Um Ambiente para Ensino de Programação com *Feedback* Automático de Exercícios

Mireille Pinheiro Moreira, Eloi Luiz Favero

Programa de Pós-Graduação em Ciência da Computação (PPGCC) – Centro de Ciências Exatas e Naturais (CCEN) – Universidade Federal do Pará (UFPA)
Rua Augusto Corrêa, No. 1 – 66075-110 – Belém – PA – Brasil

{mireille, favero}@ufpa.br

Abstract. *The teaching-and-learning process of algorithms and computer programming still has a large number of difficulties. Learning Environments (LE) have been proposed to solve that challenge. One proposal is to use the LE's to solve the problem of following each student's step in the process of constructing algorithms. This paper presents a study about techniques of algorithms auto-marking that allows immediately feedback to the student, helping the teacher in correction and grading of programming exercises, in the context of a virtual learning environment or in laboratory classes.*

Resumo. *O processo de ensino-aprendizagem de algoritmos e programação de computadores ainda apresenta um grande desafio. Ambientes virtuais (AV) de ensino têm sido propostos com algumas soluções para este desafio. Uma proposta é utilizar os AV para solucionar o problema do acompanhamento e correção de cada solução dos estudantes no processo de resolução de problemas de algoritmos. Este artigo apresenta um estudo sobre técnicas de avaliação automática de algoritmos que possibilitam um retorno imediato ao estudante auxiliando o professor na correção e avaliação de exercícios de programação, no contexto de um ambiente virtual ou em aulas práticas de laboratório.*

1. Introdução

A programação é uma das disciplinas essenciais para estudantes de computação. Constitui a base para muitos campos em que a informática se aplica, assim como capacita o indivíduo a utilizar a lógica de programação na resolução de problemas, fator muito importante em disciplinas avançadas. Dificuldades no aprendizado de programação refletem em altos índices de reprovação e conseqüentemente em mau desempenho do aluno em outras matérias que têm programação como base [Tobar *et al* 2001]. A dificuldade de aprender programação pode ser conseqüência de vários fatores, tais como: base matemática fraca, dificuldade na compreensão do problema e no entendimento do assunto [Borges 2000] [Giraffa *et al* 2003].

Segundo [Prior 2003], a habilidade de programação de computadores não pode ser adquirida sem um significante esforço em atividades práticas de laboratório. Neste contexto, muitas ferramentas foram propostas para auxiliar o educador no ensino de programação na prática. Mas mesmo com o advento de vários sistemas de ensino de programação algumas barreiras ainda são encontradas pelo educador no processo de

ensino de programação, como a dificuldade de avaliar todos os exercícios de todos os alunos em curto prazo. Em geral, os sistemas provêem um ambiente que permite ao aluno fazer seus algoritmos, porém, para o professor torna-se difícil fornecer um rápido *feedback* e disponibilizar as correções para os alunos. Devido ao sucesso obtido em pesquisas sobre o uso de avaliação automática [Lino *et al* 2007] [Saikkonen *et al* 2001] [Kay *et al* 1994], em que o próprio sistema é um dos atores do processo, este trabalho propõe a implementação de um ambiente de aprendizagem utilizando avaliação automática para ensino de programação. Os principais benefícios para o educador são entre outros [Kjollerstrom & Martensson 1999]:

- Menor esforço, uma vez que conta com o auxílio da ferramenta;
- Melhor administração dos estudantes e de suas tarefas;
- Melhor rastreamento individual dos estudantes;
- Melhor qualidade de ensino, devido o maior tempo de prática;
- Mais tempo para contato com os estudantes.

A maioria dos sistemas de ensino-aprendizagem de programação que aplicam avaliação automática utiliza avaliação baseada em testes, como em [Douce *et al* 2005] e [Higgins *et al* 2005]. O teste consiste em separar um conjunto de entradas para um algoritmo e fazer a correspondência com um determinado conjunto de saídas. Então, são executados os testes que verificam se para dada uma entrada a sua saída esperada é obtida. Com base nisso, o sistema é capaz de avaliar e atribuir notas para os exercícios.

Para complementar a abordagem de testes, este trabalho propõe a utilização de dois modelos de regressão linear: um baseado em indicadores compostos por métricas de engenharia de software e o outro com indicadores de similaridade estrutural baseado em n-gramas. O objetivo destes modelos é medir quanto uma solução de um estudante se aproxima em termos de complexidade da resposta-modelo (supostamente ideal; podendo ser uma coleção com várias respostas). Isto orienta o aluno, pois ele pode incrementalmente refinar a sua solução reduzindo a complexidade e aproximando-se da resposta-modelo, num processo iterativo/interativo. Esta solução resolve o problema dos avaliadores automáticos baseados em testes: quando uma solução retorna o mesmo resultado da resposta-modelo o estudante abandona a solução e vai para o próximo problema.

Os modelos de regressão linear foram montados a partir de um conjunto de 20 exercícios coletados com respostas de alunos (5 para cada exercício) e avaliados por professores. A partir deste conjunto criaram-se dois modelos que obtiveram precisão de 90,42% e 95,22%. Estes modelos foram implementados no ambiente virtual de ensino Moodle e pode ser utilizado tanto na educação à distância como em aulas de laboratório com a presença do professor e monitores.

2. Aprendizado de Programação

As disciplinas relacionadas a programação formam um núcleo essencial dos cursos de computação e a aprendizagem de algoritmos é importante para a maioria das carreiras de informática. Entretanto, a dificuldade dos estudantes para aprender programação é notória. [Gomes 2000] fala do insucesso verificado na aprendizagem de programação em

geral. Isto se justifica por diversos motivos, como por exemplo [Borges 2000] [Giraffa *et al* 2003]:

- Hábitos de estudo pouco disciplinados e centrados em memorização;
- Conhecimentos prévios desestruturados, principalmente nos domínios matemáticos e lógicos;
- Abordagens pouco motivadoras;
- Conteúdos pouco relacionados ao cotidiano dos sujeitos;
- Dificuldades na compreensão do enunciado dos problemas;
- Forte carga de conceitos abstratos, entre outros.

Por conta destes problemas, [Borges 2000] afirma que o índice de alunos que completam seu curso sem ter um conhecimento mínimo adequado na área de programação é alto. Ou ainda, muitos nem sequer completam o curso, abandonando-o logo nos primeiros semestres devido às dificuldades. Segundo [Tobar *et al* 2001], outro fator agravante para isto ocorrer é a dificuldade encontrada pelos professores para acompanharem efetivamente as atividades laboratoriais de programação, dado o grande número de estudantes sob sua supervisão. Além disso, sem o acompanhamento do professor ocorrem situações em que o aluno uma vez que consegue resolver um determinado problema de programação não se interessa em verificar se é a solução mais simples ou adequada, satisfazendo-se com os resultados iniciais obtidos. Muitas vezes o reduzido tempo para atividades de laboratório também contribui para que isto ocorra.

3. Avaliação Automática

Usualmente, a avaliação automática é utilizada estritamente na correção de questões objetivas como de múltipla escolha. Questões discursivas dão espaço para uma grande quantidade de respostas possíveis, sendo computacionalmente impossível prevê-las na totalidade. Uma palavra a mais, ou a menos, na resposta do aluno, pode ser considerada errada em um universo fechado de respostas corretas, mesmo que o raciocínio do estudante esteja correto. Um dos caminhos para avaliar questões discursivas é utilizar um mecanismo capaz de interpretar o conteúdo da resposta avaliando-o dentro de um conjunto de parâmetros específicos, como dos modelos propostos neste trabalho que fazem uso de indicadores de (dis)similaridade.

O tema avaliação automática é assunto de debate na comunidade de educação [Hearst 2000] [Kay *et al* 1994]. Segundo [Lino *et al* 2007], há duas dificuldades relevantes encontradas pelos educadores, no processo ensino-aprendizagem de programação em laboratórios: primeiro, como avaliar manualmente todos os exercícios dos estudantes, para turmas grandes? Segundo, como fornecer um *feedback* em curto prazo de forma que o estudante possa aperfeiçoar sua solução?

Em linguagens de programação é comum haver mais de uma solução para um determinado problema. Torna-se difícil para o educador avaliar individualmente todas as soluções de um determinado exercício e pontuar de acordo com a quantidade de acertos ou erros dos estudantes. Geralmente, exercícios de programação são avaliados levando-se em consideração duas métricas: 1) se o algoritmo retorna o resultado esperado; e 2) a complexidade da solução, isto é, se a solução está bem escrita.

Algumas soluções computacionais foram desenvolvidas a fim de avaliar estes pontos. Alguns sistemas de ensino-aprendizagem de programação, como em [Almeida *et al* 2002] e [Silva 2006] incluem um interpretador de pseudo-código que facilita o aluno a programar. [Nunes *et al* 2004] desenvolveram em seu trabalho uma aplicação para testar algoritmos e classes Java, verificando as entradas/saídas do código. Estes sistemas resolvem computacionalmente a primeira medida de correção de exercícios que é a verificação do resultado. Porém, na prática, espera-se que os alunos sejam avaliados levando-se em consideração também aspectos da qualidade da solução. [Lino *et al* 2007] propõem um ambiente para a linguagem SQL em as medidas são avaliadas pelo próprio ambiente auxiliando o professor no processo de correção de exercícios.

Algumas técnicas que podem ser utilizadas para este tipo de avaliação são a regressão linear múltipla e os n-gramas. A regressão linear múltipla como técnica de predição vem sendo aplicada em avaliação automática com sucesso [Hearst 2000] [Lino *et al* 2007]. Os modelos podem predizer um conceito numérico a partir de diversos indicadores que medem diferentes aspectos dos objetos sendo avaliados. Por exemplo, para medir algoritmos utilizam-se métricas da engenharia de software; outra abordagem é medir através de similaridade com n-gramas.

3.1. Métricas de Engenharia de Software

[Almeida 2005] enumera várias métricas de engenharia de software usadas para se avaliar a complexidade de um código-fonte. Aqui utilizamos as seguintes: quantidade de uso de funções da linguagem, número de palavras reservadas, número de declarações, número de linhas do programa, complexidade de McCabe e volume de Halstead.

3.2. N-Gramas

Um n-grama consiste em um pedaço de n-caracteres, extraído de uma cadeia de caracteres maior (uma palavra ou frase). Usualmente, n assume o valor 1, 2 ou 3 (unigrama, bigrama e trigrama, respectivamente). A técnica dos n-gramas combinada com medidas de similaridade vetorial (tipo cosseno) permite comparar objetos tais como textos livres. Quebrando-se duas cadeias de caracteres em n-gramas, pode-se medir quantos n-gramas elas compartilham. Por exemplo, se duas cadeias compartilham 90% de trigramas uma é quase igual à outra; se forem textos podemos dizer que houve plágio. Assim pode-se determinar quanto dois textos são similares [Sukkarieh *et al* 2003]. No caso de programação, dois estudantes podem fazer um mesmo programa com nomes de variáveis e classes diferentes. Neste caso para medir a similaridade dos códigos iguala-se os nomes de variáveis e classes antes de aplicar n-gramas.

Com n-gramas mede-se o quanto uma solução do estudante compartilha com a resposta-modelo. N-gramas podem também ser aplicados em código que ainda não é executável indicando de certo modo se o estudante está ou não se aproximando da(s) resposta(s)-modelo(s).

4. Estudo de Caso

Neste estudo de caso, utilizamos duas abordagens de predição da nota do aluno: a primeira, centrada na complexidade do programa, utilizando a técnica regressão linear múltipla com métricas de engenharia de software como indicadores; e a segunda,

centrada em medida de similaridade estrutural, utilizando a técnica regressão linear múltipla com n-gramas como indicadores. Estes dois experimentos foram executados com a mesma base de teste/treinamento.

4.1. Base de testes/treinamento

O conjunto de teste/treinamento é composto de respostas de exercícios de programação. Utilizou-se a sintaxe da linguagem Java, porém os programas são puramente procedurais; ou seja, não foram utilizados recursos de orientação a objetos da linguagem Java (excetuando-se o uso de métodos embutidos da linguagem). Estes exercícios são aplicados em um sistema de ensino de Java básico.

Foram selecionados para o estudo de caso 20 exercícios de programação para compor o corpus de treinamento, envolvendo o uso básico dos principais elementos de programação. A escolha dos exercícios (e suas respostas) para o estudo de caso levou em consideração: a presença de estruturas de controle, estruturas de repetição, declarações, uso de funções da linguagem e respostas com um tamanho razoável. Para cada atividade foram coletadas 5 respostas (sendo uma a resposta-modelo, que vale a nota máxima, 10 pontos), totalizando 100 respostas em formato de código-fonte. Foram selecionadas apenas respostas que retornam o resultado esperado da questão, baseando-se em um conjunto de entradas/saídas. As respostas foram avaliadas e atribuídas notas por 2 professores, sendo a nota final a média aritmética entre estas 2 notas (Figura 1). Em caso de discrepância entre as 2 notas, as respostas foram revisadas novamente até que os professores entrassem em acordo. Este conjunto de exercícios já avaliados compõe a base de testes/treinamento dos dois modelos de regressão.

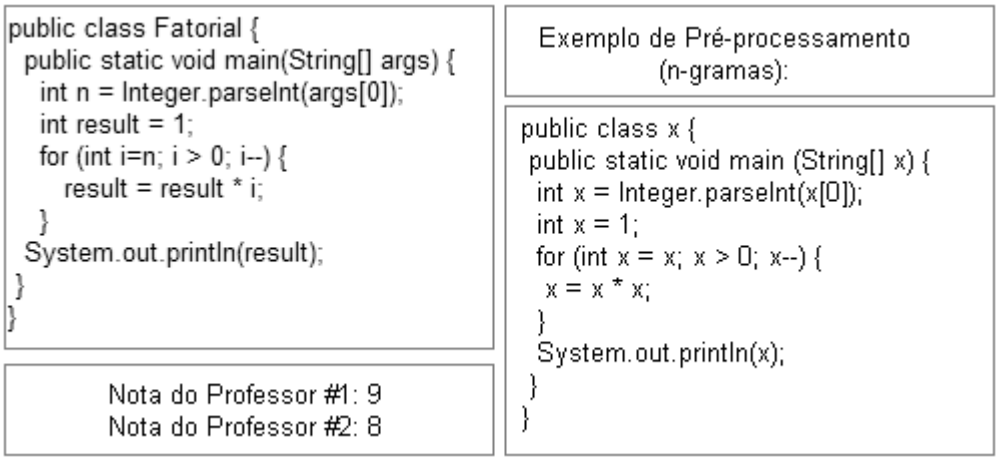

Figura 1. Exemplo de notas atribuídas (dir.) e exemplo de resposta após pré-processamento (esq.)

Para a regressão com n-gramas, as respostas foram pré-processadas (Figura 1), igualando-se nomes de variáveis e classes que podem variar de programa para programa.

4.2. Regressão Linear Múltipla utilizando Métricas de Engenharia de Software

A primeira parte do estudo de caso consiste em aplicar a técnica da regressão linear múltipla, juntamente com as métricas de engenharia de software. São extraídos, de forma

automática para cada resposta, os dados referentes às métricas de engenharia de software apresentadas anteriormente.

Na Tabela 1 mostramos a coleta das métricas de engenharia de software para alguns códigos-resposta. Por exemplo, a resposta #1 tem nota 9,25, utilizou 3 funções embutidas da linguagem, 10 palavras reservadas, tem complexidade de McCabe de valor 2, 2 declarações, volume de Halstead de 168,56 e 14 linhas. Assim, os dados estão prontos para serem aplicados no algoritmo da regressão múltipla e iniciar o treinamento.

Tabela 1. Amostra de extração de métricas

	Nota	Funções	Palavras Reservadas	McCabe	Declarações	Halstead	Linhas
#1	9,25	3	10	2	2	168,56	14
#2	8,25	3	10	2	3	190,16	12
#3	8,75	4	12	3	2	257,84	17
#4	8,5	3	14	3	4	294,03	20
#5	10	2	8	1	2	146,95	11
#6	9,25	4	13	3	3	317,29	16
#7	8,25	7	16	4	6	445,88	17

Dado um conjunto de informações iniciais, a regressão calcula uma variável dependente (Y), que é a variável na qual se deseja fazer a predição e um conjunto de variáveis independentes (X1, X2, ...), os indicadores, que influenciam na variável dependente. O algoritmo da regressão linear múltipla cria uma função matemática que seja a mais adequada para satisfazer este conjunto de dados. De posse desta função, o valor da variável pode ser predito.

$$Y = a + (b1 * X1) + (b2 * X2) + ... + (bp * Xp)$$

Figura 2. Fórmula geral da regressão linear múltipla

No estudo de caso, a variável dependente é a nota, enquanto que as variáveis independentes, ou indicadores, neste caso, são as métricas de engenharia de software. A função que aproxima estes indicadores da variável dependente foi obtida, onde *nota, volume*, *linhas*, *declaracoes*, *mccabe*, *palreservadas* e *funcoes* são as variáveis:

$$nota = 10{,}08153 + (-0{,}00059 * volume) + (-0{,}046505 * linhas) + (-0{,}026655 * declaracoes) + (0{,}063951 * mccabe) + (-0{,}021961 * palreservadas) + (-0{,}033431 * funcoes)$$

Tabela 2. Precisão da Regressão com Métricas de Engenharia de Software

Questão	**#1**	**#2**	**#3**	**#4**	**#5**	**#6**	**#7**	**#8**	**#9**	**#10**
Média de acerto (%)	90,86	91,90	88,46	72,27	95,07	88,17	92,71	94,00	94,40	92,69
Questão	**#11**	**#12**	**#13**	**#14**	**#15**	**#16**	**#17**	**#18**	**#19**	**#20**
Média de acerto (%)	89,93	93,26	91,53	93,04	92,22	88,46	90,75	86,48	93,13	89,06
Média Geral	**90,42%**									

A partir da fórmula obtida, foram substituídas as variáveis independentes pelos seus valores assumidos em cada exercício de programação. Comparando as notas previstas e as notas reais, o algoritmo da regressão múltipla utilizando as métricas de engenharia de software obteve precisão de 90,42% conforme demonstrado na Tabela 2.

4.3. Regressão Linear Múltipla utilizando N-Gramas

Nesta segunda parte do estudo de caso, analisamos a viabilidade do uso da técnica de n-gramas para comparar as respostas dos alunos com a resposta-modelo, textualmente, utilizando estes graus de similaridade como indicadores da regressão linear múltipla.

Comparam-se as respostas com a resposta-modelo, utilizando unigramas, bigramas e trigramas. A fórmula da regressão obtida foi a seguinte:

$nota = 6{,}90379 + (3{,}92293 * unigramas) + (-0{,}14665 * bigramas) + (0{,}78599 * trigramas)$

Tabela 3. Precisão da Regressão com N-Gramas

Questão	**#1**	**#2**	**#3**	**#4**	**#5**	**#6**	**#7**	**#8**	**#9**	**#10**
Média de acerto (%)	96,08	95,18	96,22	93,18	96,18	93,75	97,12	98,28	96,35	94,30
Questão	**#11**	**#12**	**#13**	**#14**	**#15**	**#16**	**#17**	**#18**	**#19**	**#20**
Média de acerto (%)	94,50	95,67	97,46	93,25	96,96	94,02	97,81	91,29	94,14	92,75
Média Geral	**95,22%**									

As médias de acertos desta técnica são demonstradas na Tabela 3, sendo a média geral de 95,22%.

5. Resultados

Comparando-se as duas formas de avaliação automática de programação propostas, a regressão linear múltipla com os indicadores de n-gramas obteve um grau de acerto um pouco maior do que com as métricas de engenharia de software: 95,22% e 90,42% respectivamente. Isto quer dizer que ambas as medidas podem ser consideradas boas na avaliação automática de programação, acertando em boa parte das notas.

Devido a regressão com as métricas considerar maior ou menor complexidade do programa baseado somente nas quantidades das métricas – basicamente, quanto menor a complexidade, maior a nota – programas com pequena complexidade ganharão uma nota alta, mesmo que a solução esteja incorreta. Portanto, tem seu grau de acerto comprometido. Para contornar este problema, sugerimos a utilização deste modelo combinado com a avaliação de testes para verificar a execução do programa também.

A utilização da regressão com n-gramas obteve um alto grau de acerto. Considerando que na etapa de pré-processamento das respostas substituímos os nomes de variáveis e classes, a comparação é considerada estrutural. Por um lado os unigramas não consideram a ordem das palavras, por outro lado os bigramas e trigramas levam em conta essa ordem, inclusive trigramas são usados como medida de plágio.

6. Utilização no Sistema Moodle

Em uma primeira versão do sistema que propomos utilizando avaliação com *feedback* automático, usamos a técnica de avaliação automática já existente, baseada em testes, juntamente com a aplicação das regressões.

O Moodle é um sistema de gerenciamento de cursos para aprendizagem online, feito em PHP. Possui diversos recursos, tais como ferramentas administrativas, fóruns, chats, lições. Por causa de seu uso crescente e popular, além de ser de uso genérico, foi escolhido para este estudo de caso. No módulo de Tarefas, desenvolvemos uma nova categoria específica para contemplar o proposto neste trabalho: a submissão de exercícios em Java com avaliação automática *online*. Cada Tarefa representa um exercício de programação.

Devido à necessidade de se compilar e executar os exercícios submetidos pelos alunos, em Java, foi desenvolvido um módulo que se comunica com o Moodle através de web service. O lado servidor é um sistema Java, que aguarda uma solicitação do lado cliente, PHP. No momento em que um aluno submete uma resposta, o PHP invoca uma chamada ao Java. No servidor, a resposta do aluno passa por diversos processos, conforme demonstrado na Figura 3.

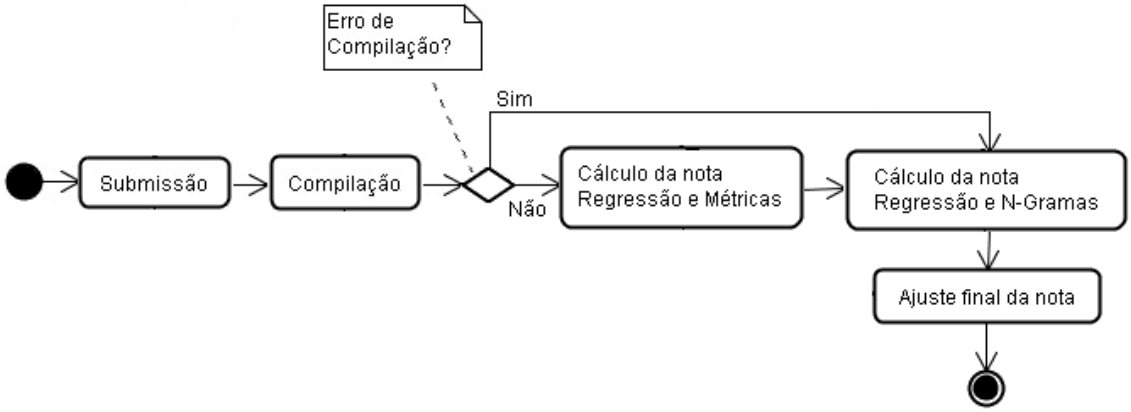

Figura 3. Fluxograma do Sistema

Na Figura 3, temos o fluxo geral do sistema. Após a submissão do exercício, este é compilado e caso não haja erros calcula-se a nota da regressão por métricas de engenharia de software e por n-gramas, finalizando com ajuste final de cada uma das notas. Caso haja erros de compilação, calcula-se somente a regressão por n-gramas que indica se o estudante está trabalhando na direção certa. No ajuste final da nota, utilizamos testes para verificar se o algoritmo está retornando os resultados esperados.

Após passar por estas fases, o sistema Java retorna ao PHP as notas finais do aluno, que são exibidas na tela. Caso haja erros de compilação ou saídas de execução, o sistema apresenta-os na tela também. Mesmo com o resultado correto o aluno pode não ter nota máxima, assim ele pode refazer a resposta e submeter novamente. Pode iterativamente fazer este passo até atingir a nota máxima.

7. Conclusões e Trabalhos Futuros

A utilização de ambientes de ensino-aprendizagem tem sido uma prática comum em universidades e de fato auxiliam no cotidiano de uma disciplina. Porém, alguns problemas permanecem, especialmente no que se refere ao ensino de programação. Uma vez que o aprendizado de programação é muito importante na carreira do estudante de computação é essencial que este consiga adquirir os fundamentos básicos, sendo capaz de avaliar, pensar logicamente e desenvolver seus próprios algoritmos. Como se trata de uma disciplina em que a prática em laboratório realiza papel fundamental no desenvolvimento destas habilidades é desejável que o professor esteja atento à evolução deste conhecimento. Contudo, nem sempre isto é possível: turmas de programação são geralmente compostas por muitos alunos e as aulas de programação têm tempo reduzido. Este artigo mostra uma abordagem para minimizar este problema com a utilização de técnicas de avaliação automática de exercícios de programação.

As métricas de engenharia de software podem ser uma boa medida de complexidade para avaliar um algoritmo. A análise textual das respostas, feita com n-gramas, também é uma forma de avaliar as questões. A técnica de regressão linear complementa-os oferecendo métricas de cálculo a partir de um conjunto de entradas previamente avaliadas por professores. O uso combinado das métricas de engenharia de software e do n-gramas na regressão linear múltipla foi bem-sucedido para avaliação de algoritmos, atribuindo, com certa precisão, as notas para os algoritmos na sintaxe Java.

Como trabalhos futuros, estamos em duas frentes: por um lado estamos viabilizando a utilização do avaliador dentro do ambiente Moodle, integrando a funcionalidade de avaliação automática também no módulo de Questionário do Moodle, para permitir sortear vários exercícios para compor uma lista ou prova. Hoje o aluno já pode submeter várias vezes, num processo de refinamento, os algoritmos que são instantaneamente avaliados, no módulo de Tarefas. Também, identificar situações de plágio: por exemplo, nas primeiras submissões se a sua resposta é idêntica de outro estudante. Além disso, criar ferramentas para o professor visualizar o andamento da turma, visando facilitar o acompanhamento. Por outro lado também estamos buscando aperfeiçoar o avaliador para trabalhar com as respostas que não compilam e/ou não geram os resultados esperados dando um *feedback* significativo para o estudante. Além disso, noutra frente estamos desenvolvendo um simulador para trabalhar passo a passo na execução do código, complementando este processo de avaliação [Mota *et al* 2008].

Referências

Almeida, E.; Costa, E.; Silva, K.; Paes, R.; Almeida, A.; Braga, J. "AMBAP: Um Ambiente de Apoio Ao Aprendizado de Programação". Workshop de Educação em Computação, Congresso anual da SBC, Florianópolis. 2002.

Almeida, Vinícius Cristiano de. "Proposta de Implementação de Métricas de Complexidade em um Avaliador Automatizado de Programas – VDSP". Monografia de diplomação do curso de Sistemas de Informação da PUC-MG. Arcos, 2005.

Borges, Marcos Augusto F. "Avaliação de uma Metodologia Alternativa para a Aprendizagem de Programação". Workshop de Educação em Computação, Congresso anual da SBC, Curitiba, PR. 2000.

Douce, Christopher; Livingstone, David; Orwell, James. "Automatic Test-Based Assessment of Programming: A Review" em: Journal on Educational Resources in Computing (JERIC). Vol 5, Nº 3, Art. Nº 4. Setembro 2005.

Giraffa, Lucia; Marczak, Sabrina; Almeida, Gláucio. "O ensino de algoritmos e programação mediado por um ambiente na Web". Congresso Nacional da Sociedade Brasileira de Computação. Campinas, SP. 2003.

Gomes, A. J. "Ambiente de suporte à aprendizagem de conceitos básicos de programação". Dissertação de Mestrado. Universidade de Coimbra, 2000.

Hearst M.A. "The Debate on Automated Essay Grading", IEEE Intelligent Systems, vol. 15, nº. 5, p. 22-37. 2000.

Higgins, Colin A. et al. "Automated Assessment and Experiences of Teaching Programming". Journal on Educational Resources in Computing. vol 5, nº 3, art. nº 5. Setembro 2005.

Kay, D. G.; Scott, T.; Isaacson, P.; Reek, K. A. "Automated Grading Assistance for Student Programs". ACM SIGCSE Bulletin, vol 26, is. 1, p. 381-382. 1994.

Kjollerstrom, B.; Martensson, M. "Assessment: The Key to Changing the Way We Learn". CAL-Laborate, vol 3, 17-20. Outubro 1999.

Lino, A.D.P; Harb, M.P.A.; Brito, S.R.; Silva, A.S.; Favero, E. "Avaliação automática de consultas SQL em ambiente virtual de ensino-aprendizagem". 2ª Conferência Ibérica de Sistemas e Tecnologias de Informação. Porto, Portugal, 2007.

Mota, Marcelle Pereira; Pereira, Lis W. Kanashiro; Favero, Eloi Luiz. "JavaTool: Uma Ferramenta para Ensino de Programação". Workshop de Educação em Computação, Congresso anual da SBC, Belém, PA. 2008.

Nunes, Ingrid de Oliveira; Lisbôa, Maria Lúcia Blanck. "Testador Automático e Método de Avaliação de Programas em Java". XVI Salão de Iniciação Científica e XIII Feira de Iniciação Científica. Porto Alegre, RS. 2004.

Prior, J. C. "Online assessment of SQL query formulation skills". In Proceedings of the Fifth Australasian Conference on Computing Education. Adelaide, Australia. 2003.

Saikkonen, Riku; Malmi, Lauri; Korhonen, Ari. "Fully Automatic Assessment of Programming Exercises". Annual Joint Conference Integrating Technology into Computer Science Education. Canterbury, Reino Unido. 2001.

Silva, Carlos da. "Ambiente de Educação a Distância Sobre a Linguagem de Programação Java". Taller Internacional de Software Educativo. Santiago, Chile. 2006.

Sukkarieh, Jana Z.; Pulman, Stephen G.; Raikes, Nicholas. "Auto-Marking - Using Computational Linguistics to Score Short, Free Text Responses". International Alliance for Learning Conference. 2003.

Tobar, C. M. et al. "Uma Arquitetura de Ambiente Colaborativo para o Aprendizado de Programação". XII Simpósio Brasileiro de Informática na Educação, Vitória, ES. 2001.