Sistema de Apoio à Prática Assistida de Programação por Execução em Massa e Análise de Programas

Márcia G. de Oliveira [1], **Matheus de Araújo Nogueira** [2], **Elias Oliveira** [1]

[1] Programa de Pós-Graduação em Informática (PPGI)
Universidade Federal do Espírito Santo
Vitória – ES – Brasil

[2]Faculdades Integradas Espírito Santenses - FAESA
Vitória – ES – Brasil

clickmarcia@gmail.com, elias@acm.org

Abstract. *Assisting the practice of exercises in a course of programming, especially in large classes, takes time and effort of teachers. In order to assist teachers in programming exercises assessment, we developed PCodigo, an integrated system to Moodle that provides tools to run and analyze programs in C Language. The PCodigo's contributions to support the teachers' work and to encourage the programming learning are daily running a lot of programs in a flexible way and providing analysis resources to recognize classes of solutions, to identify divergent solutions and to detect evidence of plagiarism.*

Keywords*: Analysis of Programs , Massive Running, PCodigo*

Resumo. *Assistir a prática de exercícios em um curso de programação, especialmente em turmas numerosas, demanda tempo e esforço de professores. Com o objetivo de auxiliar o professor na avaliação de exercícios de programação, desenvolvemos o PCodigo, um sistema integrado ao Moodle que oferece recursos para executar e analisar programas em Linguagem C. As contribuições do PCodigo para apoiar o trabalho docente e favorecer a aprendizagem de programação são as seguintes: executar programas em massa diariamente de forma flexível e oferecer recursos de análise de programas como o reconhecimento de classes de soluções, a identificação de soluções divergentes e a detecção de indícios de plágios.*

Palavras-chave*: Análise de Programas, Execução em massa, PCodigo*

1. Introdução

A prática da programação de computadores como parte de um processo de aprendizagem tem sido considerada complexa tanto para alunos quanto para professores. Para os alunos, a programação é um conhecimento de difícil aprendizagem porque o desenvolvimento de programas é um processo que envolve a combinação de várias habilidades cognitivas.

O professor, por sua vez, para assistir e avaliar individualmente seus alunos na prática da programação, dispende muito tempo e esforço, principalmente em turmas numerosas. Dessa forma, não conseguindo fornecer *feedbacks* imediatos para muitos alunos,

a tendência é que professores reduzam cada vez mais o número de exercícios aplicados. No entanto, isso pode afetar a aprendizagem dos alunos, uma vez que a programação, para ser bem aprendida, requer extensa prática de exercícios [Ihantola et al. 2010].

Atentando para a complexidade da aprendizagem de programação, nos últimos anos, muitas tecnologias têm sido desenvolvidas para apoiar o ensino, a aprendizagem e a avaliação de programação. Hoje temos tecnologias para executar programas *online* [Campos and Ferreira 2004], recomendar exercícios personalizados [De Oliveira et al. 2013] e até para avaliação automática de exercícios de programação [Moreira and Favero 2009, De Souza et al. 2011]. No entanto, ainda carecemos de tecnologias que de fato favoreçam a prática assistida de programação atendendo às demandas de avaliação em massa de exercícios e de *feedback* imediato.

Com o objetivo de auxiliar o professor na avaliação de exercícios, desenvolvemos o *PCodigo*, um sistema integrado ao *Moodle* com recursos de execução em massa e de análise de programas. A execução em massa do *PCodigo* é aplicável a diferentes linguagens de programação. Já os recursos de análise de códigos foram desenvolvidos para a Linguagem C, embora possam ser facilmente adaptados para outras linguagens.

As contribuições do *PCodigo* para apoiar o trabalho docente e favorecer a aprendizagem de programação são as seguintes: executar programas em massa diariamente de forma flexível e oferecer recursos para análise de programas como a identificação de classes de soluções, de soluções divergentes e de indícios de plágios.

Este trabalho está organizado conforme a ordem a seguir. Na Seção 2, apresentamos os trabalhos relacionados. Na Seção 3, explicamos a arquitetura do *PCodigo* e as tecnologias que o compõem. Na Seção 4, apresentamos como o *PCodigo* está sendo aplicado para execução em massa e análise de programas em Linguagem C. Na Seção 5, concluimos com as considerações finais e trabalhos futuros.

2. Trabalhos Relacionados

Muitos sistemas de apoio à prática da programação foram desenvolvidos com as finalidades de submissão, execução e avaliação de exercícios. Entre esses sistemas destacamos o sistema de submissão de exercícios BOCA [Campos and Ferreira 2004, França et al. 2011], o sistema *Pdetect* para análise e clusterização de programas suspeitos de plágios e um sistema de mapeamento de programas em vetores [Oliveira et al. 2014].

O BOCA é um sistema de internet para submissão de exercícios com autenticação, controle de tempo e disponibilização de resultados em tempo real [Campos and Ferreira 2004]. O BOCA tem sido aplicado no Brasil como apoio às competições de programação promovidas pela Sociedade Brasileira de Computação (SBC).

O processo de correção automática do BOCA consiste dos seguintes passos [Campos and Ferreira 2004, França et al. 2011]: receber o programa-fonte, compilá-lo, gerar o programa executável, testar as entradas fornecidas, comparar a saída gerada com a saída do gabarito, avaliar e enviar um relatório de submissão.

Uma adaptação do sistema BOCA é o BOCA-LAB que oferece envio múltiplo de arquivos para compilação, a integração ao *Moodle* e uma infraestrutura para prover o balanceamento de carga entre diversos servidores [França et al. 2011].

O *PDetect* é um sistema que faz análise e clusterização de plágios em exercícios de programação [Moussiades and Vakali 2005]. Para isso, recebe códigos-fontes, mapeia-os em um conjunto de *tokens* de uma linguagem de programação, identifica os pares suspeitos de plágio e os reúne em *clusters*, formados a partir de um parâmetro de corte do índice de similaridade [Moussiades and Vakali 2005].

O Núcleo de Avaliação Diagnóstica (NAD) de [Oliveira and Oliveira 2014] recebe programas e os transforma em vetores cujas dimensões são chamadas de componentes de habilidades (C_i). Cada componente de habilidade representa a frequência de ocorrência de palavras reservadas, símbolos, operadores, funções e indicadores de funcionamento.

Estendendo as ideias desses sistemas, desenvolvemos o *PCodigo*, que tem como principais vantagens executar programas em massa de forma mais flexível em relação ao formato de submissão e de oferecer mecanismos que permitem o professor extrair e analisar, de forma mais holística, informações da prática de programação dos seus alunos.

3. O Sistema PCodigo

O *PCodigo* é um sistema de apoio à prática assistida de programação que, integrado ao *Moodle*, recebe soluções de atividades de programação submetidas por alunos, executa-as e emite relatórios de avaliação para professores.

A arquitura do *PCodigo*, ilustrada na Figura 1, é formada por uma interface *web* do *Moodle*, um *Núcleo Executor*, um módulo de *Pré-processamento* e um *Núcleo de Análise*. O processamento do *PCodigo* se inicia após o professor disponibilizar via *Moodle*, na *Visão do Professor*, *Exercícios de Programação* e após os alunos iniciarem a *Submissão* de soluções para esses exercícios na *Visão do aluno*.

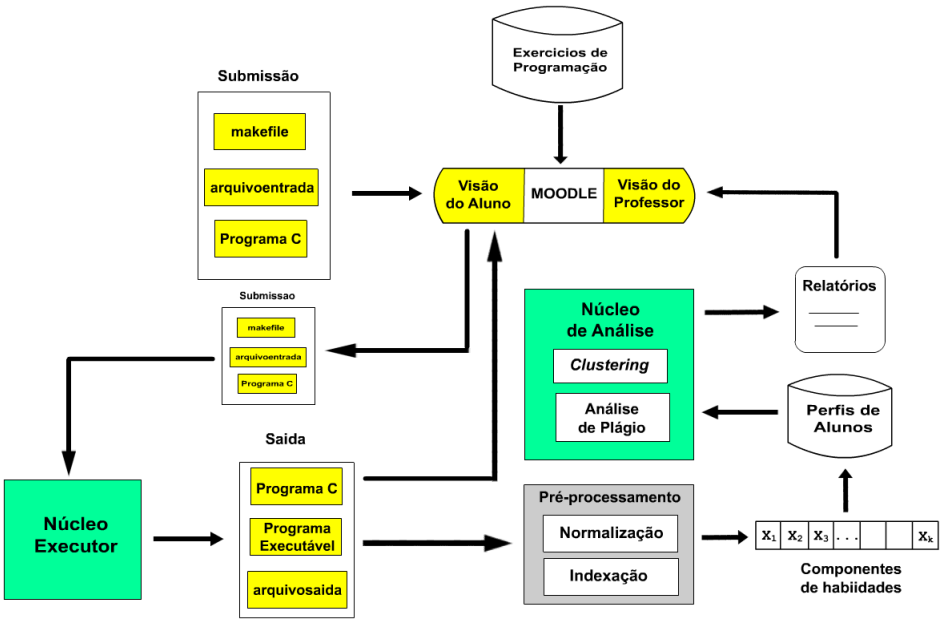

Figura 1. Arquitetura do PCodigo

A *Submissão*, conforme a Figura 1, deve conter pelo menos um *Programa .C*, um arquivo de entrada e um arquivo *makefile* com as instruções de entrada, processamento e saída para execução de um programa. Uma vantagem do modelo de submissão com

makefile é a flexibilização do formato de submissão, que permite o aluno gerenciar a execução do seu programa desde o recebimento da entrada até a geração da saída.

Quando a *Submissão* de um aluno chega ao *Núcleo Executor* do *PCodigo*, é gerada uma *Saída* contendo o *Programa .C*, o programa *Executável* e o *Arquivosaida* com os resultados do *Executável*. O *Arquivosaida* pode ser visualizado pelo aluno na *Visão do Aluno* e pelo professor, nos *Relatórios* emitidos pelo *PCodigo* na *Visão do Professor*.

O *Núcleo Executor*, a cada três horas, executa de uma só vez todas as submissões encaminhadas pelo *Moodle* ou por outras interfaces *web* ao diretório de execução do *PCodigo*. As submissões que já foram executadas e as submissões que não seguem o padrão *PCodigo* de submissão não são processadas.

No módulo *Pré-Processamento* da Figura 1, os arquivos da *Saída* são submetidos a um processo de *Normalização* que gera em arquivo uma representação compacta de uma submissão. Na *Normalização*, retiramos os comentários e os textos entre aspas (*strings*) do programa-fonte substituindo-os pelos *tokens @coment* e *@textstr*, respectivamente.

Para retirar ambiguidades do código, substituimos os símbolos ambíguos (como o operador "++", que pode ser lido como dois operadores "+" na Linguagem C) por *tokens* precedidos por @ (Exemplo:*@inc*, representando o operador ++). Além disso, por conter a palavra reservada *int* da Linguagem C, substituimos o comando *printf* pelo *token @imprimir*. Finalizando a normalização, acrescentamos ao arquivo normalizado o *token @compila*, se o programa compilou e o *token @executa*, se o programa funcionou. Os programas da Tabela 1, na Subseção 4.2, são exemplos de programas normalizados.

O arquivo gerado na *Normalização* é submetido à *Indexação*, que gera uma representação desse arquivo em vetor, onde cada dimensão é a frequência de ocorrência de cada *token* no arquivo normalizado. Na Figura 1, chamamos o vetor gerado de *Perfil de Aluno* e as suas dimensões, de *Componentes de habilidades* [Oliveira and Oliveira 2014].

É importante destacar que a retirada de *strings* e comentários na *Normalização* e a não contagem das variáveis na *Indexação* facilitam o reconhecimento de plágios, já que essas partes, em geral, são as mais alteradas por alunos para disfarçarem seus plágios.

No *Núcleo de Análise* da Figura 1, os *Perfis de Alunos* associados a um mesmo exercício de programação são reunidos em uma Matriz *A* para serem analisados e comparados. Através de algoritmos de *Clustering*, os vetores representantes das soluções dos alunos são reunidos em agrupamentos conforme as similaridades entre eles. Um exemplo de gráfico de *Clustering* gerado é apresentado na Figura 3, na Subseção 4.3.

No módulo de *Análise de Plágio* do *Núcleo de Análise* da Figura 1, os vetores da Matriz *A* são comparados dois a dois e é gerada uma Matriz A_s, formada pelos índices de similaridade entre cada par de vetores. Esses índices são calculados pela medida de similaridade *cosseno*, que é obtido pelo produto interno entre dois vetores. Os índices de similaridade variam de 0 (dissimilaridade) a 1 (similaridade total).

Para análise do professor, o módulo de *Análise de Plágio* retorna em um arquivo apenas os pares de vetores com índices de similaridade acima de 0.9, isto é, com semelhanças entre si acima de 90%. Um exemplo desse relatório de plágio pode ser visualizado na Figura 2 da Subseção 4.2.

Após os processos de execução e análise, o *PCodigo* emite, na *Visão do Professor*, para cada exercício, *Relatórios* de todas as submissões. Através desses relatórios, o professor poderá tomar decisões de avaliações, bem como ter uma visão mais ampla de como seus alunos progridem individualmente na prática da programação.

4. Estudos de Casos

O *PCodigo* está sendo utilizado desde 2014 por turmas de programação dos cursos de Engenharia e de Computação da Universidade Federal do Espírito Santo.

Através do *Moodle* e de outras interfaces *web*, o *PCodigo* já recebeu cerca de 16.390 submissões de arquivos para 665 exercícios de programação.

Na fase inicial dos cursos de programação, os alunos têm algumas dificuldades com o formato de submissão do *PCodigo*, pois eles devem submeter, para cada exercício, pelo menos três arquivos: o *makefile*, o *arquivoentrada* e o programa em Linguagem C. Mas, à medida que os programas aumentam em complexidade, os alunos compreendem melhor as vantagens desse formato para testagem, especialmente, quando há entradas com grande quantidade de dados e vários casos de testes.

Os relatórios do *PCodigo* são compostos pelos arquivos de saída gerados pela execução das submissões, por gráficos de clusterização (Figura 3) e por arquivos contendo os pares de soluções suspeitas de plágio na forma vetorial (Figura 2).

Para os professores, esses relatórios são muito úteis para o diagnóstico, comparação e classificação das soluções desenvolvidas pelos alunos.

4.1. Execução em massa de exercícios de programação

Inicialmente, os programas submetidos pelos alunos ao *PCodigo* eram processados em um único servidor no próprio sistema operacional desse servidor. Todavia, logo percebemos as desvantagens dessa abordagem. Várias vezes tivemos de reiniciar o servidor por falta de espaço em disco em decorrência de códigos que geravam enormes arquivos de saída. Outro problema com o qual tivemos que lidar foi o de programas em *loop*. Nesse caso, os demais programas, na sequência, não eram processados.

Como consequência desses e de muitos outros problemas na compilação e execução dos códigos submetidos de uma turma, adotamos o uso de máquinas virtuais distribuídas para o processamento em massa desses programas.

Em nossa estrutura, utilizamos um servidor, isto é, uma máquina *mestre*, para coordenar o particionamento das submissões entre as várias máquinas virtuais, isto é, as máquinas *escravas*. Dessa forma, uma máquina virtual pôde receber um ou mais conjuntos dos arquivos submetidos através do *Moodle* pelos alunos.

A distribuição dos arquivos entre as máquinas *escravas* é feita para se alcançar um balanceamento de carga. Com isso, tivemos como diminuir o tempo de execução do conjunto de códigos submetidos. Por outro lado, um código problemático, com essa nova arquitetura, passou a ficar retido em uma dada máquina virtual.

Quando é esgotado o tempo máximo para processamento do código, que é de cinco segundos, desativamos essa máquina e o conjunto de arquivos enviados para essa máquina é marcado como encerrado de forma anormal.

A máquina *mestre* possui um processador *AMD Phenom II X4 B95* com 3.0 Ghz e 8GB de memória RAM. Já a máquina *escrava* possui um processador Intel *Core-I7* com 2.80Ghz e 12GB de memória RAM.

Essa nova forma de apoiar a prática de programação vem abrindo oportunidades para que alguns professores possam demandar dos alunos exercícios mais complexos. Além disso, os professores estão também desenvolvendo exercícios com tempo de processamento mais longo. Isso porque o tempo de execução de todos os códigos submetidos não está mais relacionado à soma de cada um deles, mas ao tempo máximo utilizado por uma das máquinas virtuais.

4.2. Análise de indícios de plágios

No relatório de plágio emitido pelo *PCodigo* (Figura 2), são mostradas as soluções com índices de similaridade acima de 90%, em relação à solução *al80*. Essas soluções são identificadas, respectivamente, como *al85*, *al35*, *al94* e *al39*. Observamos que a solução *al85*, com similaridade de 100% em relação à *al80*, conforme indicam os vetores de ambas, são praticamente idênticas.

No entanto, as soluções *al80* e *al85* da Figura 2 diferenciam-se principalmente em relação aos dois primeiros valores de seus vetores, que representam, respectivamente as componentes de habilidades *compila* e *executa*. Nessas componentes, o valor 1 indica que o programa funciona e 0, que não funciona. Nesse caso, embora as soluções sejam muito semelhantes, *al80* funciona e *al85* não funciona.

```
al80; 1 1 0 0 0 0 0 0 6 3 0 0 0 0 0 2 1 0 13 0 0 0 0 1 15 0 0 0 0 12 0 11 2 3 0 0 0 9 3 10 0 0 0 1 0 0 4 0 0
al85; 0 0 0 0 0 0 0 0 6 3 0 0 0 0 0 2 1 0 14 0 0 0 0 1 15 0 0 0 0 10 0 11 2 3 0 0 0 9 3 10 0 0 0 1 0 0 3 0 0

Similaridade: 1 (100%)

al80; 1 1 0 0 0 0 0 0 6 3 0 0 0 0 0 2 1 0 13 0 0 0 0 1 15 0 0 0 0 12 0 11 2 3 0 0 0 9 3 10 0 0 0 1 0 0 4 0 0
al35; 1 1 0 0 0 0 0 0 0 6 0 0 0 0 0 0 0 2 1 0 9 1 0 0 0 1 15 0 0 0 0 7 0 10 6 6 0 1 0 9 3 9 0 0 0 0 0 0 1 0 0 0

Similaridade: 0.95 (95%)

al80; 1 1 0 0 0 0 0 0 6 3 0 0 0 0 0 2 1 0 13 0 0 0 0 1 15 0 0 0 0 12 0 11 2 3 0 0 0 9 3 10 0 0 0 1 0 0 4 0 0
al94; 1 1 0 0 0 0 0 0 0 9 0 0 0 0 0 0 0 2 1 0 10 0 0 0 0 1 15 0 0 0 0 3 0 14 2 3 0 0 0 9 3 10 0 0 0 1 0 0 3 0 0
Similaridade: 0.93 (93%)

al80; 1 1 0 0 0 0 0 0 6 3 0 0 0 0 0 2 1 0 13 0 0 0 0 1 15 0 0 0 0 12 0 11 2 3 0 0 0 9 3 10 0 0 0 1 0 0 4 0 0
al39; 1 1 1 3 0 0 0 0 2 1 0 0 0 0 0 1 1 0 6 0 0 0 0 1 5 0 1 0 0 8 0 3 1 0 0 1 0 3 1 3 0 0 0 0 0 0 2 0 0 0 0

Similaridade: 0.91 (91%)
```

Figura 2. Análise de Programas

Um professor de programação, considerando que as variáveis e *strings*, que são as partes que os alunos mais alteram para disfarçar plágios, não foram contadas na indexação e que a solução *al85* não funciona, suspeitaria que *al85* é um plágio de *al80*.

Ao comparar os códigos normalizados das soluções *al80* e *al85* na Tabela 1, as suspeitas se evidenciam. Nesses códigos o professor avaliou o uso de estruturas de repetição e se o aluno conseguiria resolver, com apenas duas expressões lógicas em uma estrutura *if* aninhada, por número de pontos, o problema de identificar o campeão e o vice-campeão de um campeonato de futebol. A pontuação do professor variou de 0 a 5, sendo que seria descontado um ponto se o programa não compilasse.

Na análise de soluções, observa-se que *al85* (Tabela 1, linhas 10-18) contém a mesma estrutura condicional de *al80* (Tabela 1, linhas 12-21). As duas soluções utiliza-

ram mais de duas expressões lógicas conforme os trechos de códigos. Para o professor, essa construção com excesso de expressões lógicas não está correta. Dessa forma, considerando que em programação é mais comum as soluções corretas serem mais semelhantes entre si do que as incorretas, as similaridades entre trechos incorretos trazem mais evidências de ocorrência de plágio.

Solução al80

```
1.@compila  @funciona
2.#include <stdio.h>
3.#include <stdlib.h>

4. int main(int argc, char *argv[])
5. {
6.    int x=0, y1=0, y2=0, P, GP, GN,
7.        VF, VC, E;
8.    int T1=0, T2=0, T3=0;

9. @imprimir(@textostr, &GP, &GN, &VF, &VC, &E);
10. P = 5*GP - GN + 3*VF + 3*VC + E;
11.    T1 = P;
                ...

12.     if((T1>T2) @e (T1>T3))
13.     {
14.         @imprimir( @textostr );
15.         if(T2>T3){
16.                     @imprimir( @textostr );
17.         }
18.         else{
19.                     @imprimir( @textostr );
20.             }
21.     }              ...

22.  system( @textostr );
23.  return 0;
24. }
```

Solução al85

```
1. #include <stdio.h>
2. #include <stdlib.h>

3. int main(int argc, char *argv[])
4. {
5. int time1, time2, time3, P=0, GP, GN,
6.        VF, VC, E, T1=0, T2=0, T3=0;

7.  @imprimir(@textostr, &GP,&GN,&VF,&VC,&E);
8.  P = 5*GP - GN + 3*VF + 2*VC + E;
9.  T1=P;
                ...

10.  if((T1>T2) @e(T1>T3)){
11.              @imprimir( @textostr );
12.              if(T2>T3){
13.
@imprimir( @textostr );
14.                         }
15.                         else{
16.
@imprimir( @textostr );
17.                                 }
18.                                 }
              ...

19.       @imprimir( @textostr );
20.  return 0;
21. }
```

Tabela 1. Programas suspeitos de plágio

Conforme havíamos falado, é comum o plagiador de códigos de programação fazer alterações na declaração das variáveis. Isso pode ser observado claramente na Tabela 1, comparando as linhas 5-6 de *al85* com as linhas 6-8 de *al80*. Além dessas alterações, o autor da solução *al85* fez outras pequenas modificações, mas o seu programa não funcionou, conforme a ausência dos *tokens* *@compila* e *@executa* em seu código. Isso evidencia que o autor de *a85*, além de não ter compreendido a construção de *al80*, não compreendeu as próprias modificações que fez, o que praticamente confirma sua fraude.

Nesse estudo de caso, o *PCodigo* identificou a similaridade de 100% entre as soluções *al80* e *al85*, mas o professor não. Prova disso é que o professor atribuiu nota 3.7 a *al80* e 2.7 a *al85* porque este último não funcionou.

As soluções *al80* e *al85* da Tabela 1 são apenas dois exemplos de 100 amostras corrigidas manualmente por um professor. Seria, portanto, difícil para esse professor identificar todas as suspeitas de plágio e analisá-las. Por isso, o *PCodigo* mostrou-se como uma ferramenta adequada para auxiliá-lo na correção de exercícios oferecendo-lhe recursos de representação, visualização e análise de indícios de plágios.

4.3. Reconhecimento de soluções divergentes e classes de soluções

O gráfico de *Clustering* da Figura 3 é emitido nos relatórios do *PCodigo* para auxiliar o professor na análise das soluções desenvolvidas por seus alunos. Nesse gráfico é possível visualizar simultaneamente cerca de 40 exercícios, verificar que instruções, estruturas e operadores da Linguagem C os alunos utilizaram e identificar as soluções que funcionam.

No gráfico da Figura 3, cada *cluster* é separado por uma linha horizontal preta. As linhas do gráfico são os vetores das submissões e as colunas, a frequência de ocorrência de *tokens* da Linguagem C. As tonalidades de cores em cada relação *Perfil x Componente de habilidade* indicam quão importante é uma *Componente de habilidade* em um *Perfil*. Dessa forma, quanto mais forte for a tonalidade, mais importante será a componente e, quanto mais próxima for da cor branca (de importância nula), menos importante ela será.

Observe no gráfico da Figura 3 que as componentes com tonalidades mais fortes em cada perfil de um mesmo *cluster* são as mesmas. Os algoritmos de *Clustering* apontam essas características como descritivas de um *cluster*.

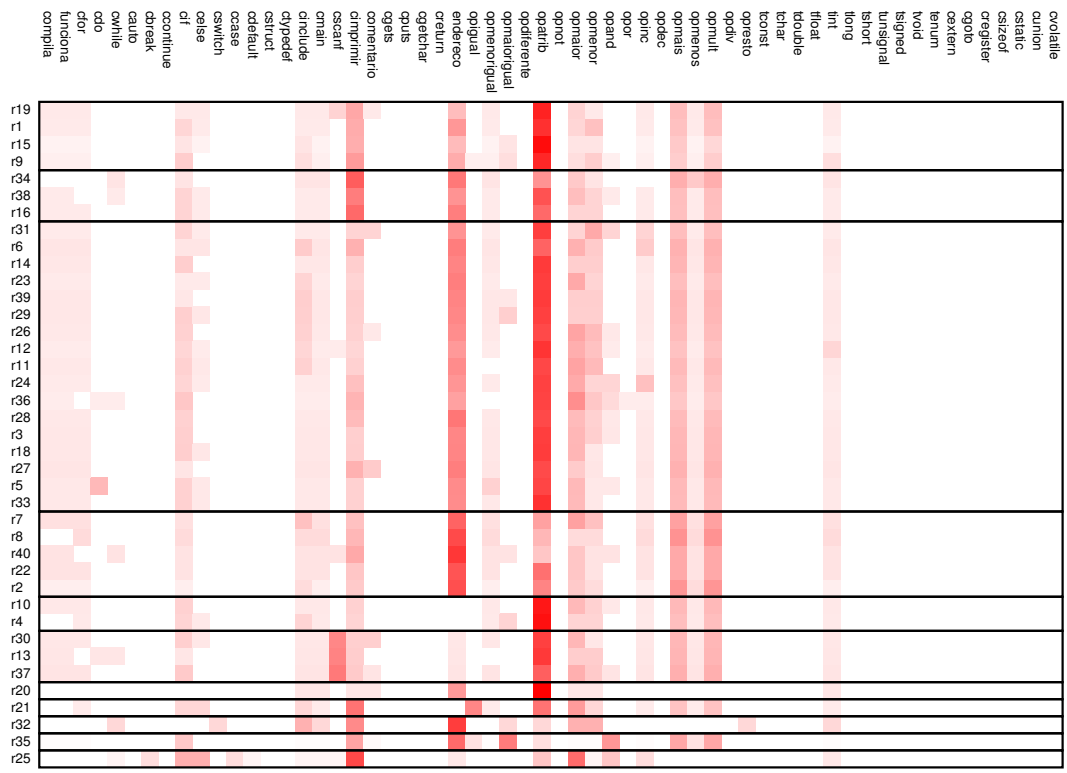

Figura 3. Gráfico de Clustering

No gráfico da Figura 3, os *clusters* da parte inferior que contêm apenas uma solução, isto é, os *clusters* de *r25*, *r32*, *r35*, *r20* e *r21*, respectivamente de baixo para cima, isolam as soluções que divergem da solução de gabarito. Uma evidência disso é que todas elas receberam notas muito baixas em uma escala de 0.0 a 5.0. A Tabela 2 apresenta as características dessas soluções e explica por que elas foram consideradas divergentes.

A solução *r32*, por exemplo, de acordo a solução do gabarito na Tabela 3, é uma solução divergente. Ao comparar as duas soluções, observa-se claramente que *r32* só

Análise de soluções

Solução	Nota	Caracterização
r25	0.25	Excesso de instruções
r32	0.0	Programa muito curto sem coerência lógica
r35	0.0	Excesso de instruções
r20	0.0	Programa contendo apenas instruções iniciais
r21	0.25	Confusão nas expressões lógicas

Tabela 2. Análise de divergência de soluções

apresenta alguma semelhança com o gabarito nas linhas iniciais, isto é, nas linhas 1 a 5. A partir daí, o autor de *r32* perdeu as rédeas no desenvolvimento de sua solução.

Gabarito

```
1. @compila  @funciona
2. #include<stdio.h>
3. #include<math.h>
4. main()

5. {
6.      int pnts, result = 0, tchamp = 0,
7.          pchamp = 0, tvice = 0, pvice = 0, i;
8.      for(i = 1; i@menorigual 3; i@inc)
9.      {
                ...

10.     if(result @maiorigual pchamp)
11.     {
12.             pvice = pchamp;
13.             tvice = tchamp;
14.             pchamp = result;
15.             tchamp = i;
16.     }
17.     else if(result @maiorigual pvice)
18.     {
19.             pvice = result;
20.             tvice = i;
21.     }
22.     }               ...
23. }
```

Solução divergente r32

```
1. #include <stdio.h>
2. #include <math.h>

3. main ()
4. {
5. int gp,gn,vf,vc,e,camp,vice,times=0;
          ...

6. while(times@maiorigual10)
7. {
8. switch
9. }
```

Tabela 3. Solução do gabarito e solução divergente

O reconhecimento automático de classes de soluções e de soluções divergentes de um exercício é importante para descobrir as possíveis soluções para um problema e avaliar os casos atípicos de soluções. Além disso, as soluções mais representativas de cada *cluster* podem ser utilizadas como gabaritos de soluções para avaliação manual e até para geração de modelos de avaliação semi-automática de exercícios.

5. Considerações Finais

Este trabalho apresentou o *PCodigo* como um instrumento adequado de apoio à prática assistida de programação por oferecer recursos de execução em massa e de análise de exercícios submetidos por alunos via *Moodle*.

As principais vantagens do *PCodigo* em relação a outros sistemas de submissão de exercícios de programação são a flexibilização do formato de submissão, a execução

distribuída de programas através de máquinas virtuais e os recursos de análise de plágios e de classificação de soluções.

Como trabalhos futuros a partir deste, sugerimos a integração dos recursos do *PCodigo* a outros ambientes virtuais e a criação de outros mecanismos de representação de códigos, como a representação de perfis por medidas que quantifiquem esforço e qualidade de programação.

Em resumo, a contribuição do *PCodigo* para a aprendizagem de programação é oferecer ao professor mais agilidade e melhor acompanhamento individual da aprendizagem e, ao aluno, *feedbacks* mais imediatos.

Referências

Campos, C. and Ferreira, C. (2004). Boca: um sistema de apoio para competições de programação. In *XII Workshop de Educação em Computação (WEI) - SBC 2004*, Salvador, BA.

De Oliveira, M. G., Marques Ciarelli, P., and Oliveira, E. (2013). Recommendation of programming activities by multi-label classification for a formative assessment of students. *Expert Systems with Applications*.

De Souza, D., Maldonado, J., and Barbosa, E. (2011). Progtest: An environment for the submission and evaluation of programming assignments based on testing activities. In *Software Engineering Education and Training (CSEE T), 2011 24th IEEE-CS Conference on*, pages 1 –10.

França, A., Soares, J., Gomes, D., and G.C.Barroso (2011). Um sistema orientado a serviços para suporte a atividades de laboratório em disciplinas de técnicas de programação com integração ao ambiente Moodle. *RENOTE - Revista Novas Tecnologias na Educação*, 9(1).

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

Moreira, M. P. and Favero, E. L. (2009). Um ambiente para ensino de programação com feedback automático de exercícios. In *XVII Workshop Sobre Educação em Computação (WEI) - CSBC 2009*.

Moussiades, L. and Vakali, A. (2005). Pdetect: A clustering approach for detecting plagiarism in source code datasets. *The computer journal*, 48(6):651–661.

Oliveira, M., Monroy, N., Zandonade, E., and Oliveira, E. (2014). Análise de componentes latentes da aprendizagem de programação para mapeamento e classificação de perfis. In *Anais do Simpósio Brasileiro de Informática na Educação (SBIE 2014)*, volume 25, pages 134–143.

Oliveira, M. and Oliveira, E. (2014). Metodologia de Diagnóstico e Regulação de Componentes de Habilidades da Aprendizagem de Programação. In *XXII Workshop sobre Educação em Computação (WEI) - CSBC 2014*, Brasília, DF. SBC.