Inferência de Conhecimento a Partir da Detecção Automática de Evidências no Domínio da Programação de Computadores

Andres J. Porfirio[1,2], Roberto Pereira[1], Eleandro Maschio[2]

[1]Departamento de Informática – Universidade Federal do Paraná (UFPR)
Curitiba – PR – Brasil

[2]Coordenação do Curso de Tecnologia em Sistemas para Internet
Universidade Tecnológica Federal do Paraná (UTFPR)
Guarapuava – PR – Brasil

{ajporfirio,rpereira}@inf.ufpr.br, eleandro@hotmail.com

Abstract. *During the computer programming learning process, the student's progress is marked by new skills acquisition. The monitoring of this progress, from the teacher's perspective, can be a complex task when performed from manual perceptions. The situation is even more complex when dealing with a large number of students simultaneously. One way of supporting this activity involves the use of student models, where abilities are mapped in order to facilitate the visualization of the concepts already acquired and the gaps to be completed. The use of this type of tool is beneficial, but it can still need a heavy workload when it requires the model manual feeding, especially because it is necessary to continuously update it. This paper proposes the use of automatic mechanisms, based in source codes analysis, followed by the evidences detection, as inputs for the student model. A set of experiments that demonstrate the viability of the method is presented. The test scenario is composed by a dynamic bayesian network student model and databases of source code in C language.*

Resumo. *Durante o processo de aprendizado de programação de computadores, o progresso dos alunos é marcado pela aquisição de novas habilidades. O acompanhamento desse progresso, na perspectiva do professor, pode ser uma tarefa complexa quando realizada a partir de percepções manuais. A situação se agrava quando se trata do acompanhamento simultâneo de uma grande quantidade de alunos. Uma das formas de apoio a essa atividade envolve o uso de modelos do aprendiz, onde as habilidades do aluno são mapeadas a fim de facilitar a visualização tanto dos conceitos já adquiridos quanto das lacunas a serem preenchidas. O uso desse tipo de ferramenta é benéfico, porém ainda pode exigir uma grande carga de trabalho quando se trata da alimentação manual dos modelos, sobretudo devido à necessidade de atualização contínua. Este trabalho propõe o uso de mecanismos automáticos, baseados na análise de códigos-fonte seguida da detecção de evidências, como fonte de alimentação para o modelo do aprendiz. É apresentado um conjunto de experimentos que demonstram a viabilidade do método. Utiliza-se como cenário de testes um modelo de aprendiz baseado em uma rede bayesiana dinâmica e bases de dados de códigos-fonte em linguagem C.*

DOI: 10.5753/cbie.sbie.2018.1553

1. Introdução

O ensino da programação de computadores é um dos tópicos mais importantes na formação de alunos em cursos da área de Computação. Um dos desafios encontrados nesse processo é a dificuldade de monitoramento do aluno com relação à aquisição de novas habilidades. Essa dificuldade se deve a vários fatores, dentre eles: grande quantidade de tópicos a serem abordados, incertezas na avaliação de conceitos subjetivos, dificuldade de tratamento individual quando existe a necessidade de lidar com um grande número de alunos simultaneamente e, dificuldade gerada pela repetição do processo em pequenos intervalos de tempo (avaliação contínua do aluno).

Alguns autores, como Woolf (2007), Pimentel e Direne (1998) e Maschio (2013) sugerem, em apoio às abordagens e técnicas de ensino tradicionais, o uso de ferramentas mais elaboradas, tais como Sistemas Tutores Inteligentes (STI). No contexto deste trabalho, um dos principais benefícios desses sistemas é a capacidade de individualização dos alunos. Isso se dá por meio da construção e atualização de um modelo do aprendiz, que pode ser entendido como um mapa de habilidades, o qual descreve um conjunto de unidades de conhecimento a serem dominadas pelo estudante ao longo do processo de aprendizagem.

A capacidade de tratamento personalizado para cada estudante, proporcionada pelo uso do modelo do aprendiz, é um recurso bastante interessante, porém dependente da identificação de evidências de aprendizado demonstradas pelo aluno. Tal identificação pode ser encarada como a avaliação dos conhecimentos do aprendiz, seguida da aplicação desse resultado no modelo. Com isso, o estado do mapa de habilidades pode ser atualizado de modo a refletir a capacidade cognitiva do aluno frente aos conteúdos ministrados.

A atualização do mapa de habilidades pode se beneficiar do uso de técnicas automáticas, entretanto, esta é uma lacuna a ser preenchida quando o contexto é a programação de computadores. Assim, este trabalho propõe o uso de mecanismos automáticos para análise de códigos-fonte e localização de informações para atualização contínua do modelo do aprendiz.

1.1. Objetivos

O objetivo geral deste trabalho é demonstrar a viabilidade da inferência de conhecimento a partir da detecção automática de evidências de aprendizado no domínio da programação de computadores. Este objetivo é alcançado por meio da elaboração de estratégias de avaliação de códigos-fonte, responsáveis pela localização de indícios de aprendizado de conceitos de programação e, sua aplicação no modelo, sob a forma de evidências.

São objetivos específicos: **(1)** definir um modelo de aprendiz; **(2)** definir estratégias de busca de indícios; **(3)** implementar as estratégias sob a forma de algoritmo; **(4)** aplicar o método em casos de teste previamente conhecidos; **(5)** aplicar o método em códigos-fonte de alunos reais; **(6)** comparar os estados do modelo em diferentes instantes de tempo a fim de verificar a progressão dos conhecimentos do aluno.

2. Trabalhos Relacionados

A construção de modelos de aprendiz é um tema vastamente abordado em pesquisas da área de STI. Especificamente, existem alguns nichos de pesquisa relacionados ao

modelo do aprendiz: **(a)** formas de representação do conhecimento [Maschio 2013]; [Vier et al. 2015]; **(b)** métodos de descoberta do modelo [Li et al. 2011]; **(c)** métodos de aperfeiçoamento do modelo [Koedinger et al. 2012]. Entretanto, há demanda por pesquisas relacionadas especificamente à identificação automática de informações para atualização de modelos de aprendiz no domínio da programação de computadores.

De maneira genérica, Beverly Woolf cita a atualização do modelo como um dos problemas e desafios relacionados à representação dos conhecimentos do aluno [Woolf 2007]. A autora cita o uso de métodos de comparação, que se baseiam na análise de semelhança entre as soluções do aluno e as soluções de referência, desenvolvidas por um indivíduo experiente no assunto. De forma análoga, se apresentam as técnicas baseadas em questionários, que podem ser entendidas como formas de comparar o conhecimento do aluno com um gabarito de soluções. Estas soluções, idealmente, refletem os conhecimentos de um indivíduo experiente no assunto. Na área da programação de computadores, pesquisas como as de Pimentel et al. (2003) comprovam que este tópico vem sendo abordado há bastante tempo.

Outros autores, como Galasso e Moreira (2014) e Aleman (2011) , citam o uso de ferramentas automatizadas para a identificação de indícios de aprendizado. Em ambos os casos, as soluções envolvem a análise automática de códigos-fonte submetidos pelo aluno. Galasso e Moreira apresentam uma extensão para o ambiente Moodle, na qual o aluno efetua resoluções de exercícios de programação e o próprio ambiente on-line realiza a correção. Tal correção é feita pela comparação dos resultados do algoritmo do aluno ao ser avaliado com casos de teste previamente definidos. De maneira similar, Aleman apresenta uma adaptação do sistema Mooshak, entretanto, além de avaliar os códigos-fonte, também realiza a avaliação de *scripts* de compilação (Makefile), depuração (GDB), documentação de código (doxygen), entre outros. Apesar do uso de diferentes fontes de informação, os trabalhos mencionados ainda pecam no que diz respeito à avaliação de detalhes da construção da solução. Por exemplo: o uso de casos de teste, com a inserção de entradas e comparação das saídas, pode ser vulnerável a soluções inadequadas que retornam um resultado correto, como o uso de múltiplos de comandos de impressão em substituição a um único comando dentro de um laço de repetição.

Tratando de detalhes da construção do código, existe um nicho de pesquisa que se concentra na avaliação automática (ou semi-automática, em alguns casos) de códigos-fonte. Pesquisas como a de Saikkonen et al. (2001) demonstram que o tema é explorado há bastante tempo. Os autores destacam que o sistema implementado é capaz de avaliar de forma individual cada procedimento, ao invés de simplesmente conferir o resultado. Além disso, o sistema PROUST, apresentado por Johnson e Soloway (1985) , possui como um de seus recursos a capacidade de interpretação dos passos utilizados pelo aprendiz para chegar à solução.

Ainda abordando o tema da correção de exercícios, porém com menos enfoque na análise da construção do código, o trabalho de Buyrukoglu et al. (2016) sugere que existem esforços recentes sendo dedicados nesta área. Nesse trabalho, os autores apresentam mecanismos para a detecção e agrupamento de blocos de código similares construídos por diferentes alunos. De acordo com os autores, o mecanismo proporciona uma redução da carga de trabalho do avaliador, uma vez que a avaliação de um bloco de código pode ser replicada aos blocos similares.

Conforme exposto, as pesquisas em STI focados em programação de computadores possuem várias vertentes quanto à avaliação de tarefas do aluno. Entretanto, nos trabalhos citados, não foram percebidas indicações do uso dessas informações como base para alimentação de modelos do aprendiz.

3. Método Utilizado

A descrição do método é organizada em duas etapas, que tratam da definição do modelo do aprendiz e da definição das estratégias de avaliação de código-fonte. Essas etapas são descritas a seguir.

3.1. Definição do Modelo do Aprendiz

A definição do modelo do aprendiz é a etapa inicial da construção do ambiente de experimentação. Dado o interesse em demonstrar a evolução do progresso do aluno, considerou-se o uso de estruturas capazes de representar informações em diferentes estados e diferentes instantes de tempo. Duas técnicas foram avaliadas: **(a)** o grafo de sobreposição, apresentado por [Maschio 2013] e, **(b)** as redes bayesianas dinâmicas (também chamadas de redes bayesianas temporais), utilizadas por [Seffrin and Jaques 2015] [Vier et al. 2015]. Considerando o escopo desta pesquisa, ambas as técnicas apresentaram características e capacidades de modelagem muito próximas. Optou-se pela segunda alternativa, devido à possibilidade de lidar com incertezas e intervalos de tempo.

O modelo foi populado com um subconjunto de unidades de conhecimento, inspirado no conjunto de perícias definido por [Maschio 2013]. O subconjunto foi baseado em conceitos ministrados em etapas iniciais de cursos de programação, especificamente em linguagem C, considerando os tipos de dados *int*, *float*, *char* e *char[]*:

- **Tipos e Literais**: declaração;
- **Entrada**: leitura de dados (*scanf*) e uso de múltiplos argumentos;
- **Saída**: impressão de dados (*printf*) e uso de múltiplos argumentos;
- **Variáveis**: inicialização de variáveis;
- **Constantes**: declaração e inicialização de constante, ausência de tentativa de alteração do valor;
- **Divisão por zero**: ausência de divisão por zero na execução de expressões aritméticas.

3.2. Definição de Estratégias

Dado o subconjunto de conceitos que compõe o modelo, foram elaboradas estratégias para avaliação de códigos-fonte e localização de indícios de aprendizado. As estratégias foram elaboradas sob a forma de algoritmos, os quais recebem como entrada um código-fonte e retornam como resultado um percentual de ativação, que indica se o aluno utiliza de forma correta determinado conceito. O retorno do algoritmo ainda fornece um relatório das evidências localizadas (esses detalhes são utilizados como métrica para avaliação das estratégias nos experimentos dos cenários controlado e real, citados posteriormente).

Os mecanismos utilizados para a composição dos algoritmos de busca de evidências, foram os seguintes: **(a)** localização de padrões com expressões regulares; **(b)** decomposição e localização de elementos com o uso de *parsers*; **(c)** execução automática

supervisionada por *scripts* de depuração (gdb), associados a casos de teste pré-definidos; e **(d)** captura de exceções de execução. Estudos preliminares desses mecanismos foram detalhados em um trabalho anterior [Porfirio et al. 2017].

Cita-se, como exemplo, o algoritmo que localiza evidências da entrada de dados com tipo *int*. Considerando o fragmento de código da Figura 1, existem duas instruções de entrada (*scanf*), sendo uma para a variável *a* e outra para a variável *b*. O mecanismo aplica um *parser* em busca de chamadas da função *scanf*. Uma vez localizadas as chamadas, é realizada a análise dos parâmetros, neste caso, verificando se a formatação do primeiro parâmetro (onde se localizam os tipos de dados a serem lidos, %d e %f) é coerente com o tipo declarado em cada variável (*a* e *b*). No exemplo, o primeiro comando está correto, pois aguarda uma variável de tipo *int* e ela é fornecida (*a*). Já o segundo comando aguarda uma variável do tipo *float*, porém uma variável *int* é fornecida (*b*). Assim, o resultado esperado é a classificação de um único *scanf* correto.

```
void main(){
  int a, b;
  scanf("%d", &a);
  scanf("%f", &b);
}
```

Figura 1. Exemplo de Código-Fonte

4. Experimentos

A demonstração do método é feita com três experimentos. Dois deles são relacionados à detecção automática de evidências. O terceiro apresenta o uso dessas informações para a inferência de conhecimentos em programação de computadores. Detalhes e resultados são descritos a seguir.

4.1. Cenário Controlado

A primeira etapa de testes envolve a aplicação do método em um cenário controlado. Esse cenário contempla uma base de dados artificial, constituída por códigos-fonte escritos pelo autor. Os códigos representam diversas situações, entendidas como comuns nas etapas inciais do aprendizado de programação de computadores: declaração de variáveis e constantes, instruções de entrada e saída e, operações aritméticas.

Foram escritos 29 programas, em linguagem C, com objetivos coerentes com os exercícios aplicados a alunos iniciantes, tais como: leitura e impressão de valores, uso de diferentes tipos de dados (*int*, *float*, *char*, e *char[]*), operações com valores numéricos, impressão de dados vinculadas a desvios condicionais e impressão de dados em laços de repetição. Juntamente com os códigos-fonte, foram escritos casos de teste, contendo valores de entrada e a saída esperada. Tais casos de teste são necessários para a alimentação do programa durante as execuções automáticas, que são realizadas pelos mecanismos de localização de evidências.

Um dos objetivos da base artificial é possibilitar a aplicação de estratégias em códigos-fonte cujo resultado é conhecido. O método envolve a aplicação de estratégias para a detecção de evidências. Assim, é necessário que tais evidências sejam inseridas e catalogadas em cada um dos códigos do cenário controlado.

Nos 29 programas, foram inseridas e catalogadas 92 evidências, classificadas de acordo com o subconjunto de conceitos citado na subseção 3.1. Dado o conjunto de testes catalogado, foi realizada a aplicação e avaliação das estratégias. O resultado de cada algoritmo foi, então, comparado ao catálogo construído por um avaliador humano. O gráfico da Figura 2 apresenta um comparativo entre os resultados dos mecanismos automáticos e o catálogo de evidências inseridas na base de testes.

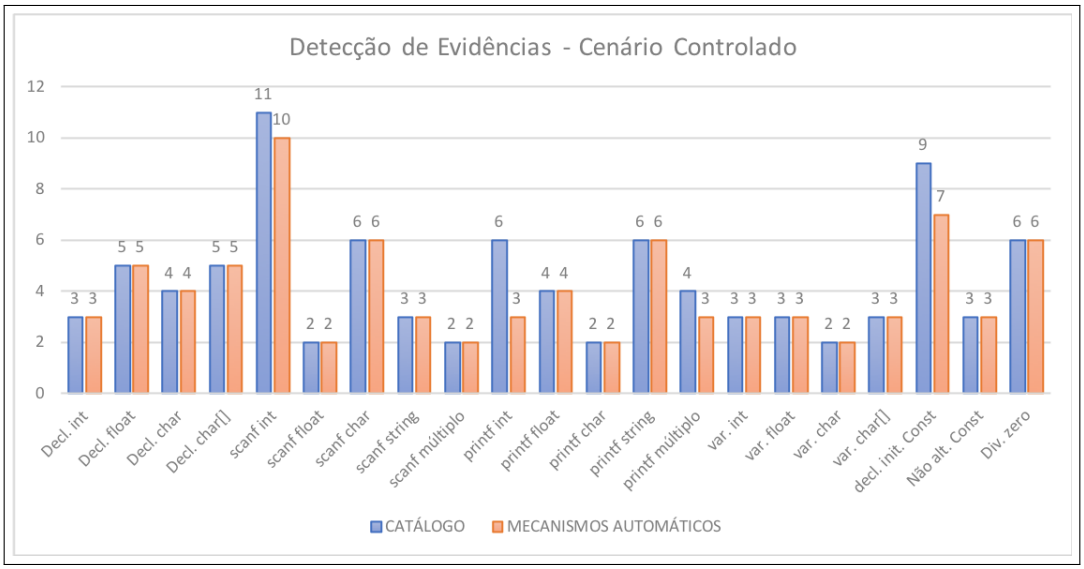

Figura 2. Detecção de evidências no cenário controlado.

O método foi capaz de detectar 85 das 92 evidências, totalizando 92,39% de acerto. Notou-se que a maior porcentagem de erros se deu na evidência de impressão de dados do tipo inteiro (*printf int*). Uma análise revelou que, devido a uma questão de implementação, o mecanismo foi incapaz de identificar argumentos do comando de impressão quando estes são expressões e/ou chamadas de função. Com isso, tais argumentos foram classificados como incorretos, invalidando a evidência. Situações similares foram encontradas na evidência *printf múltiplo*. Além disso, a evidência *scanf int* apresentou um erro relacionado à um argumento não tratado e, por fim, a evidência *declaração. e inicialização de constantes* sofreu perdas devido ao fato de o mecanismo não tratar ponteiros.

4.2. Cenário Real

Uma vez coletados os resultados do cenário controlado, o estudo foi conduzido à execução do método em ambiente real. Foram selecionados 113 códigos-fonte, e seus respectivos casos de teste, da base de dados do STI FARMA-Alg [Kutzke and Direne 2015]. Os critérios aplicados na seleção dos fontes foram os seguintes: **(a)** todos os códigos deveriam ser extraídos de uma mesma turma; **(b)** somente seriam selecionados códigos de alunos que submeteram, pelo menos, 4 soluções, evitando assim ter múltiplas soluções de um mesmo exercício e, consequentemente, aumentando a diversidade dos códigos; **(c)** admitir apenas submissões classificadas pelo STI como corretas.

Tal como realizado na base de testes, os códigos selecionados foram submetidos à catalogação de evidências. Os critérios foram os mesmos do cenário controlado. Esse

processo resultou em um total de 1020 evidências localizadas de forma manual. Devido à natureza dos códigos-fonte, provenientes de resoluções de exercícios reais, alguns recursos não foram encontrados nas resoluções dos alunos: declarações com tipo de dados *char*; variáveis, leitura e impressão de dados do tipo *char* e *char[]*; e constantes. Uma análise manual revelou que as listas de exercícios do STI exigiam, majoritariamente, o uso de tipos de dados numéricos e o uso de constantes não foi mencionado.

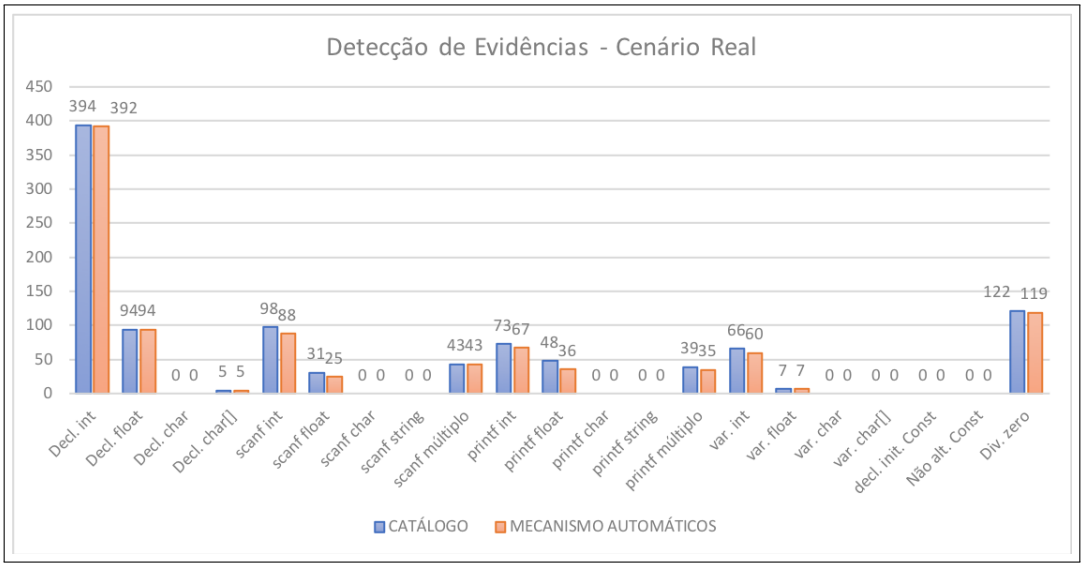

Figura 3. Detecção de evidências no cenário real.

Conforme mostrado no gráfico da Figura 3, os resultados da aplicação do método em ambiente real refletiram a capacidade de identificação de evidências observada no primeiro experimento. Das 1020 evidências, 971 foram localizadas com sucesso, totalizando 95.2% de acerto.

Foi realizada uma análise dos casos errôneos, a qual revelou situações semelhantes às apresentadas no primeiro experimento. Além disso, foram detectados erros em algumas situações não previstas durante o processo de implementação, como: falhas nas expressões regulares responsáveis pela detecção de comandos de entrada (*scanf*) decorrentes de disparidades sintáticas (por exemplo espaços em branco entre os comandos); operações de atribuição por divisão (/=) não foram reconhecidas como divisões; problemas na identificação de comandos de entrada e saída cujos argumentos são posições de vetor.

4.3. Inferência de Conhecimento

Os resultados alcançados pelos mecanismos de detecção automática de evidências estimularam a elaboração de um novo experimento. Este, por sua vez, teve como objetivo demonstrar as possibilidades do uso dessas evidências como fonte de alimentação do modelo do aluno e, ainda, permitir o acompanhamento da evolução dos seus conhecimentos.

O experimento iniciou-se com a construção de um mecanismo de agrupamento de evidências com dois critérios: **(1)** agrupamento de evidências por código-fonte e, **(2)** agrupamento de códigos-fonte por usuário. Assim, os dados foram organizados em uma hierarquia composta por usuários, que contém códigos-fonte, que por sua vez contém

evidências. Desta forma, é possível analisar o conjunto de conceitos utilizados por cada aluno em cada resolução de exercício.

Foram utilizados os mesmos códigos-fonte dos experimentos detalhados nas subseções 4.1 e 4.2. O Resultado dos agrupamentos gerou um total de 19 usuários, dos quais, 6 correspondem aos códigos-fonte da base de dados do cenário controlado, e 13 correspondem à dados extraídos de forma anônima da base real (os agrupamentos se deram pelo ID do usuário, evitando ligações com dados dos alunos).

Em seguida, os conjuntos de evidências foram aplicados no modelo do aprendiz. Este modelo apresenta um mapa das capacidades a serem dominadas pelo estudante ao longo do processo de aprendizagem. Um subconjunto dessas habilidades foi selecionado para demonstração do método, espelhando as categorias de evidências apontadas na subseção 4.1. Todas as evidências foram configuradas com pesos iguais.

Ademais, foi elaborado um mecanismo para exibição e filtragem de códigos-fonte. Com isso, a interface de visualização do modelo permite a seleção de intervalos de tempo, os quais abrigam as resoluções de exercícios submetidas pelo aluno. Esse mecanismo de filtragem pode ser visualizado na parte superior das figuras 4 e 5, onde as esferas azuis representam os códigos-fonte e a área sombreada representa o intervalo de evidências ativo no modelo. Já na parte inferior das figuras, encontra-se um fragmento do grafo de perícias definido por Maschio (2013).

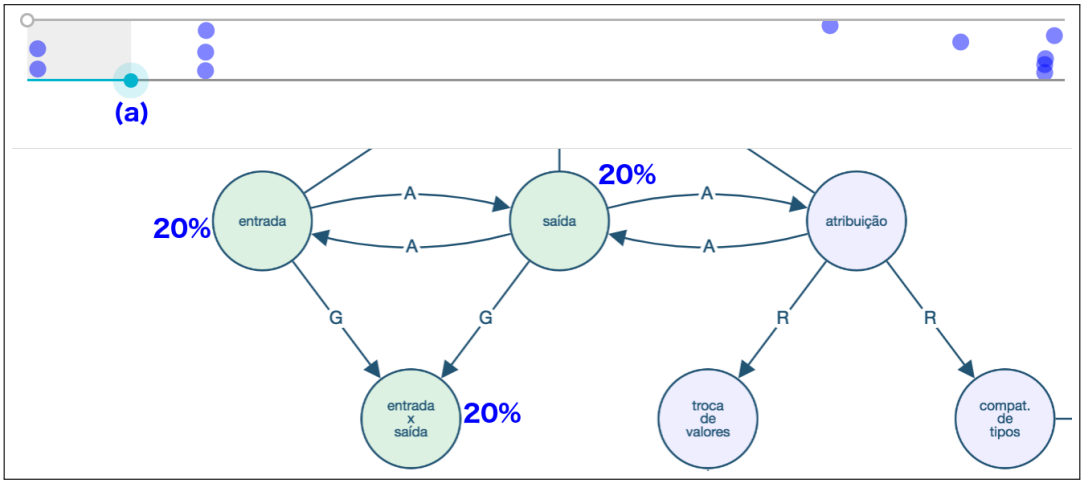

Figura 4. Inferência de Conhecimento: 2 códigos-fonte avaliados.

Ambas as figuras 4 e 5 representam dados de um mesmo aluno da base real, porém em intervalos de tempo diferentes. O estado do modelo na Figura 4 apresenta conjuntos de evidências provenientes de dois códigos-fonte, localizados à esquerda do marcador **(a)**, os quais inferem 20% de ativação em três nodos (entrada, saída e entrada x saída). É possível observar, na Figura 5, várias alterações no estado do modelo. Ao topo, o marcador **(a)** foi deslocado à direita, contemplando três novos códigos-fonte, os quais adicionam novos conjuntos de evidências responsáveis pelo aumento do percentual de ativação nos três nodos supracitados.

Foi realizada uma análise empírica dos 19 modelos gerados, procurou-se observar os agrupamentos de evidências e se alterações do intervalo de tempo refletem em alterações no estado de ativação dos nodos do mapa de habilidades. Em consonância

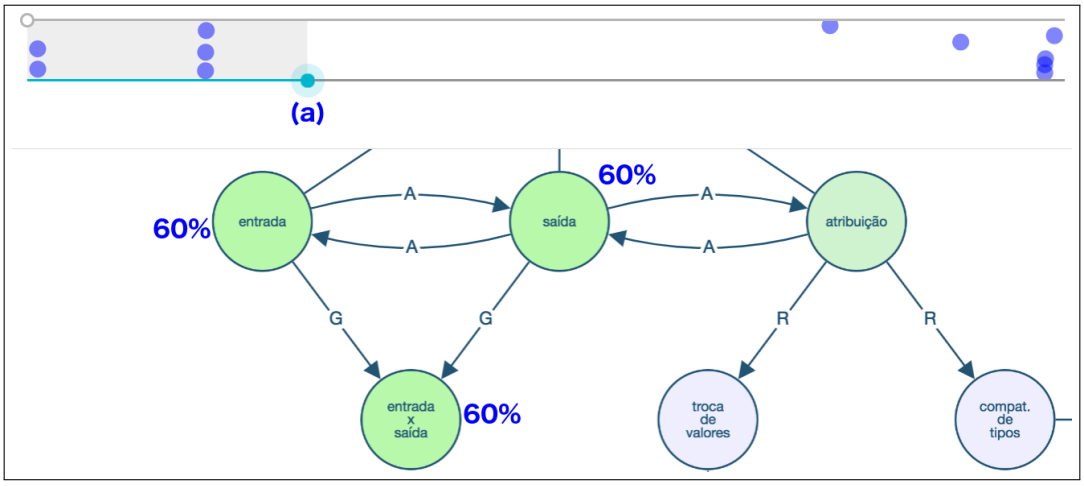

Figura 5. Inferência de Conhecimento: 5 códigos-fonte avaliados.

com o exemplo apresentado, percebeu-se que, em todos os casos, o mecanismo foi capaz de agrupar as evidências por código-fonte e exibir o estado do modelo de acordo com o intervalo de tempo selecionado.

5. Considerações Finais

Este trabalho se inseriu no contexto do ensino de programação de computadores, especificamente, atuando em um tópico que apresenta sérios desafios, o acompanhamento contínuo da aquisição de novas habilidades por parte dos alunos.

Frente às metodologias tradicionais, leia-se aplicação de questionários e correção manual de exercícios, a pesquisa apresentou uma alternativa baseada na localização automática de evidências, seguida da aplicação de informações em um modelo do aprendiz. Além disso, foi demonstrada a possibilidade do acompanhamento contínuo do progresso do estudante por meio da coleta de evidências a cada submissão de código-fonte.

A viabilidade método foi avaliada com o uso de experimentos em cenários controlados e reais, os quais atingiram taxas de acerto de 92,39% e 95,2%, respectivamente. O método se mostrou promissor, considerando que os índices de acerto dos mecanismos indicam boa confiabilidade na detecção automática de evidências. O agrupamento e filtro evidências se mostrou eficiente, proporcionando a visualização instantânea de diferentes estados do modelo.

Com isso, conclui-se que a detecção automática de evidências é uma possibilidade viável para a inferência de conhecimento no domínio da programação de computadores. A ampliação de pesquisas nesse tema pode trazer benefícios e enriquecer as funcionalidades dos atuais STI, fornecendo formas alternativas de visualização dos conhecimentos dos alunos e, consequentemente, facilitando o acompanhamento do progresso dos alunos em turmas muito grandes.

Ademais, destacam-se algumas possibilidades de expansão do trabalho: elaboração de mecanismos para outros conceitos da área de programação de computadores; integração com STI, a fim de proporcionar avaliação de evidências em tempo real; refinamento dos pesos atribuídos em cada estratégia, de modo que evidências mais relevantes tenham mais influência na ativação de cada nodo.

Referências

Aleman, J. L. F. (2011). Automated assessment in a programming tools course. *IEEE Transactions on Education*, 54(4):576–581.

Buyrukoglu, S., Batmaz, F., and Lock, R. (2016). Semi-automatic assessment approach to programming code for novice students.

Galasso, R. H. and Moreira, B. G. (2014). Integração do ambiente BOCA com o ambiente Moodle para avaliação automática de algoritmos. *Anais do Computer on the Beach*, pages 22–31.

Johnson, W. L. and Soloway, E. (1985). Proust: Knowledge-based program understanding. *IEEE Transactions on Software Engineering*, (3):267–275.

Koedinger, K. R., McLaughlin, E. A., and Stamper, J. C. (2012). Automated student model improvement. *International Educational Data Mining Society*.

Kutzke, A. R. and Direne, A. I. (2015). *FARMA-ALG: An Application for Error Mediation in Computer Programming Skill Acquisition*, pages 690–693. Springer International Publishing, Cham.

Li, N., Cohen, W. W., Koedinger, K. R., and Matsuda, N. (2011). A machine learning approach for automatic student model discovery. In *EDM*, pages 31–40. ERIC.

Maschio, E. (2013). *Modelagem do Processo de Aquisição de Conhecimento Apoiado por Ambientes Inteligentes.* Tese de doutorado, Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná (UFPR).

Pimentel, A. R. and Direne, A. I. (1998). Medidas cognitivas no ensino de programaçao de computadores com sistemas tutores inteligentes. *Revista Brasileira de Informática na Educaçao (IE)*, 3:17–24.

Pimentel, E. P., de França, V. F., Noronha, R. V., and Omar, N. (2003). Avaliação contínua da aprendizagem, das competências e habilidades em programação de computadores. In *Anais do Workshop de Informática na Escola*, volume 1, pages 533–544.

Porfirio, A., Pereira, R., and Maschio, E. (2017). Atualização do modelo do aprendiz de programação de computadores com o uso de parser ast. In *Anais dos Workshops do Congresso Brasileiro de Informática na Educação*, volume 6, page 1121.

Saikkonen, R., Malmi, L., and Korhonen, A. (2001). Fully automatic assessment of programming exercises. In *ACM Sigcse Bulletin*, volume 33, pages 133–136. ACM.

Seffrin, H. and Jaques, P. (2015). Avaliando o conhecimento algébrico dos estudantes através de redes bayesianas dinâmicas. In *Anais do Simpósio Brasileiro de Informática na Educação*, volume 26, page 987.

Vier, J., Gluz, J., and Jaques, P. (2015). Empregando redes bayesianas para modelar automaticamente o conhecimento dos alunos em lógica de programação. *Revista Brasileira de Informática na Educação*, 23(2):45.

Woolf, B. P. (2007). *Building Intelligent Interactive Tutors: Student-centered Strategies for Revolutionizing e-Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.