Modelagem Genérica de Aprendizes com Ênfase em Erros na Aquisição de Habilidades em Programação de Computadores

Andres J. Porfirio[1,2], Alexandre I. Direne[1], Eleandro Maschio[2]

[1]Departamento de Informática – Universidade Federal do Paraná (UFPR)
Caixa Postal 19.081 – 81.531-990 – Centro Politécnico
Curitiba – PR – Brasil

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

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

Abstract. *Computer programming is a major topic for Computer Science students and professionals. This activity presents challenges with respect to monitoring the students' knowledge acquisition over the domain content. The learning progress is defined by principles and skills acquisition. This work proposes an approach for evaluating and monitoring students' skills development in the field of computer programming. The method is based on the mapping of the student's input into expertise features. The setting is organized around a belief model based on the concept of bayesian networks. We expect to provide the human tutor with a partially automated way for monitoring students' learning by means of an intelligent tutoring system coupled to a virtual learning environment.*

Resumo. *O ensino de programação de computadores é um tópico de grande importância na formação de profissionais da Computação. Essa atividade apresenta desafios com relação ao acompanhamento do progresso dos estudantes frente ao conteúdo ministrado. Tal progresso pode ser definido pela aquisição de princípios seguida do desenvolvimento de perícias. Propõe-se nesta pesquisa uma abordagem para avaliação e acompanhamento da aquisição de perícias no domínio da programação de computadores. O método é baseado no mapeamento de entradas do aprendiz em características da perícia. A abordagem inclui conceitos de redes bayesianas. Espera-se, com isso, fornecer ao tutor humano uma via parcialmente automatizada para o monitoramento da aprendizagem por meio de um sistema tutor inteligente acoplado a um ambiente virtual de aprendizagem.*

1. Introdução

1.1. O Problema Central

Dada a importância da programação de computadores na área de Ciência da Computação, muito se tem discutido acerca de técnicas e metodologias alinhadas ao processo de ensino-aprendizagem. Todavia, são raras as metodologias implementadas e testadas que utilizam mais do que os compiladores das linguagens de programação como software de

DOI: 10.5753/cbie.wcbie.2016.1198

apoio central ao aprendizado. Mais raros ainda parecem ser os resultados relatados que revelam ganhos claros de aprendizagem segundo algum parâmetro específico aplicado para avaliar algum software não convencional de apoio à programação (e.g., ganho no tempo de aprendizado, no grau de dificuldade das soluções bem-sucedidas, e outros). A presente pesquisa tem como hipótese principal lidar cientificamente com a modelagem dinâmica de aprendizes através da criação de uma metodologia e ferramentas capazes de: **(a)** identificar a manifestação de múltiplas habilidades de programação por parte do aprendiz; **(b)** monitorar a evolução de tais habilidades ao longo do tempo; **(c)** oferecer acesso a tais registros para fins de auto-monitoramento do aprendiz; **(d)** oferecer o equivalente monitoramento do aprendiz por parte do professor; **(e)** testar empiricamente a eficácia do método e das ferramentas em relação a abordagens convencionais.

O monitoramento do progresso, bem como o acompanhamento da aquisição de novas habilidades por parte dos alunos, é uma tarefa que pode exigir uma grande carga de trabalho dependendo do modo como é executada. O uso de sistemas tutores inteligentes (STIs) favorece o desenvolvimento dessas atividades, proporcionando uma otimização do tempo do professor e, consequentemente, uma redução da sua carga de trabalho quando comparado ao uso de técnicas tradicionais, como a correção manual de exercícios. Destaca-se também que STIs proporcionam maior autonomia aos alunos, possibilitando que dúvidas simples sejam diretamente resolvidas por meio da interação com o software.

Dados os benefícios dos STIs para o ensino de programação, cita-se como parte do objeto de estudo deste trabalho a FARMA-Alg[1], um ambiente de ensino voltado à programação de computadores. Dentre suas principais características, permite-se a construção de objetos de aprendizagem, o gerenciamento de turmas, a aplicação de listas de exercícios, como também o armazenamento e mediação de erros. Acredita-se que esse sistema apresenta as características necessárias para o desenvolvimento e comprovação empírica da hipótese levantada neste trabalho. Acrescenta-se ainda que a capacidade de armazenamento das soluções desenvolvidas pelos alunos contribui de forma significativa para a construção de bases de dados as quais serão de grande importância para o estudo aqui proposto.

Convém salientar também o fator motivacional envolvido no processo de ensino-aprendizagem, pois estudantes motivados tendem a obter maior sucesso na execução das tarefas. A divisão da tarefa principal (aprendizado de programação de computadores) em subtarefas que representam objetivos mais simples (aquisição de perícia em conceitos individuais) é um fator importante para a motivação do aluno e também do tutor. A obtenção de sucesso em pequenas tarefas, em curtos períodos de tempo, é mais motivadora e gratificante se comparada a objetivos longínquos e de difícil alcance. Assim, o apontamento da aquisição gradual de perícias tende a ser benéfico para o sucesso do aluno em seu objetivo principal, mantendo-o motivado durante todo o processo. Destaca-se, além disso, que o benefício se estende ao tutor, que por sua vez consegue observar continuamente os frutos do trabalho desenvolvido.

A dificuldade encontrada pelo tutor quando se trata do acompanhamento da aquisição de perícias (por parte dos alunos) do domínio da programação de computadores é o principal objetivo de estudo desta pequisa. Alia-se a isso, o desafio de manter os

[1]Disponível em: http://ufpr.farma-alg.com.br/

indivíduos motivados durante todo o processo de ensino-aprendizagem.

1.2. Objetivos

O objetivo geral desta pesquisa consiste na proposta de um método de modelagem de aprendiz com base em mapeamento de erros conjugado com a aquisição de perícias no domínio de programação de computadores. Deste modo, fornecendo ao tutor uma forma organizada de visualização do progresso do aluno bem como os erros cometidos e suas possíveis causas. Este objetivo é alcançado por meio do desenvolvimento de uma extensão do STI FARMA-Alg capaz de realizar uma análise de programas incorretos submetidos pelo aluno, procurando detectar a falta de perícia em determinados conceitos. São destacados os seguintes objetivos específicos:

1. Organizar estudos a respeito das perícias no domínio citado, do arcabouço FARMA-Alg e da identificação de erros em códigos fonte;
2. Fornecer um método para classificação de erros com base em perícias, por meio da análise das soluções dos aprendizes;
3. Proporcionar mecanismos de visualização desses erros remetidos às perícias, desenvolvidas e não desenvolvidas, tanto para o tutor quanto para o aprendiz;
4. Apresentar ao tutor recomendações de conceitos a serem reforçados, considerando indícios da falta de perícia por parte dos aprendizes;
5. Otimizar as informações de *feedback* providas ao tutor e ao aprendiz.

A sequência desta proposta é organizada conforme segue: a Seção 2 apresenta uma resenha literária, a Seção 3 esboça a metodologia proposta para o desenvolvimento da pesquisa, na Seção 4 são apresentados os resultados esperados e, por fim, a Seção 5 apresenta as considerações finais.

2. Resenha Literária

Esta seção apresenta um apanhado de conceitos relacionados à realização desta pesquisa. São abordadas as perícias do domínio de programação e os trabalhos relacionados.

2.1. Perícias do Domínio de Programação de Computadores

A construção do conhecimento se dá de forma gradativa, em geral conceitos mais simples são apresentados primeiro e servem como base para o entendimento de conceitos mais complexos. No que se refere à programação de computadores, a assimilação de tais conceitos pode ser entendida como aquisição de princípios e consequente desenvolvimento de perícia [Pimentel and Direne 1998, Maschio 2013].

Segundo [Pimentel and Direne 1998], o aprendizado de programação de computadores é uma tarefa difícil por duas razões: a falta de conhecimento de princípios de programação e a falta de perícia. Com base nisso os autores destacaram as seguintes perícias necessárias para que o aluno se torne um bom programador: **(a)** precisão sintática; **(b)** precisão semântica; **(c)** identificação de estruturas principais no programa fonte (busca por palavra-chave); **(d)** simulação mental dos estados do computador durante a execução; **(e)** catálogo de erros; **(f)** mapeamento mental das estruturas do programa; **(g)** checagem de pré-condições; **(h)** análise do problema; **(i)** integração dos subproblemas; **(j)** generalização da solução; **(k)** reutilização de soluções já conhecidas; **(l)** catálogo de soluções.

As perícias citadas apresentam de forma generalizada um conjunto de conceitos e capacidades exigidas ou desejadas para um programador perito. Estes conceitos foram retomados por [Maschio 2013], que propôs uma extensão desse conjunto de características, o autor destaca as seguintes perícias adicionais: **(m)** velocidade de resolução; **(n)** legibilidade do código escrito; **(o)** otimização de soluções; **(p)** capacidade de depuração; **(q)** definição de casos básicos de teste; **(r)** construção de diálogo adequado de interface; **(s)** autoconhecimento sobre habilidades metacognitivas. O conjunto dessas 19 capacidades passou a ser denominado perícias sobrejacentes de alto nível.

Ademais, [Maschio 2013] apresenta de maneira detalhada um subconjunto de perícias específicas do domínio de programação de computadores, mais especificamente com foco no paradigma imperativista. Este subconjunto conta com 41 categorias de perícias, as quais abrangem diversos conceitos necessários para a formação inicial como programador. O autor elaborou um grafo que apresenta de maneira organizada este subconjunto de perícias. O grafo de perícias destaca as habilidades de forma ordenada, sendo que as capacidades mais simples são apresentadas anteriormente e, conforme se avança pelos nós, são atingidas perícias que exigem maior carga cognitiva e, muitas vezes, fazem uso de conceitos anteriores (pré-requisitos).

As 41 categorias de perícias modelam um subconjunto do conhecimento necessário para que um programador se torne perito naqueles conteúdos. Desta forma, pode-se ter uma noção clara das habilidades iniciais a serem adquiridas pelos estudantes de programação de computadores. Diante disso, o subconjunto de perícias pode ser utilizado como métrica para avaliar o desempenho e o progresso do aluno ao longo do processo de ensino-aprendizagem.

2.2. Análise de Erros em Programas de Computador

A análise de erros em programas de computadores é um tema bastante estudado na literatura. Pesquisas como as de [Osterweil and Fosdick 1976, Ruckert and Halpern 1993, Schorsch 1995] comprovam que este tópico é evidenciado há bastante tempo. Os trabalhos organizados nesta seção serão apresentados com enfoque na maneira como os erros são vistos na perspectiva do aluno.

Nesse ponto de vista, [Stephen and Freund] comentam que os compiladores de linguagens como ANSI C são muito sofisticados e inadequados para iniciantes em programação. Com base nesta premissa, construíram um interpretador de C com enfoque em aprendizes de programação. Dentre as principais características, a ferramenta possui um sistema mais adequado de mensagens de erro, um depurador com interface amigável e, além disso, realiza algumas verificações tais como: **(a)** divisão por zero, **(b)** utilização de variáveis não inicializadas, **(c)** liberação de ponteiros inválidos, **(d)** acesso a posições de memória inexistentes em vetores, **(e)** passagem de argumentos inválidos para funções, **(f)** saída de funções não *void* sem retorno, entre outros.

Com essa mesma premissa, [Hristova et al. 2003] implementaram uma ferramenta para Java que funciona como uma extensão do compilador, atuando no momento em que o erro acontece. A atuação da ferramenta consiste em emitir ao usuário uma mensagem detalhada a respeito do erro associada a possíveis soluções que o aluno pode adotar. Ademais, o trabalho apresenta uma contribuição interessante ao enumerar conjuntos de erros comuns da linguagem Java, destacando-se entre eles: confusão entre o operador de

atribuição "=" e o operador de comparação "=="; desbalanceamento de chaves, colchetes e parênteses; uso incorreto de ponto e vírgula após instruções *if*, *for* e *while*; uso de palavras-reservadas como identificadores de variáveis; chamada de métodos com argumentos incorretos; e, ausência de parênteses em chamadas de métodos.

Em contrapartida, [Hasker 2002] organiza um estudo a respeito de recursos adequados aos programadores iniciantes. Assim como outros autores, o texto destaca que a maioria dos estudantes de programação de computadores adota logo em seu primeiro contato ferramentas de uso comercial, as quais muitas vezes fornecem mensagens destinadas a programadores experientes e que, em virtude disso, mostram-se complexas demais para iniciantes. Os autores desenvolveram um ambiente para ensino de C++ que apresenta a linguagem de programação de forma restrita. Nesse sentido, vários conceitos foram omitidos de modo que o estudante não tenha contato com eles até que compreenda melhor os fundamentos da programação de computadores. Além disso, o ambiente desenvolvido conta com mensagens de compilação que detalham os erros e sugerem possíveis soluções.

Os autores [Sykes and Franek 2004] também argumentam a dificuldade encontrada por estudantes de programação. Nesse trabalho, é apresentado um algoritmo para correção de programas Java, com o objetivo de detectar e sugerir correções de erros de léxice e sintaxe no código fonte do aluno. A primeira etapa do processo consiste na identificação de sugestões de correção com base em um mecanismo de comparação de palavras chave, por exemplo: se o aluno insere erroneamente um tipo de dado "Int" ou "iint", o mecanismo sugere a correção para a palavra chave mais próxima, no caso "int". A segunda etapa consiste em apresentar ao aluno as palavras chave mais prováveis para que o mesmo selecione uma possível solução.

Juntamente com a análise de erros, destacam-se pesquisas que tratam da remediação dos mesmos. Neste sentido diversas abordagens têm sido utilizadas, um exemplo interessante pode ser observado na pesquisa de [Hartmann et al. 2010] que possui como enfoque a remediação de erros por meio de sugestões de conduta para o programador. As sugestões são exibidas a partir de uma base de dados colaborativa, assim quando um programador comete um erro são apresentadas soluções aplicadas por outros usuários que tiveram o mesmo problema no passado.

Também tratando de análise de erros, [Kiran and Moudgalya 2015] tentam prever o comportamento e as falhas do aluno com base nos códigos desenvolvidos pelo mesmo. Os autores desenvolveram uma interface web para programação C++, Java e Python, onde alguns conceitos foram avaliados, tais como: classes, métodos, herança, polimorfismo, encapsulamento e construtores. Uma quantia significativa de estudantes foi submetida ao uso do sistema, os erros gerados durante o processo foram armazenados em um banco de dados, a partir disto o estudo consistiu em uma análise dos *logs* de erro com a finalidade de mapear os conceitos de programação adquiridos pelos participantes. Também é interessante citar que o sistema possibilita uma análise da sequência de passos adotada pelo aluno até atingir uma solução (os *logs* são armazenados na ordem em que ocorrem).

Além disso, o conceito de redes bayesianas vem sendo utilizado para mapeamento do conhecimento do aluno em sistemas tutores inteligentes. Uma rede bayesiana é um modelo probabilístico para mapeamento de conhecimento em domínios onde existem dados incertos ou imprecisos [Seffrin and Jaques 2015]. A rede é representada por um grafo

acíclico direcionado onde os nós representam as variáveis (núcleos de conhecimento) e as arestas representam as conexões (tais como dependências analogias) entre elas.

A utilização deste tipo de rede em sistemas tutores é um tópico interessante, pois permite mensurar o quanto um aluno aprendeu sobre um determinado assunto mesmo em um ambiente com muitas incertezas. Alguns trabalhos na atual literatura abordam este tipo de utilização das citadas redes. Dentre eles, [Duijnhoven 2003, Seffrin and Jaques 2015] descrevem a implementação de redes bayesianas para o mapeamento de conhecimento algébrico do aluno. De forma alternativa, [Conati et al. 2002] as utilizam para modelar conhecimentos do modelo do domínio de física newtoniana, assim como [Vier et al. 2015] as empregam em lógica de programação, mas sem que a abordagem inclua um aprofundamento no tratamento de erros cometidos por aprendizes.

Dentre os trabalhos citados, notou-se que as pesquisas foram, em geral, norteadas por mensagens e erros de compilação. Alguns tentam modelar o conhecimento do aluno com o uso de redes bayesianas. No geral, é possível perceber que existem benefícios nestas abordagens, entretanto o mapeamento de erros conjugado com a aquisição de perícias é uma alternativa ainda não explorada. Destaca-se ainda que a maioria dos trabalhos busca transformar as mensagens de compilação em artefatos mais humanizados pela computação de dados afetivos que tentam dar informações relevantes para proporcionar a solução de eventuais problemas no código. Porém, apesar da forma como os compiladores apresentam as informações ser um tópico importante para tornar os softwares educacionais mais adequados ao ensino de programação, ainda parecem ser tímidas as iniciativas que tentam expandir o campo científico do tratamento automático de erros de lógica.

Ademais, é importante salientar que a grande maioria dos trabalhos têm como foco a melhoria do *feedback* fornecido ao estudante, o que é de inegável importância, entretanto, notou-se um fraco enfoque no mapeamento destes erros. Assim, evidencia-se um espaço de contribuição no que se refere a métodos e técnicas que aprimorem tanto o feedback para o aluno quanto o mapeamento dessas ocorrências ao tutor.

3. Metodologia de Solução

As seções anteriores apresentaram conceitos fundamentais para o entendimento desta proposta de pesquisa, nesta seção é apresentada a metodologia de trabalho.

A primeira etapa do trabalho consiste em um levantamento de informações seguido de uma documentação do subconjunto de 41 perícias apontadas por [Maschio 2013]. As perícias foram apresentadas na tese de doutorado do autor, onde foram organizadas como uma sequência de itens em conjunto com um grafo. A Figura 1 apresenta um fragmento desse grafo. Nele, pode-se observar um exemplo de organização de pré-requisitos e de perícias pertencentes ao subdomínio de estruturas de repetição, que são **(1)** estruturas condicionais; **(2)** habilidade para correção de laços infinitos; **(3)** entendimento das condições de contorno em generalizações de laços condicionais e laços de contagem.

Uma vez organizadas as perícias, o objetivo passa a ser a elaboração de um estudo detalhado de cada uma delas a fim de detectar quais delas podem ser identificadas automaticamente a partir do código fonte enviado pelo aluno. A classificação das perícias é de grande importância dado que algumas características são fortemente ligadas ao raciocínio

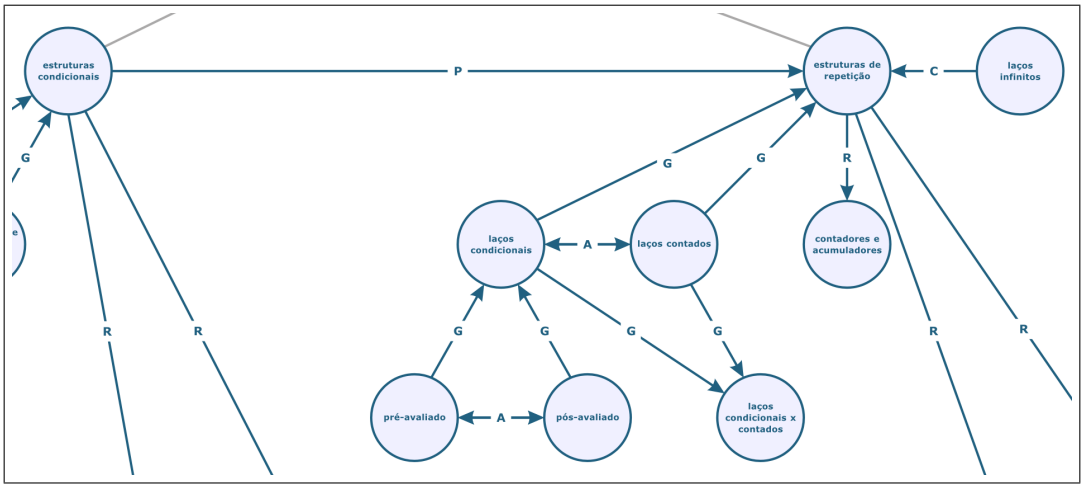

Figura 1. Fragmento do Grafo de Perícias [Maschio 2013].

lógico e percepção do estudante, sendo de difícil (ou até mesmo de impraticável) detecção por meio de heurísticas e algoritmos de Inteligência Artificial da atualidade. Toma-se como exemplo duas perícias distintas:

- Impossibilidade de tratar tipos incontáveis, pertencente à categoria que diz respeito ao domínio sobre a estrutura condicional de seleção múltipla; e
- Distinção conceitual e pragmática entre as estruturas condicionais da linguagem utilizada.

Analisando do ponto de vista de implementação, as habilidades citadas diferem-se em um aspecto muito importante: a maneira de avaliação das capacidades do aluno quanto ao domínio ou não da perícia. A primeira habilidade diz respeito ao uso de estruturas condicionais de múltipla escolha, mais especificamente tratando da variável a ser utilizada como parâmetro de comparação para cada caso da estrutura. Em diversas linguagens, como C e Java, têm-se limitações quanto aos tipos de dados suportados por esta estrutura condicional, sendo assim, o aluno precisa ter conhecimento destes detalhes e evitar o uso de tipos incompatíveis. Pode-se realizar a avaliação de domínio desta perícia por meio da verificação do tipo de dado da variável utilizada como parâmetro, assim sendo possível assumir que um aluno que utiliza sempre os tipos de dados corretos possui domínio sobre a perícia citada.

A segunda perícia apontada diz respeito à capacidade do aluno de avaliar qual a melhor estrutura condicional para um determinado problema. A avaliação deste tipo de característica depende de vários fatores, como por exemplo o tipo de problema que está sendo abordado e até mesmo o tipo de situação em que a estrutura foi utilizada, por exemplo: o aluno pode fazer uso de vários tipos de estruturas condicionais em situações diferentes na resolução de um mesmo problema, assim sendo necessária uma análise detalhada de cada ocorrência de condicionais no código a fim de determinar se o estudante é perito ou não. Diante disso, percebe-se que perícias com este tipo de característica não são tão simples de serem analisadas de forma automática (por meio de heurísticas ou sistemas inteligentes).

Com base nos conceitos e exemplos citados, percebe-se que a análise de diferentes tipos de perícias exigem abordagens diferentes, sendo que algumas podem ser automati-

camente analisadas enquanto que outras apresentam maior desafio e podem exigir intervenção humana. Justifica-se com esses argumentos a realização de uma etapa de análise e ordenação das perícias de modo definir aquelas que possam ser avaliadas automaticamente com base no código fonte escrito pelo aluno e que possam gerar e apresentar ao tutor uma visão detalhada das habilidades adquiridas e necessárias para o aluno.

Ademais, dado o grande número de perícias (41) do subconjunto citado, cabe salientar a possibilidade de seleção de um terceiro subconjunto de perícias para a prova da hipótese deste trabalho. Considerando-se que o grupo de perícias com potencial de identificação automática seja muito grande, é importante deixar clara a possibilidade de implementação de uma parcela deste grupo, de modo que o método proposto seja comprovado.

Uma vez ordenadas as perícias, terá início a etapa de implementação de algoritmos com a finalidade de identificar, com base em uma análise automática do código fonte do aluno, trechos de códigos, sequências lógicas e/ou comportamentos indesejados (podendo ser utilizados como ponto de partida os erros gerados pelo compilador). A detecção destas características proporcionará um meio de apontamento de deficiências de conceitos por parte do aluno em determinadas situações das tarefas de programação de computadores. Deste modo, poderão ser exibidos indícios da falta de perícia em determinados assuntos. Considera-se o uso de técnicas de Inteligência Artificial nesta etapa.

A implementação dos algoritmos citados anteriormente deverá ser feita de modo a facilitar sua integração com o arcabouço FARMA-Alg. Portanto, considera-se o uso da linguagem de programação Ruby (já utilizada na construção da ferramenta). Além disso, destaca-se que é bem vindo o uso de *frameworks* e bibliotecas (caso existam) que auxiliem na análise de códigos fonte e apontamentos de erros nos mesmos. Para a construção dos algoritmos citados, considera-se realizar um estudo sobre técnicas de análise automática de códigos fonte, dentre os principais tópicos a serem abordados cita-se: a identificação de trabalhos que realizem tarefas similares; a identificação e compreensão de técnicas e métodos para a localização de erros em programas de computadores; bem como técnicas e métodos para classificação destes erros.

Posteriormente à etapa de detecção das características dos erros, é previsto o desenvolvimento de um método de mapeamento destas características no grafo de perícias, fornecendo assim uma visão detalhada dos conceitos já adquiridos bem como dos ausentes. Vale ressaltar que o mapeamento das perícias será baseado em indícios apontados pelos algoritmos de detecção de erros, assim sendo esta etapa totalmente dependente do sucesso da anterior.

Ademais, considera-se o mapeamento do conhecimento do aluno com o uso de redes bayesianas, de modo que os nodos da rede representam as perícias e suas dependências. Tais nodos podem ser valorados de acordo com indícios apontados pelos algoritmos de detecção de erros/sucessos. Ainda tratando disso, as arestas direcionadas da rede simbolizam as dependências entre os conceitos. Acredita-se que o uso deste tipo de rede pode servir como modelo de representação das crenças, ou seja, fornecer um meio do sistema tutor mensurar quanto o aluno está absorvendo do conhecimento e, consequentemente, medir seu progresso ao longo do estudo.

Por fim, prevê-se uma etapa de testes. Os testes do método serão baseados no

arcabouço FARMA-Alg, o qual já provê armazenamento de respostas de exercícios de programação de computadores, assim, fornecendo a possibilidade de criação de bases de dados (provenientes da aplicação de questionários em turmas de programação). Com o uso de bases de dados pretende-se aplicar o método proposto, identificar e mapear as perícias desenvolvidas e ausentes em cada estudante, sendo possível após esta etapa elaborar uma discussão sobre os resultados do trabalho aqui proposto. Além disso, havendo necessidade, pode-se criar novas bases de dados a partir do uso do arcabouço FARMA-Alg para a aplicação de listas de exercícios em sala de aula (turmas de alunos de programação de computadores).

4. Resultados Esperados

Espera-se, com a implementação desta pesquisa, fornecer aos tutores de disciplinas e conteúdos da área de programação de computadores uma forma alternativa de visualização do desempenho dos alunos e suas possíveis aquisições ou faltas de perícias relacionadas aos conceitos de programação.

Também se espera agilizar o processo de correção e identificação de erros nos códigos fonte desenvolvidos pelos alunos uma vez que se busca automatizar parte do processo. Destaca-se ainda que a detecção de erros possui enfoque em detalhes específicos de conceitos da área de programação de computadores, não ficando limitado apenas à saída textual do compilador e/ou depurador.

Além disso, procura-se fornecer ao tutor um detalhamento da progressão do aluno durante o processo de ensino-aprendizagem por meio do uso de grafos de perícias. Assim fornecendo indícios de deficiências em determinadas áreas ou conceitos bem como apontando características de sucesso na aquisição de perícias quando oportuno.

5. Considerações Finais

O presente artigo foi motivado por dificuldades e desafios presentes no ensino de programação de computadores. Também foram apresentadas pesquisas relacionadas à enumeração e organização de perícias no domínio de programação de computadores. Tais perícias foram citadas como métricas para o acompanhamento da evolução do conhecimento de aprendizes da área. Além disso, trabalhos relacionados à identificação e *feedback* de erros em programas de computadores também foram expostos. Por fim, foi proposta uma metodologia para desenvolvimento de um método capaz de melhorar a forma de visualização dos erros em códigos fonte desenvolvidos pelos alunos, agrupando-os e classificando-os de acordo com o subconjunto de perícias.

Espera-se com isso proporcionar ao tutor uma maneira alternativa de visualização e acompanhamento da situação dos estudantes. Destaca-se que as perícias adquiridas e ausentes poderão ser visualizadas de forma mais clara. Do mesmo modo, acredita-se que o desenvolvimento desta pesquisa trará benefícios aos estudantes, possibilitando um melhor entendimento dos conceitos a serem estudados bem como a sequência em que devem ser abordados.

Referências

Conati, C., Gertner, A., and Vanlehn, K. (2002). Using bayesian networks to manage uncertainty in student modeling. *User modeling and user-adapted interaction*, 12(4):371–417.

Duijnhoven, J. (2003). *Knowledge assessment using bayesian networks: A case study in the domain of algebraic expectation*. PhD thesis, Master's thesis cognitive artificial intelligence, Utrecht University, Utrecht.

Hartmann, B., MacDougall, D., Brandt, J., and Klemmer, S. R. (2010). What would other programmers do: suggesting solutions to error messages. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1019–1028. ACM.

Hasker, R. W. (2002). Hic: a c++ compiler for cs1. *Journal of Computing Sciences in Colleges*, 18(1):56–64.

Hristova, M., Misra, A., Rutter, M., and Mercuri, R. (2003). Identifying and correcting java programming errors for introductory computer science students. In *ACM SIGCSE Bulletin*, volume 35, pages 153–156. ACM.

Kiran, E. L. and Moudgalya, K. M. (2015). Evaluation of programming competency using student error patterns. In *Learning and Teaching in Computing and Engineering (LaTiCE), 2015 International Conference on*, pages 34–41. IEEE.

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).

Osterweil, L. J. and Fosdick, L. D. (1976). Dave—a validation error detection and documentation system for fortran programs. *Software: Practice and Experience*, 6(4):473–486.

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.

Ruckert, M. and Halpern, R. (1993). Educational c. In *ACM SIGCSE Bulletin*, volume 25, pages 6–9. ACM.

Schorsch, T. (1995). Cap: an automated self-assessment tool to check pascal programs for syntax, logic and style errors. In *ACM SIGCSE Bulletin*, volume 27, pages 168–172. 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.

Stephen, N. and Freund, R. an ansi c programming environment designed for introductory use, march 1996. In *ACM SIGCSE Bulletin, Proceedings of the twenty-seventh SIGCSE technical symposium on Computer science education SIGCSE*, volume 96.

Sykes, E. R. and Franek, F. (2004). Presenting jeca: A java error correcting algorithm for the java intelligent tutoring system. In *IASTED International Conference on Advances in Computer Science and Technology, St. Thomas, Virgin Islands, USA*.

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(02):45.