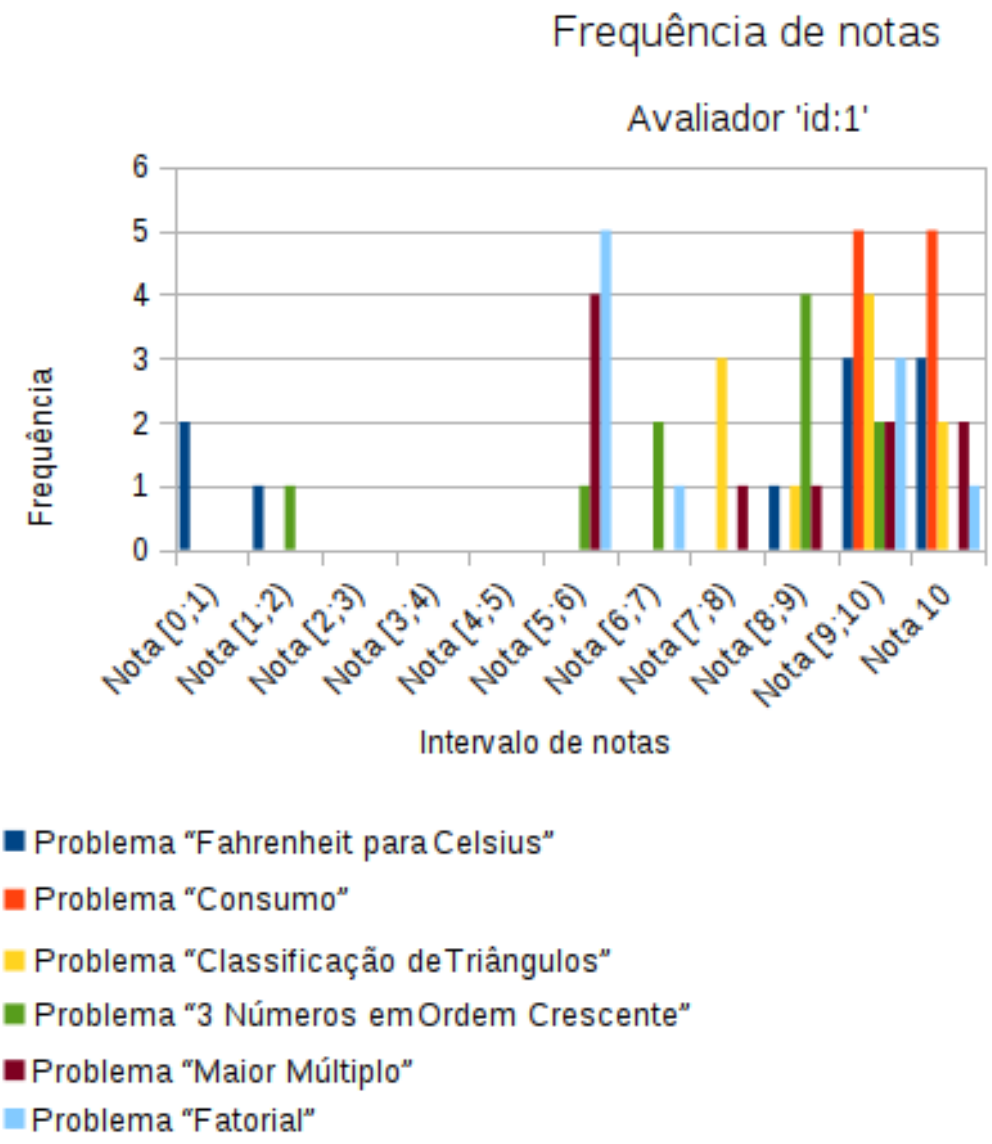


Figura E.1: Frequências de notas fornecidas pelo avaliador professor 'id:1' em cada um dos problemas apresentados.

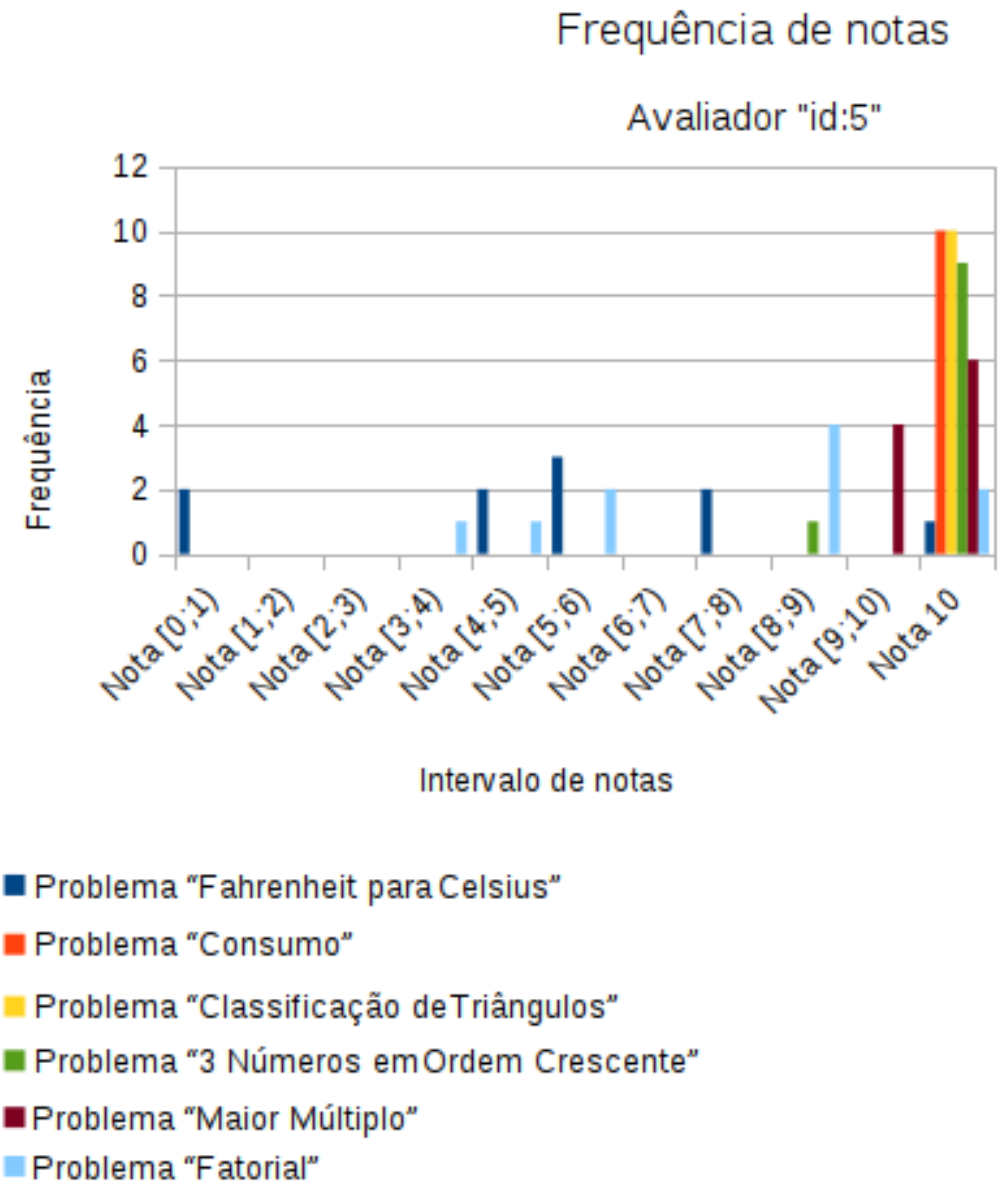


Figura E.2: Frequências de notas fornecidas pelo avaliador professor ‘id:5’ em cada um dos problemas apresentados.

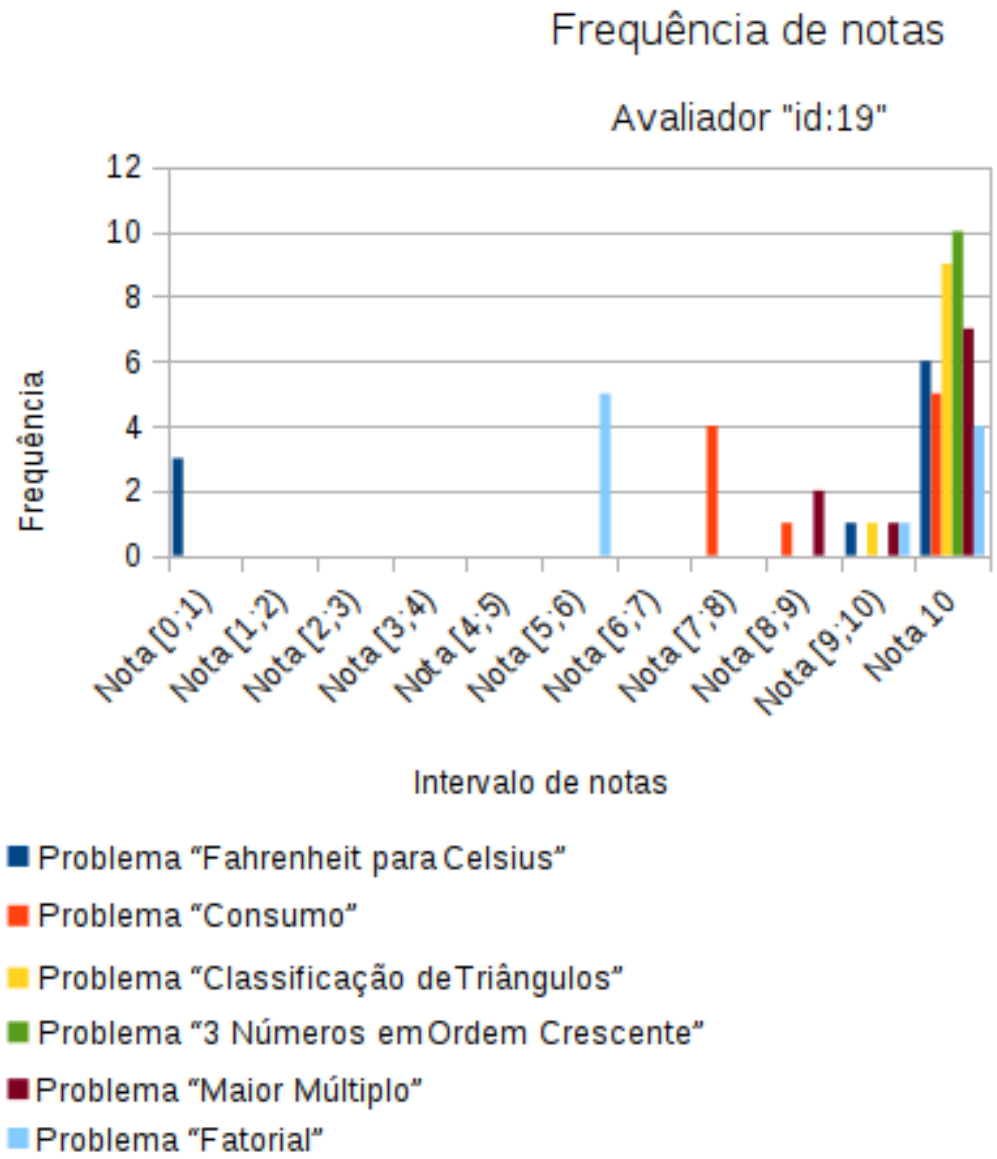


Figura E.3: Frequências de notas fornecidas pelo avaliador professor ‘id:19’ em cada um dos problemas apresentados.

Na Figura E.4, E.5, E.6, E.7, E.8, E.9, são exibidas as frequências de notas fornecidas pelos avaliadores monitores em cada um dos problemas apresentados.

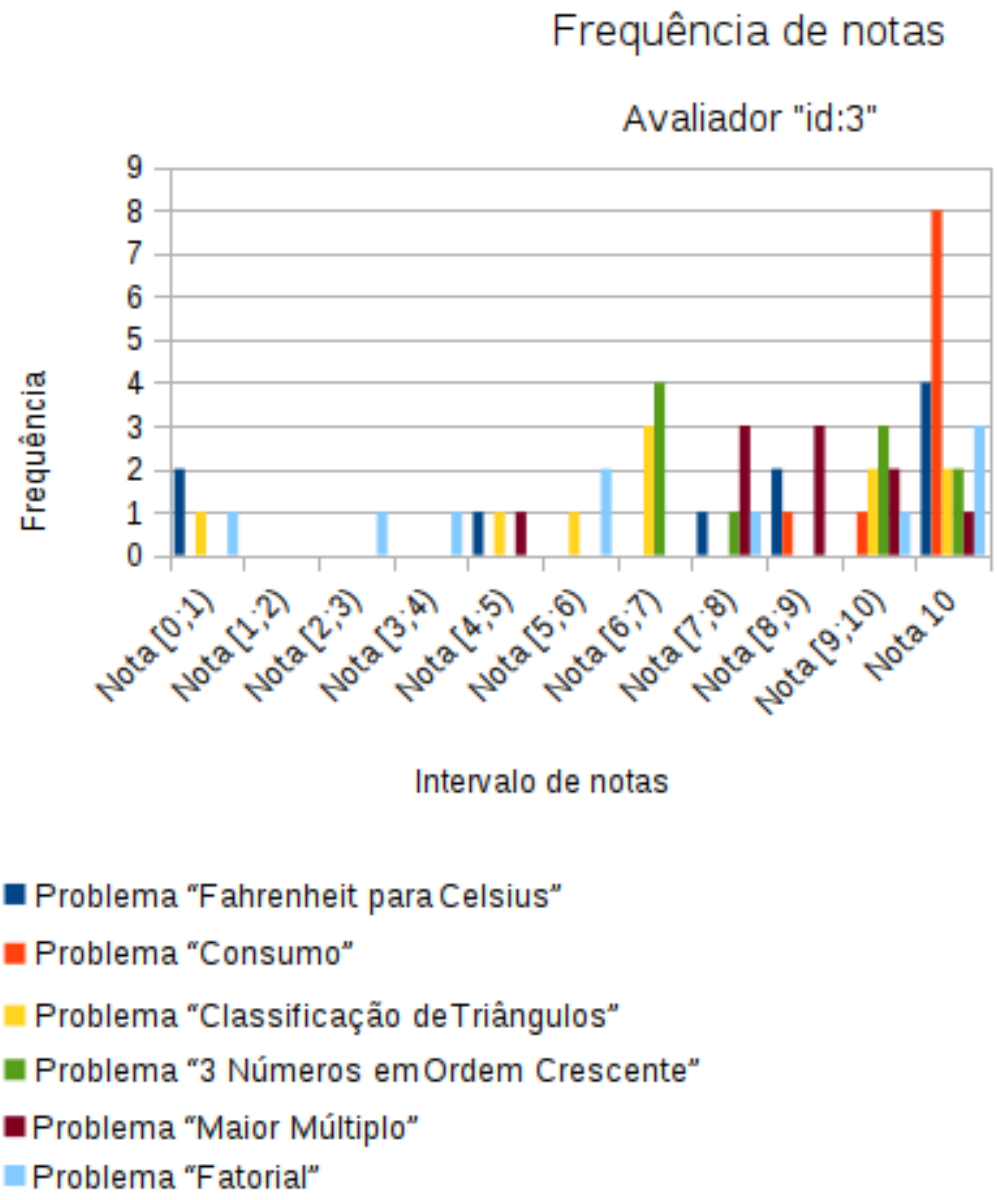


Figura E.4: Frequências de notas fornecidas pelo avaliador monitor ‘id:3’ em cada um dos problemas apresentados.

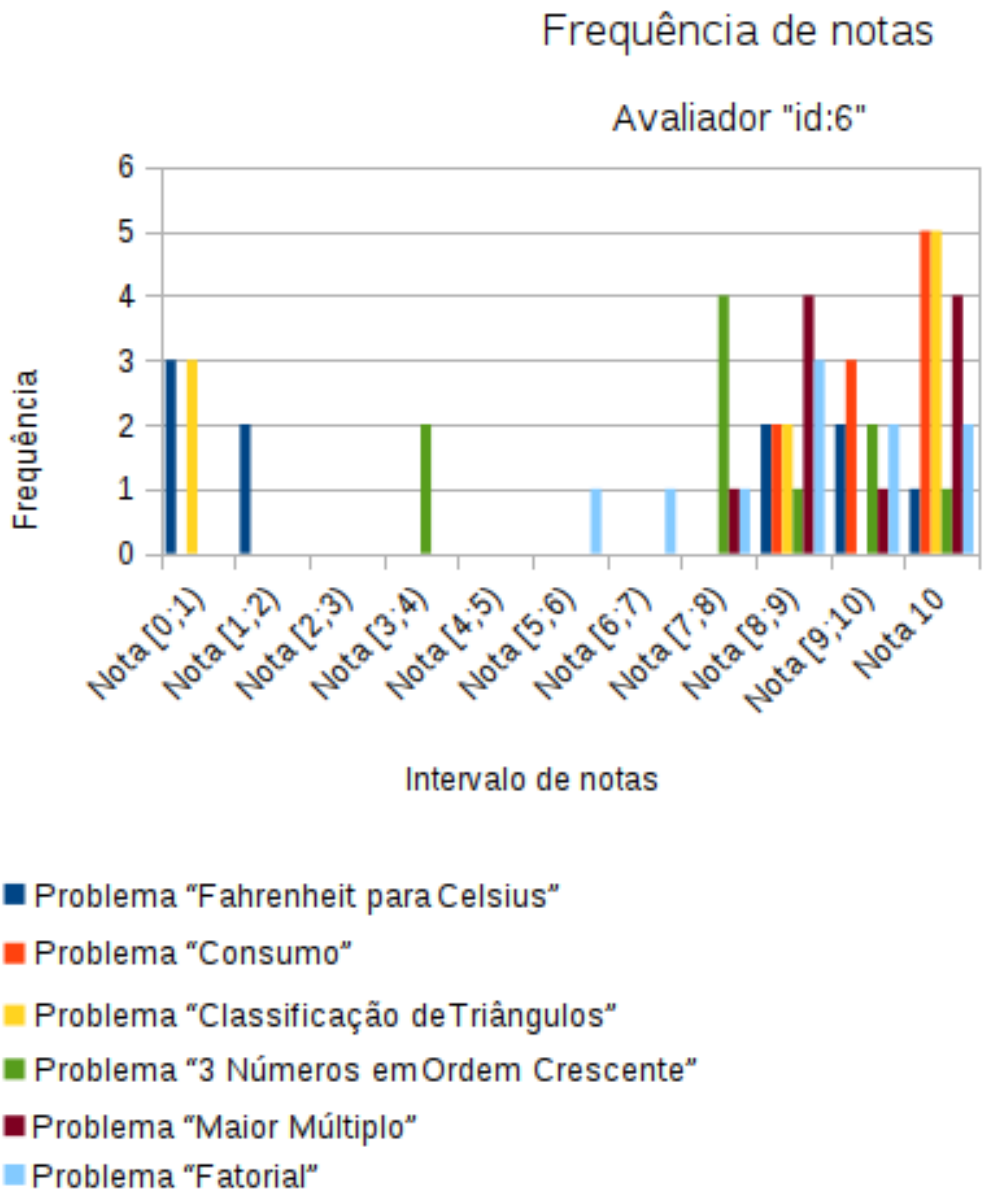


Figura E.5: Frequências de notas fornecidas pelo avaliador monitor ‘id:6’ em cada um dos problemas apresentados.

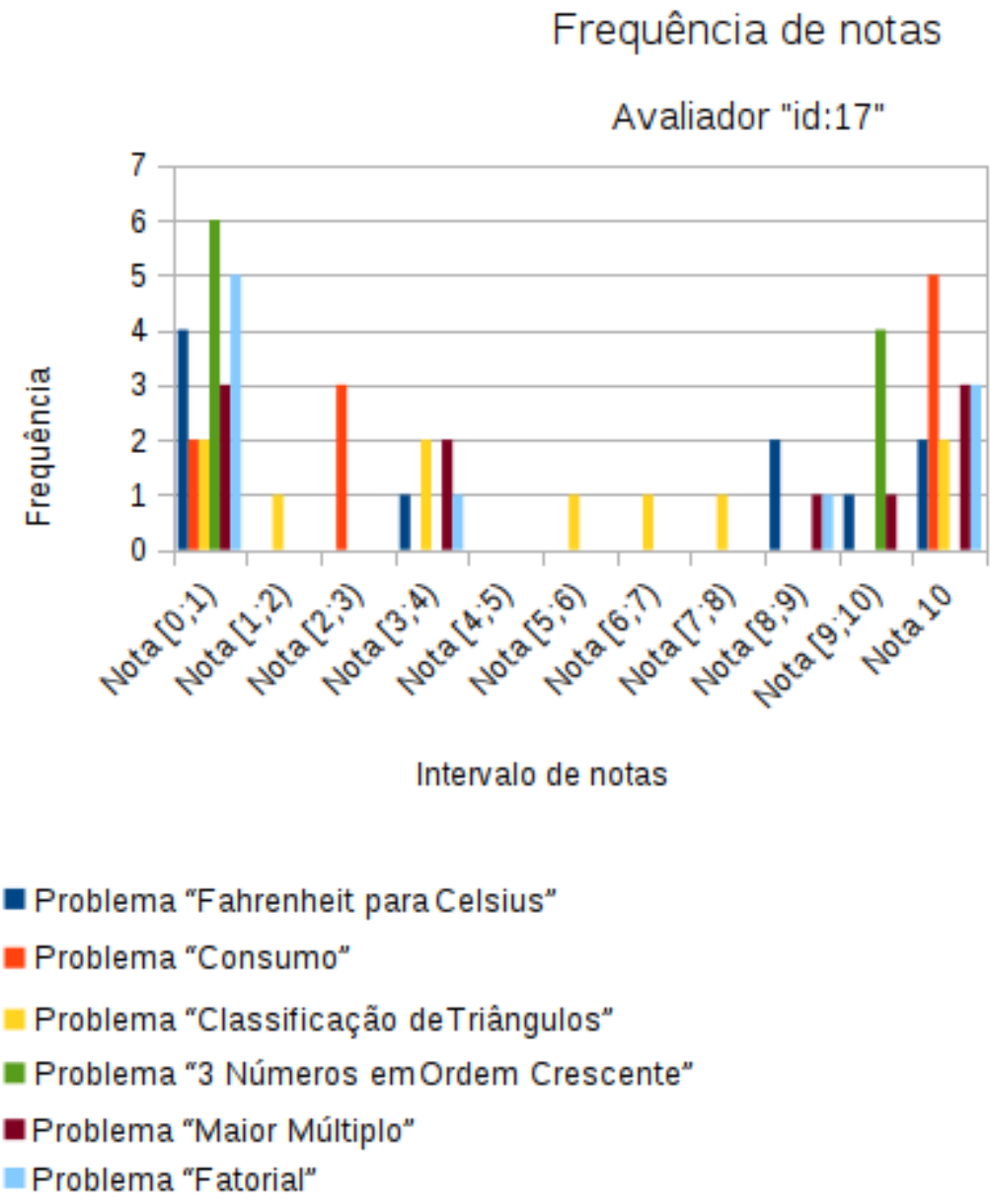


Figura E.6: Frequências de notas fornecidas pelo avaliador monitor ‘id:17’ em cada um dos problemas apresentados.

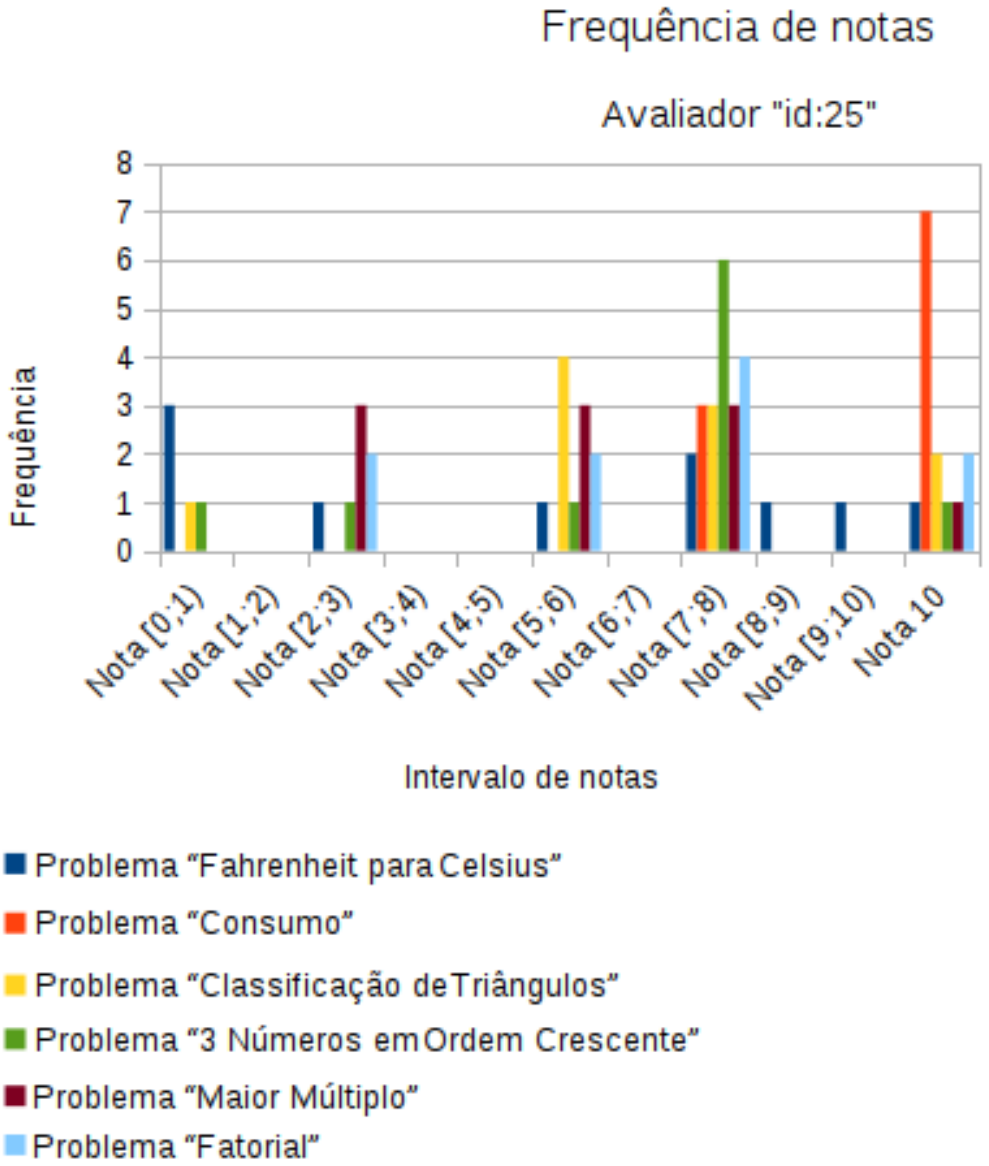


Figura E.7: Frequências de notas fornecidas pelo avaliador monitor ‘id:25’ em cada um dos problemas apresentados.

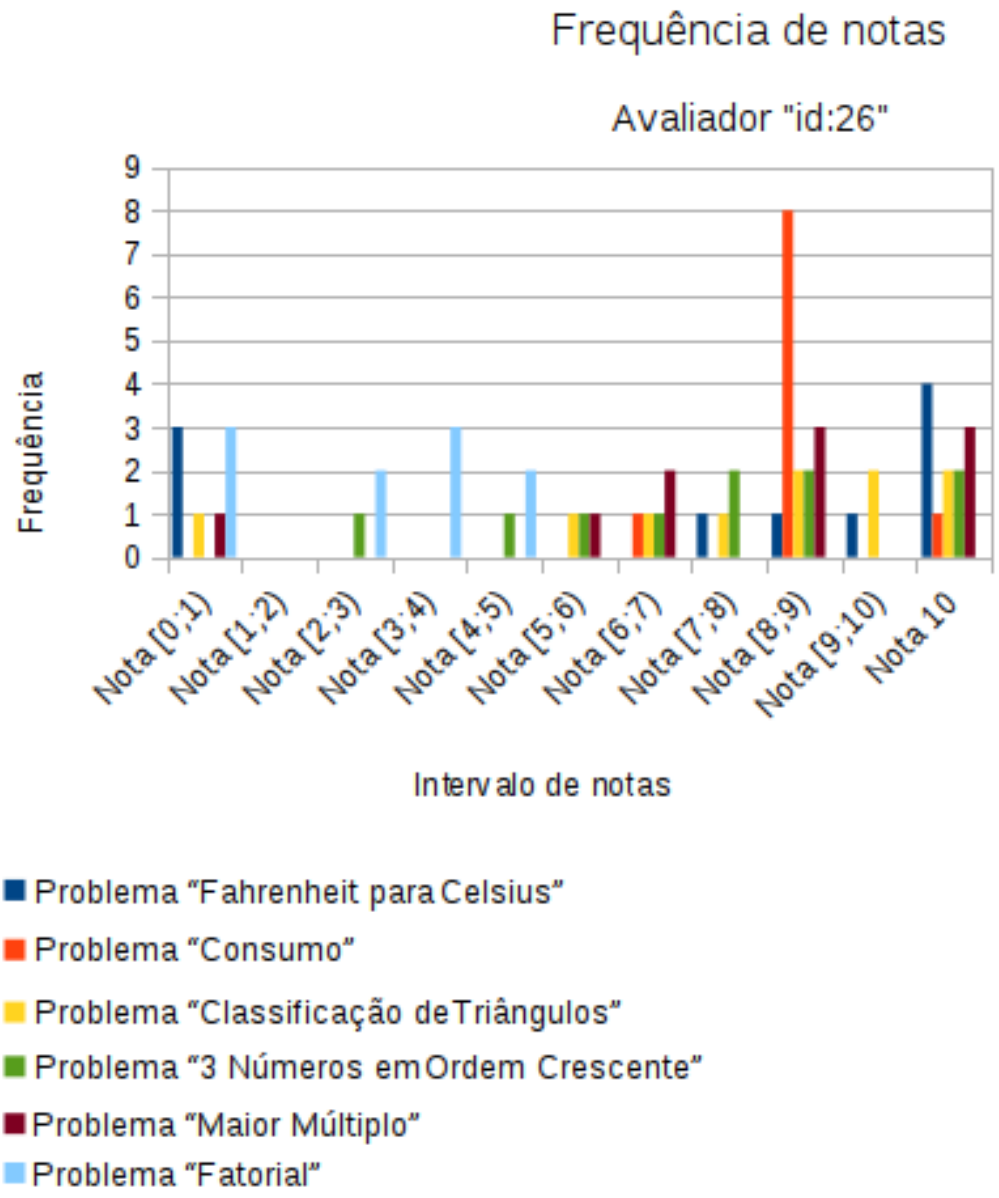


Figura E.8: Frequências de notas fornecidas pelo avaliador monitor ‘id:26’ em cada um dos problemas apresentados.

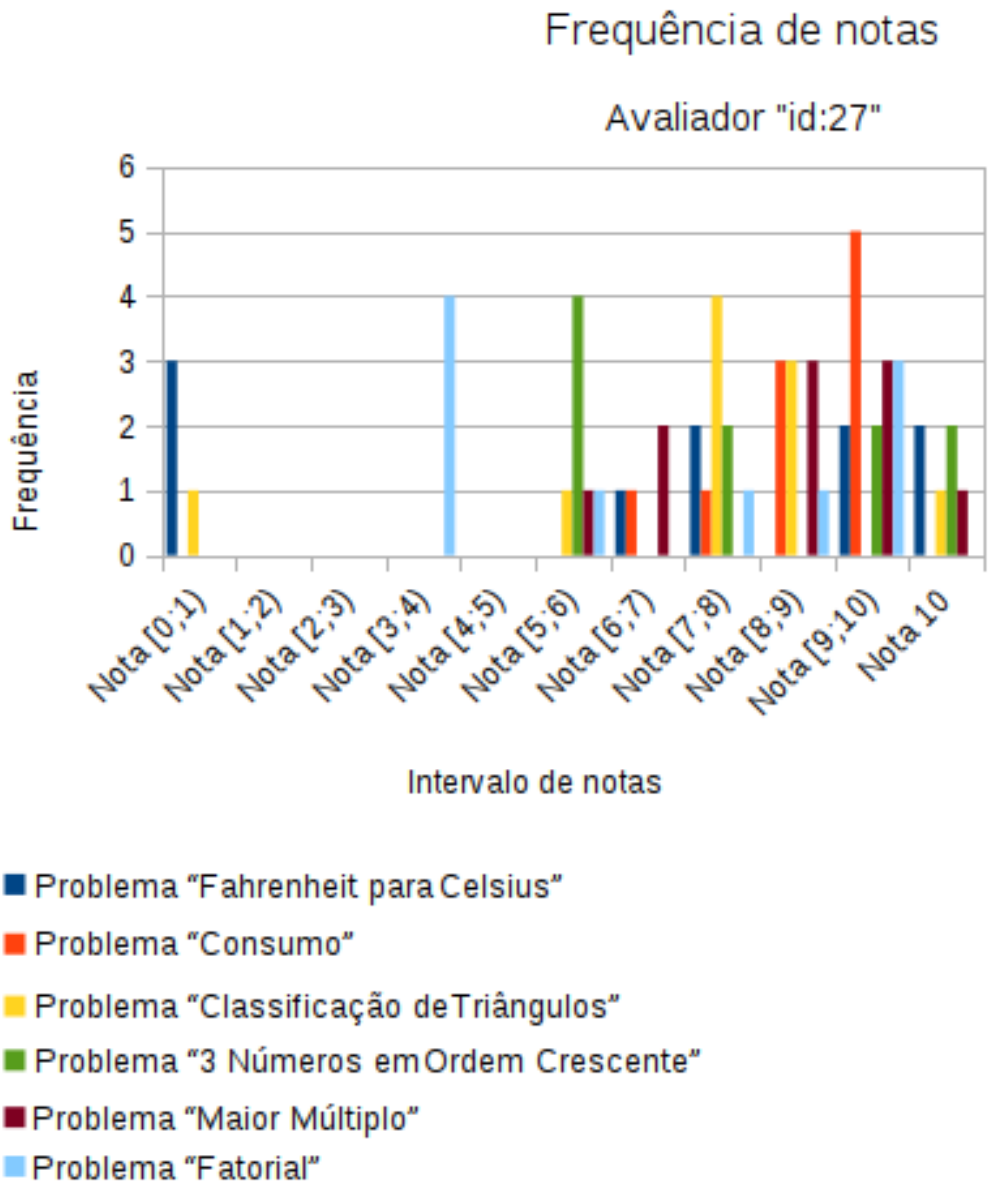


Figura E.9: Frequências de notas fornecidas pelo avaliador monitor ‘id:27’ em cada um dos problemas apresentados.