Tempos de Transição em Estados de Corretude e Erro como Indicadores de Desempenho em Juízes Online

Ingrid Lima dos Santos[1], David B. F. Oliveira[1], Leandro S. G. Carvalho[1], Filipe Dwan Pereira[2], Elaine H. T. Oliveira[1]

[1]Instituto de Computação – Universidade Federal Amazonas (UFAM)

[2]Departamento de Ciência da Computação – Universidade Federal de Roraima (UFRR)

{ils,david,galvao,elaine}@icomp.ufam.edu.br, filipe.dwan@ufrr.br

Resumo. *Turmas de Introdução à Programação de Computadores (IPC) tipicamente apresentam altos índices de reprovação. Com efeito, muitos estudos vêm sendo conduzidos para entender de modo holístico como os alunos desenvolvem soluções e quais comportamentos podem ter um impacto positivo ou negativo no desempenho deles. Nesse sentido, este estudo propõe e valida um modelo de transições de estados em um Juiz Online com ênfase em turmas de IPC. O modelo leva em consideração o tempo de transição de estados de corretude e erro das soluções desenvolvidas e avaliadas por 489 alunos de 9 turmas de IPC em um Juiz Online. Como resultado, pode-se verificar uma correlação moderada para alta entre o tempo gasto em transições de submissão incorreta e correta e o desempenho final do estudante de IPC.*

Abstract. *Introduction to Programming (CS1) classes typically show high failure rates. Indeed, many studies have been carried out to understand how students develop code solutions and which behaviors may impact positively or negatively on their performance. In this sense, this study proposes and validates a model of state transitions in an Online Judge (OJ) with an emphasis on CS1 classes. The model takes into account the transition time of error and correctness states of the solutions developed by 489 students from 9 CS1 classes in an OJ. As a result, we could find a moderate correlation between the time spent on incorrect and correct transitions, and the CS1 student final grade.*

1. Introdução

Disciplinas de Introdução à Programação de Computadores (IPC) tendem a apresentar altos índices de reprovação [Watson and Li 2014], [Dwan et al. 2017], [Robins 2019]. Na literatura, muitos autores argumentam que um dos motivos para esses índices é o fato de a disciplina ser complexa para alunos iniciantes, já que ela demanda a mobilização de várias habilidades cognitivas, que vão desde a compreensão e análise de aspectos importantes de um contexto (abstração) até escrever o código da solução em uma linguagem formal com vocabulário restrito e não ambíguo [Pereira et al. 2018]. Além disso, existe um consenso na literatura de que, para aprender a programar, é necessário prática, isto é, resolver muitos exercícios de programação [Robins 2019]. Com efeito, muitas instituições de ensino vêm combinando aulas presenciais com práticas de programação através de ferramentas como Juízes Online (JO) [Wasik et al. 2018], [Fonseca et al. 2019].

Tais ferramentas também podem ser denominadas por ambientes de correção automática de código (ACAC) e, neste trabalho, os dois termos serão usados como

DOI: 10.5753/cbie.sbie.2020.1283

sinônimos. Eles permitem que o professor disponibilize exercícios de programação para os alunos solucionarem e submeterem seus códigos para correção no próprio JO [Wasik et al. 2018]. Uma vez que o código é submetido, o sistema retorna um *feedback* imediato, informando ao aluno se sua solução está correta ou incorreta. Além disso, com o JO surgem novas oportunidades de estudo na área de mineração de dados educacionais, pois as interações dos alunos com o sistema possibilitam a coleta automatizada de dados e, subsequentemente, sua análise [Fonseca et al. 2019], [Wasik et al. 2018].

Dentro desse escopo, trabalhos como o de Dwan et al. (2017) e Carter et al. (2015) utilizam os dados coletados durante as aulas para inferir a nota final dos estudantes em disciplinas de programação. No estudo de Dwan et al. (2017), os autores propuseram e validaram um método para predizer a zona de aprendizagem de alunos de IPC que aprendem a programar em Python e usam uma metodologia de aprendizagem híbrida com ACAC. Para isso, os autores analisaram uma combinação de 17 atributos coletados a partir dos logs de um ACAC. O modelo preditivo construído obteve 78,3% de acurácia nas primeiras duas semanas de aula em um experimento com a base de dados desbalanceada. E em outro experimento com a base balanceada, a precisão foi de 72,8%. Por outro lado, Carter et al. (2015) desenvolveram um modelo preditivo baseado no tempo gasto em um conjunto de estados de programação derivados da correção sintática e semântica de um programa. Os estudos foram conduzidos com turmas de CS2 (que costuma ser o curso que vem na sequência de CS1), onde o modelo proposto obteve entre 36% e 67% de acurácia. As turmas utilizavam diferentes linguagens de programação (C/C++ e Java) em diferentes ambientes de desenvolvimento (Visual Studio ou BlueJ).

Como o trabalho de Carter et al. (2015) foi executado em um contexto bem diferente do de Dwan et al. (2017), é interessante verificar se o modelo proposto é adaptável a outros contextos educacionais como, por exemplo, turmas de IPC que aprendem a programar com a linguagem Python e que utilizam metodologias de aprendizagem híbrida com ACAC. Sendo assim, este trabalho busca propor um modelo de estados adaptado de Carter et al. (2015) para o contexto de Dwan et al. (2017), afim de verificar a eficácia no uso de um modelo de estados para determinar a correlação entre tempos de transição em estados corretos e estados incorretos e o desempenho final.

2. Trabalhos Relacionados

O potencial da análise e mineração de dados – metodologias que extraem informações úteis e acionáveis de grandes conjuntos de dados – transformou um campo de investigação científica após o outro [Baker and Inventado 2014]. Com a utilização dessas metodologias, é possível explorar grandes bases de dados para identificar padrões que ocorrem em apenas um pequeno grupo de estudantes ou então apenas esporadicamente, analisar como o design de ambientes de aprendizagem podem impactar variáveis de interesse. Visando compreender mais profundamente os fenômenos e a complexidade dos processos que envolvem o processo de ensino e aprendizagem da computação, muitos pesquisadores têm adotado e modificado métodos originários da mineração de dados convencional, sendo os principais: a predição; o agrupamento e a mineração de relações [Baker and Inventado 2014, Pereira et al. 2020].

Mais particularmente, muitos trabalhos têm proposto o uso de técnicas de aprendizado de máquina para predizer o desempenho final de alunos em turmas de programação [Pereira et al. 2020]. Dwan et al. (2017) e Alves et al. (2019) realizaram seus trabalhos com turmas de IPC que utilizam um ACAC. Dwan et al. (2017) focou em determinar as

zonas de aprendizagem dos alunos, construindo um perfil de programação baseado nos registros deixados por eles à medida que resolviam os exercícios. Para isso, foi analisada uma combinação de 17 atributos. Semelhantemente, Alves et al. (2019) buscou inferir o desempenho final dos estudantes utilizando apenas os dados relativos ao comportamento do mesmo em um ACAC, mas para tal foram extraídos 8 atributos. Em ambos os estudos, os autores uniram os atributos com a nota das avaliações e introduziram em um classificador *Random Forest* obtendo precisões de até 86% [Dwan et al. 2017] e acima de 80% [Alves et al. 2019]. Como uma extensão do trabalho de Dwan et al. (2017), Pereira et al. (2019) atingiram uma acurácia média de aproximadamente 83% na classificação utilizando algoritmos genéticos para otimização do classificador, usando dados do início do curso.

Carter et al. (2015) analisou o processo de programação dos alunos e desenvolveu um modelo de estado de programação normalizado (*Normalized Programming State Model* – NPSM) para descrever vários estados de correção do programa (sintático ou semântico) e as transições entre esses estados. Eles especularam que o tempo relativo gasto em estados corretos e incorretos pode ser um indicador do desempenho geral dos alunos no curso. A intuição desse método é que os alunos que cometem muitos erros de sintaxe provavelmente têm seus programas em estados incorretos a maior parte do tempo, enquanto estudantes que estão trabalhando para solucionar erros de tempo de execução provavelmente estão trabalhando e testando programas que, ao menos, estão corretos sintaticamente. A partir do modelo NPSM, os autores puderam identificar 4 atributos diretamente ligados ao bom desempenho do aluno, e então chegaram a uma fórmula preditiva que obteve precisão entre 36% (para a amostra de dados referente às primeiras atividades do semestre) e 67% (referente a todas as atividades do semestre).

Em seu estudo, Jadud (2006) detectou um ciclo de edição, compilação e execução de código dos alunos em turmas de IPC que utilizavam a linguagem Java. Com isso, o autor propôs um algoritmo para quantificar os erros de compilação usando a métrica *Error Quotient* (EQ), que é baseada no número de submissões subsequentes que retornam o mesmo erro de compilação, e é capaz de indicar a dificuldade ou a destreza do aluno em corrigir os eventuais erros encontrados. Verificou-se ainda que o EQ possui correlação com a nota do aluno. Tabanao et al. (2011) utilizaram o EQ em turmas de IPC que utilizavam a linguagem Java para quantificar o comportamento de compilação dos alunos, além disso também derivaram os erros encontrados e o tempo entre as compilações dos registros de dados. Com isso, os autores construíram um modelo de regressão linear baseado no EQ, nos erros encontrados, no tempo entre as compilações e as notas dos exames de meio de semestre dos alunos. Assim, os autores constataram que os erros encontrados pelos alunos podem afetar negativamente seu desempenho, mas também que um tempo maior entre as compilações produz efeito positivo no desempenho final.

Apesar dos trabalhos de Dwan et al. (2017) , Alves et al. (2019) e Pereira et al. 2019 apresentarem contribuições na predição precoce do desempenho dos alunos de IPC, os atributos utilizados por ambos são baseados tão somente em médias e não levam em consideração o tempo que os alunos passam em estados de transição de corretude e erro de suas soluções. O estudo de Jadud (2006) apresenta contribuições no processo de identificar a dificuldade ou destreza do aluno na resolução de erros levando em consideração apenas a quantidade dos erros de compilação. Enquanto isso, o estudo de Tabanao et al. (2011) apresenta contribuições na detecção do impacto dos erros dos alunos no desempenho final, levando em consideração o tempo entre compilações de código. Seguindo uma

linha diferente, o trabalho de Carter et al. (2015) busca predizer o desempenho dos alunos levando em consideração o tempo gasto entre a transição de estados corretos e incorretos, baseado no modelo de estados desenvolvido ao observar o comportamento que os alunos apresentam durante o processo de resolução das atividades propostas. Nesse caminho, o presente trabalho apresenta um modelo de estados adaptado para um JO utilizado por turmas de IPC e analisa a correlação entre o desempenho dos estudantes na disciplina e o tempo despendido por eles entre as transições de estados de corretude e de erro enquanto resolvem exercícios de programação.

3. Metodologia

3.1. Instrumento

Sistemas de Juízes Online têm como propósito compilar, executar e testar códigos-fonte realizando a avaliação automática desses códigos [Wasik et al. 2018].

O JO utilizado neste trabalho visa automatizar a correção dos exercícios de programação e auxiliar no processo de aprendizado. Nele, os professores podem disponibilizar exercícios de programação aos alunos, que por sua vez podem codificar soluções e submetê-las através da interface do sistema. Assim que o aluno submete uma solução para um exercício, o JO informa se ela está correta ou não.

Para julgar a corretude de um código, o JO CodeBench segue dois passos principais:

1. **Análise sintática do código**, que verifica se o código submetido possui algum erro sintático, de acordo com a gramática da linguagem de programação adotada na disciplina. Caso haja algum problema sintático com o código, o aluno é imediatamente informado sobre a natureza do problema, bem como a linha do código em que ele se encontra.
2. **Análise lógica do código**, que verifica se o código desenvolvido pelo aluno é capaz de solucionar corretamente o problema proposto pelo professor.

A análise lógica dos códigos é feita por meio de casos de testes cadastrados pelo professor ao criar um exercício de programação. Cada caso de teste possui dois valores: um valor de entrada, passado como entrada para o programa do aluno, e um valor de saída, que é a saída correta para o valor de entrada informado.

Caso o sistema notifique que o código-solução não está correto, seja por erros de sintaxe ou de lógica, o aluno poderá rever seu código e ressubmetê-lo quantas vezes desejar, até o prazo estipulado pelo professor ao cadastrar a questão. A política de ressubmissão de códigos incentiva o aluno a identificar e solucionar erros por si próprio, ao contrário da abordagem tradicional, em que o aluno precisa esperar pela correção do professor, sem oportunidade de identificar os erros e corrigir seus códigos.

3.2. Base de Dados

Os dados[1] utilizados neste estudo foram coletados a partir de soluções e submissões a atividades propostas no JO CodeBench em turmas de IPC. Há dois tipos de atividades: exercícios práticos e exames. Os exercícios práticos consistem em questões relacionadas a um dado tópico abordado durante as aulas de IPC, com prazo de quinze dias para serem

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

resolvidos. Os exames são compostos por duas ou três questões, com duração fixa de duas horas para serem realizados, e ocorrem em laboratórios de informática sob a fiscalização do professor. A nota final na disciplina corresponde a uma média ponderada de sete exames e a média aritmética dos exercícios práticos. Para ser aprovado, o aluno deve obter uma nota final maior ou igual a 5,0.

Os dados utilizados neste trabalho foram coletados no primeiro semestre de 2019 da Universidade Federal do Amazonas, abarcando 11 cursos de graduação presencial nas áreas de Ciências Exatas e Engenharia. A Tabela 1 apresenta as principais características dessa base de dados.

Tabela 1. Caracterização do dataset utilizado.

Atributos	Quantidade
Nº de turmas	9
Nº de estudantes	489
Nº de exercícios práticos	1.559
Nº de exercícios de exames	176
Nº de códigos submetidos	38.417
Nº de testes e submissões	540.537

Para representar o comportamento dos alunos na plataforma e então extrair os atributos, optou-se por utilizar dois tipos de logs gerados pelo JO. O primeiro tipo armazena todas as interações dos estudantes com o IDE (Integrated Development Environment), tais como: teclas pressionadas, clique de mouse, cópia e colagem de texto, etc. O segundo tipo de log armazena todas as submissões e execuções de códigos feitas pelos estudantes, tais como: erros de compilação, hora da submissão ou teste do código, resultados gerados pelo código e mensagens de respostas incompletas.

Os dados desses dois tipos de logs foram coletados e analisados a fim de identificar quais deveriam ser transformados, agrupados ou removidos. A proposta inicial desta pesquisa era utilizar todos os exercícios disponíveis. Contudo, os exercícios práticos foram desconsiderados, visto que boa parte deles estava incompleta ou não respondida. Além disso, havia casos de códigos submetidos com um valor baixo de logs gerados, o que pode sugerir plágio, conforme explicado por [Pereira et al. 2020]. Dessa forma, foram considerados apenas os dados gerados a partir dos exames, pois têm alto impacto na nota final e são realizados em um ambiente controlado, contando com a presença do professor no laboratório, e tempo determinado para concluir a avaliação.

4. Modelo de Estados de Transição de Corretude e Erros

Com base na coleção de dados descrita na seção anterior, foram identificados os estados e as transições que se destacam em relação ao comportamento do aluno no ACAC. A partir deles, o modelo de estados foi construído conforme apresentado na Figura 1 e com ele identificados 4 atributos relacionados ao tempo de duração das transições, em minutos.

Os estados e suas transições ocorrem da seguinte forma: **IDE** representa o estado em que o estudante está no ambiente integrado de desenvolvimento codificando uma solução para uma questão. Ao finalizar a codificação, é possível realizar duas transições: **#saída_testar** ou **#submissão_solução**. A transição **#saída_testar** ocorre quando o estudante opta por realizar apenas o teste da solução, passando ao estado **TESTE**. Nesse estado, podemos obter dois tipos de resposta: i) **#retorna_saída**, que significa que o código

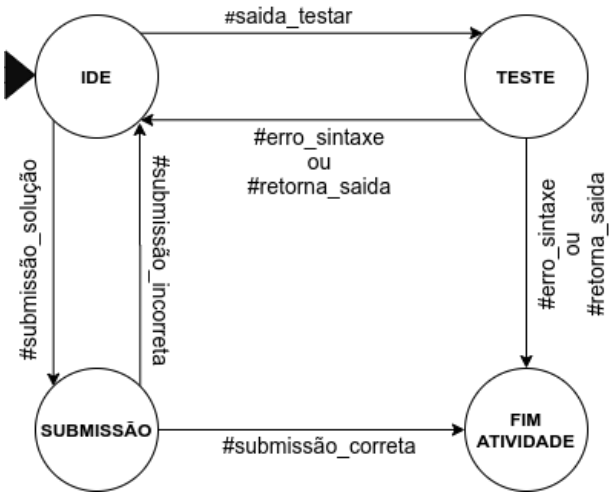

Figura 1. Modelo de estados que representa a resolução de um problema de programação usando um Juiz Online

é executado sem erros de sintaxe; e ii) **#erro_sintaxe**, quando o estudante é notificado que sua solução possui algum erro de sintaxe. Nesse caso, o compilador da linguagem indica o tipo de erro e sua localização no código. Aqui temos a extração do atributo **duração_testes** – correspondente ao tempo em que o estudante passa realizando testes na sua solução até um estado de corretude ou de erro – e do atributo **duração_erro_sintaxe** – correspondente ao tempo gasto entre a transição de um teste com erro de sintaxe até uma execução sem erros de sintaxe. Em ambas as transições, o aluno também pode optar por sair do JO, assim chegando ao estado FIM ATIVIDADE.

A transição **#submissão_solução** para o estado SUBMISSÃO representa o momento em que o estudante envia sua solução para avaliação, quando o ACAC verifica se o código passa nos casos de teste cadastrados pelo professor. Se a solução for aceita em todos os casos de teste, o aluno recebe um feedback positivo, sinalizado nos logs pela transição **#submissão_correta**, e sai da atividade. Contudo, caso a solução não passe em um ou mais casos de teste, o aluno recebe um feedback sinalizado nos logs pela transição **#submissão_incorreta**. Nesse último caso, o aluno retorna à **IDE** para revisar e corrigir seu código. Para tanto, temos aqui os atributos **duração_submissão_correta**, que é o tempo despendido entre a transição do momento de codificação na IDE e uma submissão 100% correta e **duração_submissão_incorreta**, que é o tempo despendido entre a transição de uma submissão incorreta e uma submissão 100% aceita.

5. Resultados

5.1. Análise Descritiva das Variáveis de Transição

A Tabela 2 apresenta algumas estatísticas descritivas das variáveis de transição analisadas e as notas finais. Em geral, observa-se que as distribuições possuem assimetria moderada ($|0{,}5| < A < |1{,}0|$) ou baixa ($A < |0{,}5|$). Para ilustrar, podemos observar uma assimetria positiva (cauda à direita) da **duração_testes**. Isso indica que essa variável possui uma frequência maior de valores baixos, o que pode ser confirmado visualizando a mediana e os valores dos percentis dessa coluna na Tabela 2, considerando também os valores máximos e mínimos. Isso evidencia que, após realizar alguns testes (execuções do código), tipicamente os estudantes submetem suas soluções rapida-

mente para a avaliação do JO. Também se observa assimetria positiva na distribuição da **duração_submissão_correta**, entretanto, com um desvio padrão maior e um alto coeficiente de variação, indicando uma alta heterogeneidade no tempo (minutos) despendido pelos estudantes para sair de um estado de solução incorreta para um código 100% aceito.

Por outro lado, observa-se uma assimetria negativa na duração_erro_sintaxe e na duração_submissão_correta, o que indica uma maior concentração de valores mais altos nas distribuições. Com isso, em uma análise univariada dessas variáveis, pode-se observar que poucos estudantes resolvem rapidamente os problemas (tempo mais baixo de duração_submissão_correta) e que uma parcela menor de alunos saem rapidamente de um estado de erro de sintaxe para um estado sem esse tipo de erro (tempo baixo duração_erro_sintaxe). Em relação às notas finais, percebe-se uma distribuição mais simétrica e com valores centrais em torno de 5 (média e mediana), valor utilizado como limiar para classificar o aluno como aprovado ou reprovado na disciplina de introdução à programação. Por fim, analisando os Coeficientes de Variação (CV) de uma forma geral, percebe-se que, com exceção da duração_erro_sintaxe, existe uma alta variação (CV $>$ 0,5) na duração das transições, o que indica um heterogeneidade nos comportamentos dos alunos ao resolverem os problemas.

Tabela 2. Estatísticas descritivas das variáveis de transição.

	Medida	Submissão incorreta	Erro de sintaxe	Testes	Submissão correta	Nota final
	Média	2,85	0,82	3,18	1,49	5,48
	Mediana	1,04	1,00	2,64	1,60	6,78
	Desv. padrão	3,20	0,25	2,76	0,75	4,70
	Coef. variação (CV)	1,12	0,30	0,86	0,50	0,85
	Coef. assimetria (A)	0,72	–0,92	0,92	–0,58	–0,20
	Mínimo	0,00	0,19	0,00	0,02	0,00
	Máximo	9,42	1,00	11,31	2,60	10,00
Percentis	**25**	0,05	0,56	1,02	0,95	0,00
	50	1,04	1,00	2,64	1,60	6,78
	75	6,16	1,00	4,74	2,17	10,00

5.2. Análise da Correlação entre os Tempos de Transição e o Desempenho Final

Para verificar a correlação entre os tempos de transição apresentados na Figura 1 e o desempenho final dos alunos, optou-se pelos coeficientes de correlação de Pearson (ρ_P) e Spearman[2] (ρ_S). Note que [Hauke and Kossowski 2011] explicam a importância da análise de ambos os coeficientes (Pearson e Spearman) para ter uma maior segurança quanto à significância estatística da relação bivariada. Isso porque o coeficiente de Pearson mensura a relação linear entre duas variáveis contínuas. Logo, se a associação entre elas não for linear, o coeficiente de Spearman é a opção mais adequada, já que ele mensura a relação monotônica entre as variáveis.

Assim, observou-se uma correlação de Pearson moderada (0,663) entre **duração_submissão_incorreta** e a nota final. Por ser um valor positivo, temos que a correlação de dependência linear, ou seja, o tempo que o aluno levou entre a transição de um estado incorreto para um estado correto possui influência estatisticamente significante

[2]Interpretação dos coeficientes: $\rho > |0{,}9|$: correlação muito forte; $|0{,}7| < \rho < |0{,}9|$: correlação forte; $|0{,}5| < \rho < |0{,}7|$: correlação moderada; $|0{,}3| < \rho < |0{,}5|$: correlação fraca; $0 < \rho < |0{,}3|$: correlação desprezível.

na variação do desempenho final. Por serem linearmente dependentes, podemos inferir que ambas as variáveis tendem a crescer em uma mesma taxa. A correlação de Spearman é ainda mais alta (0,791) e também estatisticamente significante, o que ratifica o argumento de que variáveis tendem a crescer juntas. Com isso, supõe-se que a resiliência e o esforço é um fator relevante e moderador do desempenho do aluno. Em outras palavras, alunos que passam mais tempo tentando consertar o código até chegar a uma solução válida tendem a alcançar notas finais mais altas (correlação positiva).

Tabela 3. Correlação de Pearson e Spearman entre os tempos de transição e a Nota Final

		Submissão incorreta	Erro sintaxe	Testes	Submissão correta
Nota Final	Correlação de Spearman	0,791*	–0,491*	0,515*	0,721*
	Sig. (bicaudal)	0,000	0,000	0,000	0,000
	Correlação de Pearson	0,663*	–0,451	0,424*	0,743*
	Sig. (bicaudal)	0,000	0,000	0,000	0,000

* A correlação é significativa no nível 0,01 (bicaudal).

Para a **duração_submissão_correta** e a nota final, o coeficiente também mostrou uma correlação de Pearson (0,743) e Spearman (0,721) forte e estatisticamente significante. Esse resultado sugere que alunos que passam mais tempo resolvendo o problema proposto a fim de alcançar uma solução válida e que seja aceita pelo JO, tendem a alcançar melhores notas finais.

Para a correlação entre **duração_testes** e a nota final, obtivemos uma correlação de Pearson moderada (0,424) e uma correlação de Spearman forte (0,515). Isso significa que ambos podem não ter uma relação linear forte, mas que elas tendem a crescer mononicamente juntas (correlação de Spearman forte). Além disso, ambas as correlações são estatisticamente significantes. Logo, alunos que passam mais tempo testando, e potencialmente debugando o código da sua solução tendem a ter um melhor desempenho.

E por fim, para a **duração_erro_sintaxe** e a nota final, observa-se uma correlação moderada tanto em relação à correlação de Pearson (–0,451) quanto à de Spearman (–0,491). Por ser um valor negativo, indica-se que temos um crescimento inversamente proporcional, ou seja, quanto menor o tempo que o aluno passa em erros de sintaxe, maior será o seu desempenho final e vice-versa.

Com a análise dos resultados obtidos pode-se verificar que, para a população de estudo do presente trabalho, o tempo gasto em erros de sintaxe possui correlação moderada em sentido negativo com o desempenho final do estudante, ou seja, à medida que o tempo gasto em estado de erros de sintaxe cresce, o desempenho final diminui e vice-versa. Ao mesmo passo que o tempo gasto em estados de submissão incorreta e submissão correta possuem relação direta com o desempenho final, bem como o tempo gasto em estados de teste da solução. Logo, podemos partir do pressuposto de que aqueles alunos que passam mais tempo em estados de codificação ou erro buscando solucionar os erros em seu código tendem a ter um desempenho melhor, bem como aqueles que despendem tempo testando repetidas vezes a solução antes de submetê-la para avaliação.

Portanto, com os resultados aqui apresentados podemos explorar o modelo de estados proposto para servir como base para intervenções pedagógicas derivadas do estado de um aluno. Por exemplo, um aluno que aparentemente está preso em estado de submissão

incorreta pode ser orientado a buscar ajuda. Além disso, podemos usar o comportamento no processo de programação para incentivar os alunos a melhorarem suas técnicas de programação. Por exemplo, para alunos que passam longos períodos de tempo presos em estados de erro de sintaxe, uma intervenção seria sugerir aplicar a prática de depuração de código a fim de solucionar problemas semânticos. Além de incentivar os alunos a testarem mais suas soluções antes de submetê-las efetivamente.

Consequentemente, podemos também usar tais dados para a construção de um *dashboard* que pode vir a apresentar de modo contínuo e atualizado informações sobre os estados dos alunos e os progressos em programação. Com essas informações, os professores podem verificar com maior facilidade os alunos em situação de dificuldade, bem como dedicar mais tempo para as aulas de tópicos que o *dashboard* pode apontar como dificultosos para alguns alunos.

6. Considerações Finais e Trabalhos Futuros

A modelagem de estados existentes em ambientes de correção automática de código, como o Juiz Online CodeBench, mostra-se uma boa estratégia para a identificação dos tipos de comportamento que os alunos possuem no processo de codificação de soluções. A compreensão do processo seguido pelos alunos de cursos introdutórios é essencial para a identificação daqueles que estejam tendo dificuldades e para elaboração de medidas de intervenção a fim de evitar baixos desempenhos.

Com o modelo de estados gerado a partir da análise dos dados, pôde-se verificar que o tempo entre as transições de submissões incorretas até a obtenção de submissões corretas influencia, ainda que de forma moderada, o desempenho final que o aluno terá ao final do período. Essa descoberta possibilita a aplicação do modelo preditivo para identificar com antecedência o desempenho final dos alunos e aqueles que necessitam de intervenção do professor.

Dessa forma, caberá a trabalhos futuros a execução de novos experimentos com uma amostra maior, englobando as turmas do segundo semestre de 2019 e também de anos anteriores disponíveis no dataset de estudo, a fim de verificar se as correlações se mantém ou não. Visando assim, propor um modelo preditivo baseado no modelo de estados deste estudo a fim de inferir precocemente o desempenho final dos alunos em turmas de IPC que fazem uso de um Juiz Online.

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

Alves, A., de Carvalho, L. S. G., Oliveira, E., and Fernandes, D. (2019). Análise comportamental em juízes online para predição do desempenho final de alunos em disciplinas de computação. In *Simpósio Brasileiro de Informática na Educação (SBIE)*, volume 30, page 1906.

Baker, R. S. and Inventado, P. S. (2014). Educational data mining and learning analytics. In *Learning analytics*, pages 61–75. Springer.

Carter, A. S., Hundhausen, C. D., and Adesope, O. (2015). The normalized programming state model: Predicting student performance in computing courses based on programming behavior. In *Proceedings of the eleventh annual International Conference on International Computing Education Research*, pages 141–150. ACM.

Dwan, F., Oliveira, E., and 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., Oliveira, E., Pereira, F., Fernandes, D., and de Carvalho, L. S. G. (2019). Adaptação de um método preditivo para inferir o desempenho de alunos de programação. In *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)*, volume 30, page 1651.

Hauke, J. and Kossowski, T. (2011). Comparison of values of pearson's and spearman's correlation coefficients on the same sets of data. *Quaestiones geographicae*, 30(2):87–93.

Jadud, M. C. (2006). Methods and tools for exploring novice compilation behaviour. In *Proceedings of the second international workshop on Computing education research*, pages 73–84. ACM.

Pereira, F., Oliveira, E., Fernandes, D., de Carvalho, L. S. G., and Junior, H. (2019). Otimização e automação da predição precoce do desempenho de alunos que utilizam juízes online: uma abordagem com algoritmo genético. In *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)*, volume 30, page 1451.

Pereira, F. D., Oliveira, E. H., Oliveira, D. B., Cristea, A. I., Carvalho, L. S., Fonseca, S. C., Toda, A., and Isotani, S. (2020). Using learning analytics in the amazonas: understanding students' behaviour in introductory programming. *British Journal of Educational Technology*.

Pereira, F. D., Oliveira, E. H. T., and 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.

Robins, A. V. (2019). Novice programmers and introductory programming. In *The Cambridge Handbook of Computing Education Research*, chapter 12, pages 327–376. Cambridge University Pres, Cambridge.

Tabanao, E. S., Rodrigo, M. M. T., and Jadud, M. C. (2011). Predicting at-risk novice java programmers through the analysis of online protocols. In *Proceedings of the seventh international workshop on Computing education research*, pages 85–92. ACM.

Wasik, S., Antczak, M., Badura, J., Laskowski, A., and Sternal, T. (2018). A survey on online judge systems and their applications. *ACM Computing Surveys (CSUR)*, 51(1):1–34.

Watson, C. and Li, F. W. (2014). Failure rates in introductory programming revisited. In *Proceedings of the 2014 conference on Innovation & technology in computer science education*, pages 39–44.