Recomendação Automática de Problemas em Juízes Online Usando Processamento de Linguagem Natural e Análise Dirigida aos Dados

Hermino Barbosa de Freitas Júnior [1], **Filipe Dwan Pereira**[1], **Elaine H. T. Oliveira**[2],**David B. F. Oliveira**[2], **Leandro S. G. Carvalho**[2]

[1] Departamento de Ciência da Computação - Universidade Federal de Roraima (UFRR)
[2]Instituto de Computação - Universidade Federal do Amazonas (UFAM)
{filipe.dwan,hermino}@ufrr.br,{elaine,david,leandro}@icomp.ufam.edu.br

Abstract. *Typically, learners struggle to find suitable problems in online judges due to the huge volume of problems available on these systems. In this sense, we propose and validate methods for automatic recommendation of problems in online judges, where the recommendations are made based on a target problem, previously solved by the learner. In total, 15 students and 3 instructors evaluated 324 problems recommended by our methods, using a double blind control approach. As a result, we showed that our methods presented better recommendations for the students in terms of effort employed and success achieved (higher success rate and lower failure and dropout rate). In closing, we believe that our methods can be used to support instructors of selecting problems to create assignment lists.*

Resumo. *Estudantes regularmente enfrentam dificuldades para encontrar problemas adequados em juízes online devido ao grande volume de exercícios disponíveis nesses sistemas. Nesse sentido, este estudo propõe e avalia métodos para recomendação automática de problemas em juízes online, em que as recomendações são realizadas a partir de um problema alvo, resolvido anteriormente pelo aluno. Um total de 324 recomendações foram avaliadas por 15 estudantes e 3 professores de programação, utilizando uma abordagem duplamente cega. Como resultado, mostrou-se que as recomendações dadas pelos métodos propostos foram mais adequadas do que as do baseline em termos de esforço empregado e sucesso obtido (maior taxa de acerto e menor taxa de erro e desistência). Por fim, acredita-se que os métodos de recomendação podem ser usados para auxiliar professores a criarem listas de exercícios.*

1. Introdução

Juiz *Online* (JO) é um tipo de sistema capaz de avaliar automaticamente a corretude de códigos desenvolvidos para solucionar determinados exercícios ou problemas de programação. Tipicamente, os JOs disponibilizam uma *Integrated Development Environment* (IDE) onde os códigos podem ser desenvolvidos e avaliados em tempo real [Dwan et al. 2017]. Atualmente, existe uma crescente aceitação desses sistemas por parte de educadores e universidades [Yera e Martínez 2017, Pereira et al. 2020]. Além disso, muitos estudantes usam JOs para treinamento e aperfeiçoamento de suas habilidades e preparação para olimpíadas de programação [Zhao et al. 2018].

Com a crescente popularidade dos JOs, novos exercícios de diferentes categorias e níveis de dificuldade são cadastrados constantemente [Zhao et al. 2018, Fonseca et al.

DOI: 10.5753/cbie.sbie.2020.1152

2020]. Consequentemente, esse aumento contínuo dos exercícios disponíveis pode trazer uma sobrecarga de informação aos estudantes que acabam encontrando dificuldade de escolher problemas adequados ao seu conhecimento em programação [Yera e Martínez 2017]. Com efeito, isso pode provocar sentimentos de frustração e até a desistência. Logo, é importante adotar medidas que possibilitem lidar com esse cenário de forma eficiente e escalável. Nesse sentido, sistemas de recomendação podem ser bastante úteis, pois eles auxiliam usuários a identificarem conteúdos de interesse em um conjunto de opções que poderiam caracterizar uma sobrecarga. Além disso, esses sistemas podem ser úteis para os professores selecionarem problemas adequados para compor suas listas de exercícios.

Como uma proposta de solução para o problema apresentado, este estudo propõe, comparar e validar métodos de recomendação de problemas para JOs, onde as recomendações são realizadas a partir de um *Problema Alvo* (PA), resolvido anteriormente pelo estudante. Assumimos a hipótese de que, se o estudante resolveu o PA e o problema recomendado é *similar* ao PA, logo o problema recomendado é provavelmente compatível com o nível de conhecimento do estudante.

Para validar nossa hipótese, três métodos de recomendação foram comparados por meio de uma avaliação duplamente cega com alunos e professores. Os métodos experimentais variam em relação à definição de similaridade das questões, sendo o primeiro o Método Baseado no Enunciado (MBE), que recomenda problemas com enunciados semanticamente similares a um Problema Alvo (PA). Para tanto, foram utilizadas técnicas de Processamento de Linguagem Natural (PLN) para pré-processar os enunciados dos problemas e analisar a similaridade entre os exercícios. O segundo é o Método Baseado no Comportamento (MBC), que recomenda problemas similares com base em um grupo de métricas que medem o nível de esforço requerido para os alunos resolverem problemas. Essas métricas foram extraídas a partir da interação do usuário com a IDE do JO e são chamadas de comportamento *data-driven*. Por fim, como um *baseline* para os dois métodos anteriores, foi elaborado o Método com padrão Estocástico (ME) (*placebo*), que recomenda os problemas aleatoriamente a partir de listas de exercícios criadas e predefinidas por um conjunto de professores.

Resumidamente, as contribuições do presente trabalha são: i) propor e validar recomendações com uma análise profunda de comportamentos dirigidos aos dados de estudantes; ii) propor e validar pela primeira vez, no melhor do nosso conhecimento, o uso de PLN para recomendação de problemas em juízes online; iii) demonstração de como os métodos de recomendação são importantes para os alunos, pois levam a uma taxa de acerto maior e a um emprego de esforço mais adequado na resolução; iv) apresentação de como os métodos podem ser úteis para professores, auxiliando na sua árdua tarefa de criar listas de exercícios em turmas de programação.

2. Trabalhos Relacionados

A literatura apresenta diferentes abordagens para recomendar conteúdos disponíveis em sistemas de *e-learning* para turmas de programação [Yera e Martínez 2017, Fonseca et al. 2020, Pereira et al. 2020]. Por exemplo, [Paula et al. 2014] propuseram um sistema que recomenda tópicos relacionados a conteúdos de programação, deixando a cargo do aluno a tarefa de buscar exercícios de interesse dentro dos tópicos recomendados. O sistema realiza as recomendações com base em metadados dos problemas, isto é, em informações previamente anotadas por humanos como o nível de dificuldade e a catego-

ria dos problemas. Tal estratégia falha em escalabilidade, visto que as anotações exigem esforço manual e, muitas vezes, os problemas não são anotados com os metadados necessários. [Chau et al. 2017] realizaram uma análise estrutural de códigos-fonte feitos por alunos ao longo de uma disciplina de programação. A partir dela, foi desenvolvido um assistente de conteúdo que tem o objetivo de recomendar aos instrutores tópicos que devem ser cobertos dentro de uma disciplina de programação com temas necessários para criar uma unidade curricular adequada para a disciplina. Diferente de [Chau et al. 2017] e [Paula et al. 2014], nosso foco está na recomendação de *problemas* e não tópicos. Ademais, além dos códigos, os métodos deste estudo são construídos com base nos enunciados dos problemas e nas interações dos alunos com o JO.

[Hosseini e Brusilovsky 2017] utilizaram técnicas oriundas da área de processamento de linguagem natural em códigos-fonte para recomendar trechos úteis desses códigos para alunos durante o desenvolvimento de um determinado problema. Para tanto, os autores testaram diversas métricas de similaridade para mensurar a semelhança dos códigos. Após os testes, a similaridade de cosseno apresentou os melhores resultados. Seguindo esse caminho, o presente estudo também emprega similaridade de cosseno para realizar o cálculo de semelhança entre problemas dos sistemas de recomendação propostos. Além disso, acredita-se que nosso estudo é complementar ao trabalho de [Hosseini e Brusilovsky 2017], pois o presente trabalho tenta facilitar o processo de aprendizagem de programação através da recomendação de problemas adequados ao nível de conhecimento dos estudantes, enquanto que [Hosseini e Brusilovsky 2017] usa dicas com trechos de código para auxiliar no processo de resolução dos problemas.

Por fim, [Yera e Martínez 2017] usaram as interações dos alunos com JOs para construir uma matriz *user-problem M*, que incorpora as falhas e acertos dos alunos ao tentarem resolver um problema. Ambos os sistemas construíram a matriz utilizando uma tupla *(u, p, j)*, sendo que *u* representa o usuário, *p* o problema que está sendo resolvido e *j* o julgamento feito pelo JO. Com base nas tuplas e no número de tentativas de resoluções dos problemas, a matriz *M* é preenchida com valores categóricos do conjunto C = 0, 1, 2, 3 e 4. Para ilustrar, *M(u,p) = 0* significa que um usuário *u* nunca tentou resolver um problema *p* (ausência de esforço) e *M(u,p) = 4* significa que ele resolveu o problema *p* utilizando poucas tentativas (pouco esforço). Neste estudo, nós adaptamos as variáveis categóricas de [Yera e Martínez 2017] ao MBC e as estendemos com um conjunto de variáveis propostas por [Pereira et al. 2018, Pereira et al. 2019], que se mostraram eficientes para representar o esforço empregado pelo aluno para resolver problemas em JOs.

3. Metodologia

Este estudo utilizou dados abertos[1] coletados a partir do JO CodeBench, desenvolvido pelo Instituto de Computação (IComp) da Universidade Federal do Amazonas (UFAM). Os dados estão relacionados a diferentes turmas de introdução à programação do período de 2016 a 2018. Nas disciplinas, os alunos resolviam 7 listas de exercícios utilizando a IDE do JO. Nas próximas subseções serão apresentados os sistemas de recomendação propostos e seus fluxos de funcionamento.

Antes disso, é importante ressaltar que as recomendações realizadas pelos três métodos propostos neste estudo usam como referência um *Problema Alvo* (PA), que o aluno já resolveu previamente. Partimos da hipótese de que, se o aluno já resolveu o PA e o

[1] http://codebench.icomp.ufam.edu.br/dataset/

problema recomendado é *similar* ao PA, então o problema recomendado é provavelmente compatível com o nível de conhecimento do aluno. Assim sendo, o que também precisa ser definido é o conceito de *similaridade* e é nisso que os três métodos de recomendação diferem, conforme será explicado nas próximas subseções.

3.1. Método Baseado no Enunciado

Neste método, após o usuário submeter uma solução válida para um dado PA, o recomendador que possui um módulo de PLN vai analisar o enunciado do PA, que passará por um *pipeline*, para então representar o texto dos enunciados em vetores com valores quantitativos para o cálculo da similaridade de cosseno. Dessa forma, os vizinhos mais próximos ao PA serão os problemas recomendados.

Essas são as etapas do pipeline: ***rPunct***, remove os símbolos e pontuações que não agregam informações nas tarefas de recomendação; ***sNumber***, substitui os valores numéricos por #; ***rStopWords***, remove *stopwords*, que são palavras que não agregam e que afetam o desempenho do modelo; ***SpellChecker***[2], algoritmo probabilístico que corrige os erros ortográficos encontrados nos enunciados; ***Tokens***, *tokeniza* as palavras dos enunciados dos problemas; ***Sttemer***[3], padroniza as palavras, reduzindo-as ao seu radical (ex.: casa, casinha, casarão são reduzidas para cas); ***Continuos Bag of Words***, representação preditiva dos enunciados dos problemas em vetores representativos com valores contínuos (*word embeddings*), através do uso de uma rede neural profunda com uma camada de projeção, compartilhada por todas as palavras do enunciado; ***Padding***, padroniza os *word embeddings* adicionando zeros ao final deles, fazendo com que possuam o mesmo tamanho; ***Singular Value Decomposition***, fatoração da matriz com todos os *word embeddings* para redução da dimensionalidade.

Tais etapas foram realizadas para limpar os dados e retirar prováveis ruídos, mantendo a semântica e o contexto do enunciado. De fato, as práticas supracitadas são padrões para pré-processamento de texto em modelos de aprendizagem de máquina [Aljohani et al. 2020] e elas foram escolhidas após uma análise de uma série de combinações do estado da arte em PLN. Com efeito, elas serão úteis para estruturar o texto de modo que o algoritmo seja capaz de processá-lo e efetuar melhores recomendações. Perceba que o esperado com essas técnicas é que o MBE consiga extrair do PA a semântica e potencialmente o tópico do problema através da análise de similaridade dos *word embeddings* e, consequentemente, de contextos semelhantes e termos frequentes. Dessa forma, as recomendações também serão potencialmente do mesmo tópico e semanticamente semelhantes ao PA. Para ilustrar, problemas de condicionais (*if-then-else*) tipicamente compartilham semânticas similares e muitas vezes termos em comum. Por isso, o uso dessas técnicas de processamento de linguagem tem sido bem sucedido para a extração de tópico em diversos tipos textuais, incluindo na extração de contexto a partir de enunciados de problemas [Fonseca et al. 2020].

3.2. Método Baseado no Comportamento

O MBC mensura o esforço médio esperado para solucionar um problema. Após o aluno resolver o PA, um módulo *data-driven* analisa os *logs* das soluções já submetidas para o PA e depois busca problemas com médias de *features* similares para serem recomendados

[2]O post de Peter Norvig, no qual o algoritmo SpellChecker foi baseado pode ser encontrado em `https://norvig.com/spell-correct.html`.

[3]Função usada para Sttemer: https://www.nltk.org/_modules/nltk/stem/rslp.html

aos alunos. Dizemos que esse método é baseado no comportamento porque a similaridade é calculada a partir de métricas comportamentais dos alunos quando estão resolvendo os problemas na IDE. As *features* que representam tais métricas são provenientes dos estudos de [Pereira et al. 2018, Pereira et al. 2019] e [Yera e Martínez 2017].

Neste método, cada problema é representado por um vetor de N-dimensões, onde N é o número de *features* disponíveis. Cada dimensão do vetor de um problema representa o valor de uma *feature* obtida a partir dos *logs* gerados durante as tentativas dos alunos de solucionarem esse problema. Desse modo, se temos M problemas no JO e N *features*, ao final é gerada uma matriz M×N contendo todos os problemas. Como é comum em sistemas de recomendação que usam dados tabulares como esses, os valores das *features* nos vetores foram padronizados utilizando a técnica estatística *z-score*. No mais, usou-se o cosseno entre os vetores para calcular a similaridade entre os problemas.

```
2017-3-13 17:33:23.537#viewportChange#0
2017-3-13 17:33:26.038#change#{"from":{"line":1,"ch":7,"xRel":1},"to":{"line":1,"ch":11,"xRel":-1},"text":["t"],"removed":["soma"],"origin":"+input"}
2017-3-13 17:33:26.371#change#{"from":{"line":1,"ch":8},"to":{"line":1,"ch":8},"text":["o"],"removed":[""],"origin":"+input"}
2017-3-13 17:33:26.659#change#{"from":{"line":1,"ch":9},"to":{"line":1,"ch":9},"text":["t"],"removed":[""],"origin":"+input"}
2017-3-13 17:33:26.779#change#{"from":{"line":1,"ch":10},"to":{"line":1,"ch":10},"text":["a"],"removed":[""],"origin":"+input"}
2017-3-13 17:33:27.098#change#{"from":{"line":1,"ch":11},"to":{"line":1,"ch":11},"text":["l"],"removed":[""],"origin":"+input"}
```

Figura 1. Exemplo de *log* coletado. Adaptado de [Pereira et al. 2018].

Os dados utilizados por este método foram coletados durante o processo de desenvolvimento das soluções, onde todas as ações que os alunos realizavam na IDE foram registradas em um arquivo de *log* como é mostrado na Figura 1. No arquivo, é apresentado, por exemplo, o momento em que o aluno digitou a variável total, após remover a variável soma (2ª linha do log). Dessa forma, as seguintes *features* foram extraídas para cada problema no MBC: ***noAttempts***, proporção de alunos que não tentaram resolver um problema; ***unsucNoRes***, proporção de alunos que, após poucas[4] tentativas, não conseguiram resolver o problema; ***unsucRes***, proporção de alunos que, após muitas[5] tentativas, não conseguiram resolver o problema; ***sucNoRes***, proporção de alunos que, após poucas tentativas, resolveram o problema; ***sucRes***, proporção de alunos que, após muitas tentativas, resolveram o problema; ***attempts***, média de tentativa dos alunos para resolver o problema; ***IDEUsage***, tempo[6] médio em que os alunos utilizaram na IDE para resolver o problema; ***contCicle***, média de *loops* nos códigos submetidos; ***contCondition***, média de estruturas condicionais nos códigos submetidos; ***cyclomaticComplexity***, complexidade ciclomática[7] média nos códigos submetidos; ***averageLogRows***, média de linhas de *log* (Figura 1) geradas nas resoluções dos problemas; ***nDistinctOperands***, média de operandos aritméticos diferentes encontrados nos códigos submetidos; ***nDistinctOperators***, média de operadores aritméticos diferentes encontrados nos códigos submetidos; ***quocienteErro***, média de erros repetidos pelos alunos; ***quocienteWatson***, tempo médio que os alunos demoram entre erros consecutivos para o mesmo problema; ***sloc***, média de linhas dos códigos submetidos; ***test***, média de vezes que os alunos testaram os códigos antes da submissão; ***sucessAverage***, média de sucesso de envio de resposta; ***totalOperands***, média de operandos encontrados nos códigos; ***totalOperators***, média de operadores encontrado nos códigos; ***variable***, média de variáveis encontradas nos códigos.

[4]Adotamos como poucas tentativas um valor abaixo de 8 tentativas. Esse valor foi escolhido com base no estudo de [Yera e Martínez 2017].

[5]Adotamos como muitas tentativas um valor maior ou igual a 8.

[6]Removeu-se o tempo de inatividade do aluno (60 segundos sem nenhuma atividade na IDE).

[7]Número de caminhos independentes do grafo de fluxo correspondente ao código.

4. Contexto Experimental

Neste estudo, foi utilizado um sistema de recomendação com padrão estocástico para servir de *baseline* para os métodos propostos. O Método com padrão Estocástico (ME) consiste em buscar de forma aleatória, em listas de exercícios pré-dispostas por professores, um problema para recomendar aos alunos após eles resolverem um PA no JO. As listas pré-dispostas foram criadas por um conjunto de docentes em disciplinas de introdução à programação de computadores na UFAM. Os exercícios das listas estão relacionados com tópicos de programação que vão desde variáveis e estrutura sequencial a vetores e matrizes. Dado que a similaridade de exercícios no ME é definida por aleatoriedade, esse método funciona como um placebo na comparação com o MBE e MBC.

Utilizou-se o ME como baseline porque ele pode ser visto como a ausência de um sistema de recomendações. Para ilustrar, após o aluno resolver um problema qualquer (PA) em um juiz online, o processo de busca de uma próxima questão é potencialmente aleatório, já que nos juízes online pesquisados, com o melhor do nosso conhecimento, não existe este recurso de recomendação automática.

As recomendações do MBE, MBC e ME foram avaliadas por 3 professores universitários que já ministraram disciplinas de programação e 15 acadêmicos (ambos da Universidade Federal de Roraima) que já possuem créditos em introdução à programação, isto é, 18 indivíduos participaram do experimento. Para realizar as avaliações, preparou-se listas de exercícios personalizadas para cada participante. Perceba que os PAs servem como um o *cold-start* para as recomendações, ou seja, a partir das características deles é que o sistema de recomendação busca problemas similares para serem recomendados. Assim, para a formação das listas, os PAs foram selecionados manualmente pelos autores seguindo os critérios: i) cada lista de recomendação possuiria dois PAs; ii) o primeiro PA possuiria um tópico de problema fácil (operação, entrada e saída, condicionais) e o segundo, um tópico de problema intermediário (repetição, vetores); iii) para cada PA, eram geradas três recomendações. Dessa forma, cada lista de exercícios possuía 8 problemas que seguiam a seguinte ordem: 1 PA fácil seguido por 3 recomendações referentes ao PA fácil; 1 PA intermediário seguido por 3 recomendações referentes ao PA intermediário. Assim, cada participante analisou 24 problemas (8 problemas para cada um dos métodos), totalizando 324 (24×18) questões avaliadas. A avaliação aconteceu de modo duplamente cego, isto é, nem os participantes sabiam qual método de recomendação estavam avaliando, nem os pesquisadores sabiam a identidade dos participantes.

Para diversificar as listas de recomendações dos participantes, utilizou-se duas estratégias. Primeiro, foram selecionados em cada método 5 pares de PAs fáceis e intermediários. Dessa forma, a partir de cada par de PAs, é possível montar uma lista de recomendação. Segundo, para cada par de PAs, foram ranqueados os 10 problemas mais similares de acordo com a definição de similaridade de cada método de recomendação. Depois disso, foram escolhidos, aleatoriamente, dentro desse top 10, os problemas recomendados que iriam compor as listas de cada participante.

Os 3 professores realizaram a avaliação das listas de exercícios personalizadas através de um survey[8], onde eles liam o problema alvo e cada problema recomendado e depois avaliavam a recomendação em uma escala Likert, que variava de muito ruim (1) a muito bom (5). Já para os alunos foi pedido que eles resolvessem toda a lista de exercícios.

[8]Clique para visualizar o formulário.

Os alunos avaliavam a recomendação (muito ruim (1) a muito bom (5)) ao final de cada solução. Além disso, para os alunos, foi realizada uma análise *data-driven* do processo de resolução dos problemas recomendados. Para tanto, extraiu-se as seguintes métricas a partir dos *logs* e códigos dos alunos: número de tentativas (*attempts*), tempo de uso da IDE *online* (*IDEUsage*), a média de sucesso de envio de resposta (*sucessAverage*), o número de linhas de logs (*averageLogRows*) e o tamanho do código fonte (*sloc*). Essas métricas foram escolhidas porque [Pereira et al. 2018] apontam que elas podem ser usadas para mensurar de modo holístico o sucesso e o esforço dos alunos ao resolverem problemas de programação em JOs. Finalmente, o sucesso do aluno também é calculando com base nas taxas de acerto, erro e desistência.

5. Resultados e Discussões

A Figura 2 (A) apresenta o *likert plot* das respostas dos alunos quando perguntados qual a sua avaliação sobre as recomendações, considerando o problema alvo resolvido anteriormente. Na figura, pode-se observar que o MBE possui 47,8% das avaliações nas classes positivas (bom e muito bom), enquanto 25,6% das opiniões ficaram entre as categorias negativas (muito ruim e ruim). Também houve 26,7% de avaliações na classe intermediária (regular), o que pode indicar que os alunos tiveram dúvidas ao avaliar as recomendações dadas. Já o MBC detém o melhor resultado em relação aos demais métodos, conseguindo atingir 63,3% das avaliações nas classes positivas (bom e muito bom). Em contrapartida, o MBC é o método com mais pareceres na categoria muito ruim, atingindo até 11,1%.

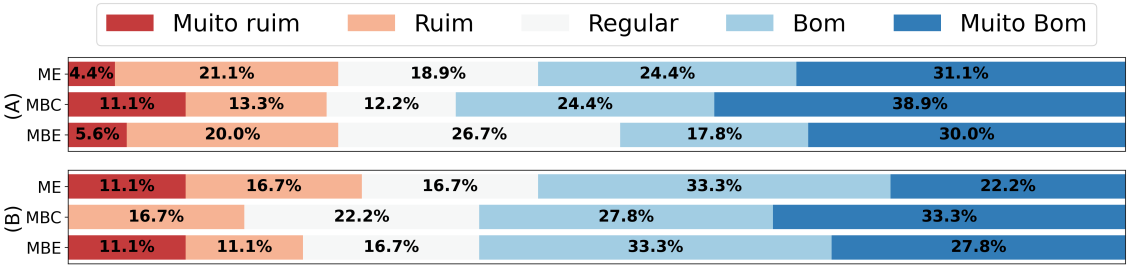

Figura 2. (A) Avaliações dos alunos; (B) Avaliações dos professores.

No ME, as avaliações positivas (muito bom e bom) foram de 55,5%, valor superior ao do MBE. Além disso, o ME e MBE tiveram ∼25% das opiniões nas categorias negativas (muito ruim e ruim). Dessa maneira, é possível observar que as recomendações do MBE nas avaliações dos 15 estudantes não foram satisfatórias.

Para atestar as avaliações dos alunos, foram analisadas as opiniões do professores de programação. Logo, pode ser observado na Figura 2 (B) que o MBC obteve uma melhor avaliação em relação aos demais métodos. Entretanto, diferente da avaliação dos alunos, o MBE supera o ME. Mais especificamente, o ME obteve 55,5% de avaliações positivas (muito bom e bom) e 27,8% de avaliações nas classes negativas (muito ruim e ruim). De fato, o MBE e MBC obtiveram 61,1% de avaliações positivas (muito bom e bom), sendo que o MBC não teve nenhuma avaliação na classe muito ruim e apenas 16,7% de opiniões ruins. Nota-se a discordância em relação à avaliação dos professores de programação e dos alunos em relação ao MBC, pois na avaliação dos alunos os resultados apontaram que o MBC era o que mais possuía avaliações na classe muito ruim. Além disso, também pode ser notado que o MBE, quando comparado com o ME, conseguiu obter mais avaliações positivas do ponto de vista dos professores.

Adicionalmente, o sucesso das submissões dos alunos para os problemas recomendados também foi analisado, a fim de atestar se as avaliações dos alunos tiveram algum viés. Logo, são apresentados na Figura 3 (A): i) a taxa de acertos, que é o número de problemas recomendados com submissões corretas enviadas pelos alunos, dividido pelo total de problemas recomendados; ii) a taxa de erros, similar ao item *i)*, só que levando em consideração as submissões incorretas; e iii) a taxa de problemas não tentados, similar ao item i), só que levando em consideração os problemas que o aluno desistiu de resolver sem nenhuma submissão. Pode-se observar que o MBE e MBC tiveram mais de 60% das recomendações respondidas corretamente, enquanto que apenas 25% das recomendações do ME foram enviadas corretamente. Também nota-se que 51% dos problemas no ME não foram tentados. Essas informações podem indicar que as recomendações dadas pelo ME eram mais complexas e menos adequadas aos alunos, o que mostra uma incoerência com as avaliações aferidas a partir dos questionários aplicados aos alunos para este método (Figura 2 (A)). De fato, após realizar o teste Qui-quadrado para comparar as taxas supracitadas dos três métodos, observou-se que o MBC e MBE são estatisticamente superiores ao ME, mesmo após a correção de Bonferroni (p-value $<$ 0,05/3). No mais, a diferença entre o MBC e MBE não é estatisticamente significante.

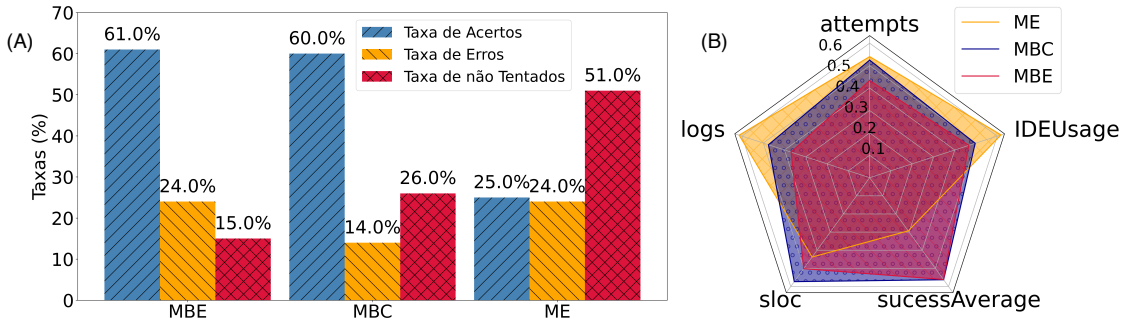

Figura 3. (A) Taxa de recomendações enviadas corretamente, incorretamente e taxa de recomendações não tentadas; (B) Análise dos *logs* obtidos na avaliação dos alunos de programação

Ainda neste estudo, foi mensurado o esforço exigido para o aluno responder os problemas recomendados usando métricas propostas por [Pereira et al. 2018]. Os valores médios encontrados de cada método podem ser vistos na Figura 3 (B). Ao comparar os métodos observa-se que os alunos permaneceram mais tempo programando na IDE (maior *IDEUsage*) nos exercícios recomendados pelo ME. Além disso, foram geradas mais linhas de *logs* (maior *averageLogRows*) enquanto eles resolviam os problemas do ME, o que sugere que eles estavam reescrevendo o código, várias vezes. Observa-se ainda que os códigos submetidos pelos alunos nas recomendações do ME eram menores (menor valor do *sloc*), o que mostra que os estudantes tiveram que empregar mais esforço (maior *IDEUsage*, *averageLogRows* e *attempts*) para submeter soluções com menos linhas de código. Adicionalmente, no ME os alunos tiveram mais submissões (maior *attempts*) e uma taxa menor de sucesso por tentativa (menor *successAverage*), o que é mais uma evidência de que os alunos tiveram mais dificuldade ao responderem os problemas recomendados pelo ME. De fato, as diferenças citadas são estatisticamente significantes (p-value $<$ 0,05/3 – Mann-Whitney pareado) para todas as métricas, mesmo após a correção de Bonferroni. Assim, demonstra-se uma superioridade das recomendação do MBC e MBE em relação ao *baseline* (ME). Além disso, novamente não se achou diferenças estatisticamente significantes entre o MBC e MBE em relação a essas métricas de esforço.

6. Considerações Finais

Neste estudo, são propostos e avaliados métodos de recomendação de problemas para JOs. Podemos considerar que a hipótese que norteia a construção dos métodos propostos é válida, ou seja, é verdadeiro que se o aluno resolveu o PA e o problema recomendado é similar ao PA, logo o problema recomendado é provavelmente compatível com o nível de conhecimento do estudante. Afirmamos isso porque os dois métodos propostos (MBC e MBE) apresentaram recomendações mais adequadas do que o *baseline* em termos de esforço empregado ($p - value < 0,05$ - *Mann-Whitney*) e sucesso obtido (maior taxa de acerto e menor taxa de erro e desistência).

Destacam-se duas aplicações imediatas dos métodos: uma perspectiva mais voltada ao professor e outra direcionada ao aluno. Os professores tipicamente criam variações de listas de exercícios sobre um mesmo tópico para diferentes turmas e em diferentes semestres [Bradley 2020], a fim de evitar o plágio. Considere cada problema de uma lista de exercícios já criada por um professor como um problema-alvo. Ao gerar N recomendações para cada um desses problemas, podemos compor N novas listas de exercícios semanticamente similares à lista original (MBE), ou que requerem esforço e conhecimentos similares aos exigidos para resolver a original (MBC). Do lado do aluno, os métodos facilitam a penosa tarefa de achar problemas adequados ao seu nível de conhecimento, promovendo assim uma auto-regulação no processo de aprendizagem.

Por fim, observa-se que ambos os métodos propostos (MBC e MBE) possuem pontos fortes e fracos. Para ilustrar, o MBC apresenta como grande benefício o fato de modular o esforço esperado para o aluno resolver um problema e, assim, recomendar problemas com esforço esperado similar. Entretanto, modular apenas o esforço pode ser insuficiente, pois, além dele, também são necessárias habilidades prévias para que o aluno resolva um problema. Por exemplo, utilizando a biblioteca *numpy* do Python, o esforço esperado para um aluno resolver um problema de somar dois vetores e um problema para somar dois escalares é potencialmente bastante similar, já que ambas as operações são implementadas de modo praticamente idêntico através da equação $a = b+c$, onde b e c são escalares ou vetores. Entretanto, para o aluno conseguir resolver o problema de vetores ele precisa ter conhecimento prévio nesse tópico de programação. Perceba que o MBE tem como ponto forte o fato de potencialmente extrair o tópico (e.g. variáveis, condicionais, vetores, etc.) de um problema alvo. Assim, as recomendações serão potencialmente do mesmo tópico do problema alvo. Entretanto, mesmo problemas do mesmo tópico podem requerer do aluno esforços muito diferentes. Nesse sentido, como trabalhos futuros, tem-se como objetivo propor e validar um método híbrido, que realize as recomendações com base no conteúdo (e.g. condicional, repetição, matrizes, etc.) relacionado ao problema alvo (similar ao MBE) e no esforço esperado para resolvê-lo (similar ao MBC).

7. Agradecimentos

Esta pesquisa, realizada no âmbito do Projeto Samsung-UFAM de Ensino e Pesquisa (SUPER), nos termos do artigo 48 do Decreto nº 6.008/2006 (SUFRAMA), foi parcialmente financiada pela Samsung Eletrônica da Amazônia Ltda., nos termos da Lei Federal nº 8.387/1991, por meio dos convênios 001/2020 e 003/2019, firmados com a Universidade Federal do Amazonas e a FAEPI, Brasil. O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

Referências

Aljohani, T., Pereira, F. D., Cristea, A. I., e Oliveira, E. (2020). Prediction of users' professional profile in moocs only by utilising learners' written texts. In *International Conference on Intelligent Tutoring Systems*, pages 163–173. Springer.

Bradley, S. (2020). Creative assessment in programming: Diversity and divergence. In *Proceedings of the 4th Conference on Computing Education Practice 2020*, pages 1–4.

Chau, H., Barria-Pineda, J., e Brusilovsky, P. (2017). Content wizard: concept-based recommender system for instructors of programming courses. In *Adjunct Publication of the 25th Conference on User Modeling, Adaptation and Personalization*, pages 135–140. ACM.

Dwan, F., Oliveira, E., e Fernandes, D. (2017). Predição de zona de aprendizagem de alunos de introdução à programação em ambientes de correção automática de código. In *Simpósio Brasileiro de Informática na Educação (SBIE)*, volume 28, page 1507.

Fonseca, S., Pereira, F. D., Oliveira, E. H. T., Olveira, D. B. F., Carvalho, L. G., e Cristea, A. I. (2020). Automatic subject-based contextualisation of programming assignment lists. In *Proc. of 13th International Conference on Educational Data Mining*, pages 81–91. EDM.

Hosseini, R. e Brusilovsky, P. (2017). A study of concept-based similarity approaches for recommending program examples. *New Review of Hypermedia and Multimedia*, 23(3):161–188.

Paula, L. C., Oliveira Fassbinder, A. G., e Barbosa, E. F. (2014). A recommendation system to support the students performance in programming contests. In *2014 IEEE Frontiers in Education Conference (FIE) Proceedings*, pages 1–8. IEEE.

Pereira, F. D., Oliveira, E. H., Fernandes, D., e Cristea, A. (2019). Early performance prediction for cs1 course students using a combination of machine learning and an evolutionary algorithm. In *2019 IEEE 19th International Conference on Advanced Learning Technologies (ICALT)*, volume 2161, pages 183–184. IEEE.

Pereira, F. D., Oliveira, E. H. T., Oliveira, D., Cristea, A. I., Carvalho, L., Fonseca, S., Toda, A., e Isotani, S. (2020). Using learning analytics in the Amazonas: understanding students' behaviour in CS1. *British journal of educational technology.*

Pereira, F. D., Oliveira, E. H. T., e Oliveira, D. F. B. (2018). Uso de um método preditivo para inferir a zona de aprendizagem de alunos de programação em um ambiente de correção automática de código. Mestrado em informática, Universidade Federal do Amazonas, Manaus.

Yera, R. e Martínez, L. (2017). A recommendation approach for programming online judges supported by data preprocessing techniques. *Applied Intelligence*, 47(2):277–290.

Zhao, W. X., Zhang, W., He, Y., Xie, X., e Wen, J.-R. (2018). Automatically learning topics and difficulty levels of problems in online judge systems. *ACM Transactions on Information Systems (TOIS)*, 36(3):1–33.