Análise da Complexidade Ciclomática como Apoio ao Processo de Desenvolvimento do Pensamento Algorítmico

Vinícius Ajala, Denilson Rodrigues da Silva, Cristina Paludo Santos, Carlos Oberdan Rolim

Universidade Regional Integrada do Alto Uruguai e das Missões (URI)
98.802-470 – Santo ângelo – RS – Brasil

viniciusajala@outlook.com,
{deniro,paludo}@san.uri.br,oberdan@gmail.com

Abstract. *One of the premises to establish the basis of algorithmic thinking is a concern for the best possible solution for a given problem. This paper proposes a descriptive computational model, which applies cyclomatic complexity to help students develop algorithmic solutions through demonstration of how far was their solution regarding the reference solution considered. As contributions of this study, we can highlight a tool designed from the model proposed with a view to strengthening teaching methods, theory sedimentation and achievement of better results in learning*

Resumo. *Uma das premissas para se estabelecer a base do pensamento algoritmico é a preocupação pela melhor solução possível para determinado problema. O presente trabalho propõe um modelo computacional descritivo que aplica complexidade ciclomática para auxiliar o aluno a desenvolver soluções algorítmicas através da demonstração do quão distante esteve a sua solução com relação à solução considerada referencial. Como contribuições do trabalho pode-se destacar uma ferramenta concebida a partir do modelo proposto com vistas ao fortalecimento dos métodos de ensino e a sedimentação da teoria e obtenção de melhores resultados no aprendizado.*

1. Introdução

Os desafios da Informática na Educação são muitos, inclusive na própria formação de competências e habilidades de acadêmicos de cursos de computação [Bombasar 2015]. Dentre tais desafios destaca-se a constituição do pensamento computacional e algorítmico que, segundo Wing (2014) apresenta-se como uma das mais importantes contribuições da ciência da computação para o mundo, pois permite que os profissionais pensem nos problemas de forma analítica e desenvolvam soluções computacionais para os mais diversos domínios de aplicação.

Embora os princípios do pensamento computacional e algorítmico sejam explorados em diversas disciplinas curriculares na área de Computação [Silva 2015], sua maior dificuldade encontra-se no contato de acadêmicos em estágios iniciais de um curso de computação com o estudo de algoritmos, o raciocínio lógico e a programação. Logo, o desafio está em desenvolver modelos e ferramentas computacionais que auxiliem o processo de aprendizagem desde a fase inicial, contribuindo para despertar no aluno seu espírito crítico e estabelecendo estratégias para organizar as ideias, o

raciocínio e a representação simbólica, a fim de que o aluno incorpore esses hábitos e os utilize na resolução de problemas durante a trajetória acadêmica e, posteriormente, como profissional.

Nesse contexto, o presente trabalho propõe um modelo computacional descritivo que aplica complexidade ciclomática para auxiliar o aluno a desenvolver soluções algorítmicas através da demonstração do quão distante está a sua solução com relação à solução considerada referencial. A elaboração deste modelo computacional busca, de forma didática, tornar o ensino do conteúdo abordado mais prático e abrangente, de forma a despertar o interesse do aluno, o seu espírito de pesquisa e a busca de informações que possam torná-lo um profissional crítico e de opinião sólida. Como contribuições do trabalho pode-se destacar a ferramenta concebida a partir do modelo descritivo com vistas ao fortalecimento dos métodos de ensino e a sedimentação da teoria e obtenção de melhores resultados no aprendizado.

O trabalho está organizado da seguinte forma: a seção 2 apresenta os principais conceitos que nortearam o desenvolvimento do modelo, bem como alguns trabalhos relacionados que serviram de suporte para as decisões de projeto. A seção 3 apresenta o modelo descritivo proposto e a seção 4 expõem as principais características e interfaces da ferramenta concebida a partir do modelo. Por fim, a seção 5 apresenta as considerações finais e perspectivas de trabalhos futuros.

2. O Pensamento Algorítmico e a Complexidade Ciclomática

Um dos maiores desafios que se impõe ao aluno durante o processo de formação acadêmico profissional é o de deixar de ser um mero utilizador ou executor de determinados programas e passar a ser criador. Para tal, antes de ver o mundo como um conglomerado de desafios computacionais é necessário visualizar e assimilar os problemas como uma estrutura que pode ser solucionada, isto é, o aluno precisa desenvolver suas faculdades mentais de forma a tornar-se um indivíduo de pensamento crítico e sólido, para depois então dirimir os problemas da sociedade com sua especialidade. É neste contexto que surge a aptidão que é vista pelos pesquisadores de educação em computação como imprescindível – o pensamento algorítmico [Souza e Lencastre 2014].

Uma das premissas para se estabelecer o pensamento computacional é a preocupação pela melhor solução possível. Esta é a base do pensamento algorítmico que expressa soluções em diferentes passos de forma a encontrar uma saída mais eficiente e eficaz de resolver um problema [Resnick 2012]. O pensamento algorítmico é um ramo do pensamento computacional, sendo uma representação mais abstrata da forma de se pensar. Não é ensinado diretamente na graduação como uma área de estudo, mas sim de forma indireta quando se coloca o aluno em situações de aprendizagem de lógica de programação por meio de fatos do cotidiano, para depois formular uma solução através de procedimentos computacionais. Em outros termos, segundo Cuny (2010) este pensamento uma vez desenvolvido no sujeito, torna-o capaz de identificar problemas e solucioná-los de forma objetiva e eficaz de acordo com as necessidades dispostas.

2.1. Análise da Complexidade Ciclomática

A complexidade ciclomática é uma métrica cujo objetivo é medir a complexidade de determinado código fonte. Quanto maior o seu valor, maior a dificuldade de se entender, modificar e, consequentemente, testar o código fonte [McCabe 1976]. De forma a facilitar a visualização do fluxo de um código fonte, McCabe utiliza o conceito de Grafo de Fluxo de Controle como forma de representação, composto por vértices que fazem referência a blocos ou linhas de código fonte, e arestas que conectam os vértices formando caminhos do fluxo de execução [Allen 1970], identificando quais são os possíveis caminhos de execução de um algoritmo.

O grafo da complexidade ciclomática tem suas bases na teoria dos grafos e oferece uma métrica de software extremamente útil. Existem métodos, tanto analíticos quanto matemáticos para o cálculo da complexidade ciclomática [Pressman 2006]. Os métodos analíticos resultam em uma verificação visual mais detalhada, sendo grande base de apoio à didática. Já, o método matemático retorna um valor preciso, independente de análise visual, sendo o preferido para programas computacionais. Além disso, a complexidade ciclomática é usada para analisar aspectos de manutenibilidade e testabilidade de software. Em relação a manutenibilidade, o valor da complexidade gerado é analisado podendo assumir os valores baixo, médio ou alto, refletindo assim o nível de dificuldade na manutenção. Já quanto a testabilidade, faz-se uso de casos de teste conforme o valor ciclomático obtido [Sommerville 2003].

Neste trabalho, a complexidade ciclomática é empregada de forma inovadora no contexto de aprendizagem algorítmica, servindo de apoio no processo inicial de programação e constituição do pensamento algorítmico. Assim, a complexidade ciclomática como fator de qualidade na aprendizagem é a motivação deste trabalho a fim de iniciar uma proposta educacional de fortalecimento do pensamento algorítmico.

2.2. Trabalhos Relacionados

A complexidade ciclomática pode ser aplicada em diversos contextos. Cunha (2006) fez uso de sua aplicação como um instrumento para auxílio na testabilidade de banco de dados. Para tanto, implementou um sistema que calcula a complexidade ciclomática a partir de um código-fonte PL/SQL do Oracle. Seibt (2011) abordou o cálculo de métrica de software para códigos orientados a objetos codificados na linguagem Delphi. A ferramenta permite calcular métricas de projeto e de construção, como por exemplo, profundidade da árvore de herança. Faêda (2012) desenvolveu a ferramenta ComplexGraph, a fim de auxiliar na etapa de teste de software, em específico o teste unitário, gerando o grafo de fluxo de controle onde é possível identificar a quantidade de caminhos de execução contidos no código-fonte.

O modelo apresentado neste artigo assemelha-se em alguns pontos com os propostos por Cunha (2006), Seibt (2011) e Faêda (2012), principalmente no tratamento de leitura de códigos para cálculo da complexidade ciclomática. Nos três trabalhos o cálculo é efetuado extraindo de cada linha do código-fonte os *tokens*, que são uma sequência de um ou mais caracteres contidos na gramática da linguagem. Para os *tokens* extraídos, são verificados se algum deles são palavras reservadas de estruturas que mudam o fluxo de execução, como por exemplo, um comando *if*. Caso existam estes

tipos de *tokens*, o sistema contabiliza a quantidade de suas ocorrências e retorna esta contagem como resposta do número ciclomático.

Entretanto, o modelo proposto difere dos demais trabalhos quanto ao foco e objetivos finais. Enquanto os citados propuseram ferramentas com o objetivo de mensurar códigos em uma linguagem específica por meio da métrica, o modelo proposto tem como objetivo despertar uma reflexão do aluno por intermédio de uma representação visual de diferentes soluções algorítmicas. Assim, o uso da métrica e seu valor ciclomático não buscam ser fatores decisivos na análise do aluno, mas sim o grafo de fluxo que é gerado.

3. Modelo Descritivo Proposto

Neste estudo a complexidade ciclomática é empregada de uma forma inovadora no contexto da aprendizagem algorítmica, servindo de apoio no processo inicial de programação e constituição do pensamento algorítmico. O trabalho atenta para o uso de indicadores quantitativos abordados para o aluno de forma qualitativa como, por exemplo, arestas e vértices representados por um grafo que visa refletir conceitos de qualidade de software na aprendizagem. Portanto, a complexidade ciclomática como fator de qualidade na aprendizagem é a motivação deste trabalho a fim de iniciar uma proposta educacional de fortalecimento do pensamento algorítmico.

O modelo proposto respalda-se nos princípios de praticidade e apoio à etapa de aprendizagem de programação básica, buscando refletir melhores soluções perante o aluno. De forma geral, a construção do modelo considera a ideia de que o aluno forneça a solução de um desafio a ele dado (algoritmo), que será analisado, considerando que exista uma base de conhecimento onde estarão os desafios e suas respectivas soluções (código-fonte). Após a análise é gerado o grafo ciclomático da solução do aluno e da solução armazenada na base de conhecimento, elucidando assim uma comparação gráfica que permite abstrair os conceitos de complexidade algorítmica.

Além disso, são retornados ao aluno os dois códigos (referencial e o do aluno) e alguns resultados quantitativos como a complexidade ciclomática, número de estruturas de seleção e repetição, que têm o propósito de auxilia na compreensão das distinções, se houver, entre as soluções. A Figura 1 demonstra o ciclo de execução do modelo conforme Diagrama de Atividades da UML.

O modelo foi desenvolvido usando o *Model-view-controller* (MVC) - um padrão de arquitetura de software que separa a representação da informação da interação do aluno com ele. O modelo (*model*) consiste nos dados da aplicação. Uma visão (*view*) pode ser uma saída de representação de dados. É possível ter várias visões do mesmo dado, como um gráfico de barras para gerenciamento e uma visão tabular para contadores. O controlador (*controller*) faz a mediação da entrada, convertendo-a em comandos para o modelo ou visão. Justifica-se o uso deste padrão por questões de reusabilidade de código e separação de conceitos. Cabe ressaltar que o modelo pode ser implementado em uma linguagem estruturada, desde que seja feita uma adaptação das etapas do desenvolvimento e um estudo aprofundando para tratar cada componente do código.

Para a concepção do modelo as seguintes etapas fizeram-se necessárias: (a) definição de uma base de conhecimento; (b) extração de estruturas de desvio de fluxo do código-fonte; (c) definição dos componentes do grafo; (d) definição dos componentes externos para a geração dos grafos; (e) implementação do cálculo do número ciclomático e (f) comparação das soluções com retorno de indicadores ao aluno. Tais etapas compreendem:

a) Base de Conhecimento - Armazena uma coleção de códigos que se constitui em soluções apontadas na literatura como referências para determinados algoritmos. Esses algoritmos serão os mesmos que o aluno deverá solucionar, e partir do principio que a solução esteja correta, isto é, as saídas do algoritmo sejam as desejadas respeitando as limitações impostas pelo enunciado, será feita uma comparação entre as duas soluções.

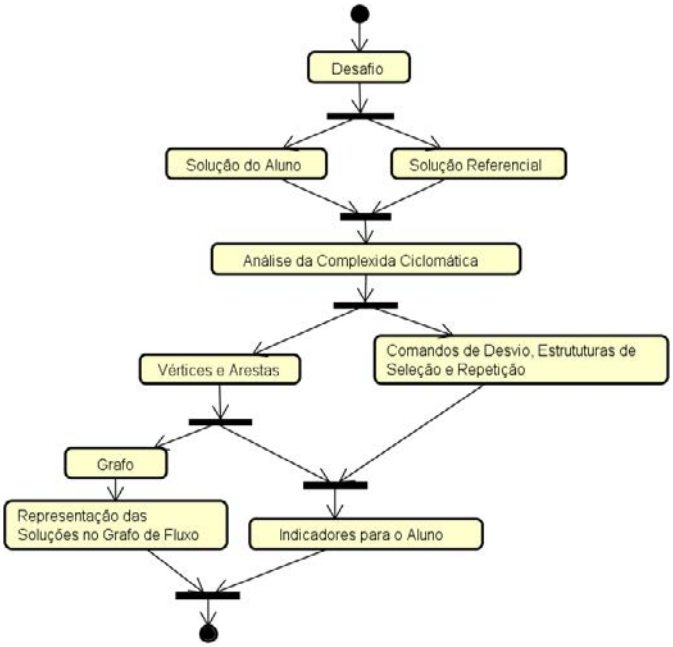

Figura 1. Diagrama de atividades do modelo

b) Extração das Estruturas de Desvio de Fluxo - Para construção do grafo da complexidade ciclomática e da resposta à solução do aluno faz-se necessário identificar nos códigos-fonte (do aluno e do referencial) as estruturas responsáveis por criar desvios de fluxo de execução do código. No modelo proposto foram consideradas as estruturas de repetição, constituídas de comandos como *for*, *while* e *do while*, e as estruturas condicionais, formadas por *if, else* e *case.*

c) Componentes para Geração do Grafo - Um dos requisitos funcionais mais importantes deste modelo é a geração do grafo da complexidade ciclomática. Assim, para geração do grafo utilizam-se os comandos de desvio extraídos do código, que passam a representar os vértices. Alguns comandos geram dois vértices, um com a sua denominação e outro acrescido da palavra *end* para simbolizar o fim da execução do comando de desvio. Os componentes necessários para formar o grafo são os vértices, as arestas e as coordenadas dos vértices. Cada comando de desvio é representado de forma

distinta no grafo, como mostra a Figura 2. No grafo os vértices representam os comandos de desvio e as arestas o fluxo de execução do código-fonte.

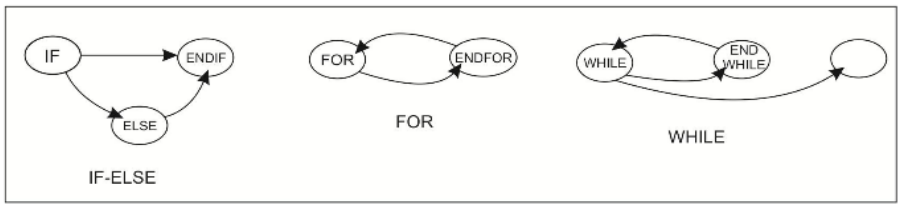

Figura 2. Notação do grafo da complexidade ciclomática

Uma vez encontrados os vértices, é possível definir as arestas do grafo. As arestas utilizadas no grafo são direcionadas, ou seja, representam o sentido do fluxo de execução do código. Além disso, as arestas dizem em quais vértices um determinado vértice irá se conectar. A Figura 3 apresenta a relação entre vértices e arestas, por exemplo, o comando *if* que consta na lista de vértices possui os números 6 e 2, isso significa que o comando irá ligar-se a sua esquerda ao vértice contido na posição 6 da lista e, a sua direita ao vértice da posição 2.

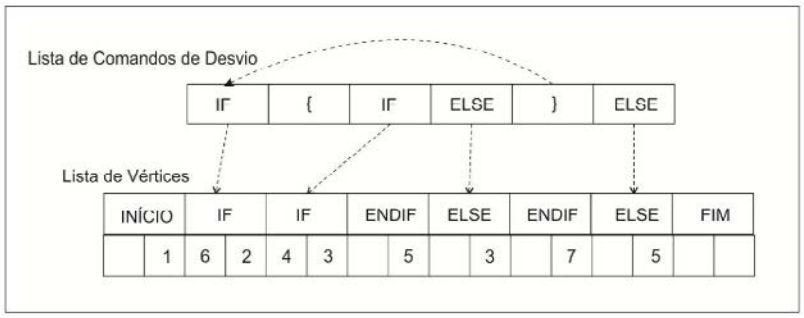

Figura 3. Relação entre vértices e arestas

d) Geração do Grafo - A etapa de geração do grafo da complexidade ciclomática se torna possível em virtude das informações extraídas do código-fonte. Com as informações dos vértices, arestas e as coordenadas x e y, que servem para garantir que um vértice não sobreponha outro, é possível plotar o grafo. O modelo foi concebido para a linguagem Java, uma vez que faz uso de uma biblioteca externa desta linguagem para criar os grafos.

O componente *mxGraph* foi utilizado por este constituir-se em uma biblioteca que permite a criação de forma ágil e prática de diferentes tipos de diagramas. A biblioteca também é eficiente quanto a personalização, permitindo mudar o layout do grafo, sendo possível alterar a fonte, a cor e o tamanho das bordas, a cor de fundo dos vértices, o formato do vértice, entre outros.

e) Cálculo da Complexidade Ciclomática- Para o cálculo da complexidade ciclomática utilizou-se a fórmula: CC = A – V +2 [McCabe 1976], onde CC é o número ciclomático, A é o número de arestas e V é o número de vértices do grafo. Este método foi escolhido por ser uma fórmula matemática, enquanto a maioria dos outros métodos são analíticos, assim facilitando sua implementação em um programa computacional e atendendo plenamente as solicitações do modelo.

f) Indicadores após Análise da Complexidade Ciclomática- Um dos principais objetivos ao desenvolver este modelo foi o de retornar ao aluno um resultado da comparação entre a solução do aluno e a solução referencial. Este resultado nada mais é que um indicador ao aluno que visa abstrair os conceitos de complexidade ciclomática ao nível de aprendizagem, isto é, de forma que o aluno aprenda a buscar uma solução mais eficiente do que a forma habitual de resolução.

3.1. Classes desenvolvidas

O modelo consiste em quatro classes principais: *CalculaCC, GeraEstrutura, AnaliseEstrutura e GeraGrafo*. A classe *GeraEstrutura* é responsável por varrer o código a procura de estruturas de desvio, *AnalisaEstrutura* é responsável por transformar as estruturas em vértices. As outras duas preocupadas na geração dos resultados, *GeraGrafo* implementa o componente que trata a geração dos grafos e *CalculaCC* gera os indicadores ao usuário. Também foram criadas classes intermediárias que auxiliam no processamento de informações, empregadas no pacote *Model*.

Na Figura 4 é representada por intermédio de um diagrama de blocos a estrutura principal do modelo, para as principais classes são detalhadas as instruções que compõem cada etapa O relacionamento entre classe e instrução é representado por uma seta. Na validação deste modelo foi desenvolvida uma ferramenta computacional, usando da estrutura definida anteriormente para gerar os grafos e indicadores. As classes definidas anteriormente foram o ponto de partida do desenvolvimento, outras classes foram concebidas buscando tratar a interação com o usuário.

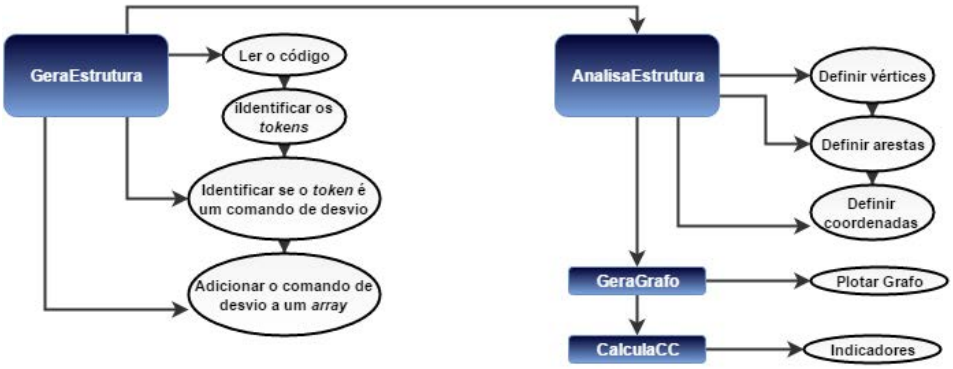

Figura 4. Representação estrutural do modelo

4. A Ferramenta CCEduc

As funcionalidades do modelo proposto foram materializadas na ferramenta CCEduc. Essa ferramenta foi codificada na linguagem Java e desenvolvida conforme o padrão de engenharia de software MVC. A Figura 5 apresenta o diagrama de classes utilizado. Do lado esquerdo estão as classes vinculadas ao Modelo conforme apresentado na seção 3.1 e do lado direito as classes relacionadas as funcionalidades da ferramenta em si.

Para fins de prova de conceito, optou-se por empregar o uso de códigos desenvolvidos na linguagem C, mas pode ser usado, com as devidas adaptações no modelo, para outras linguagens procedurais.

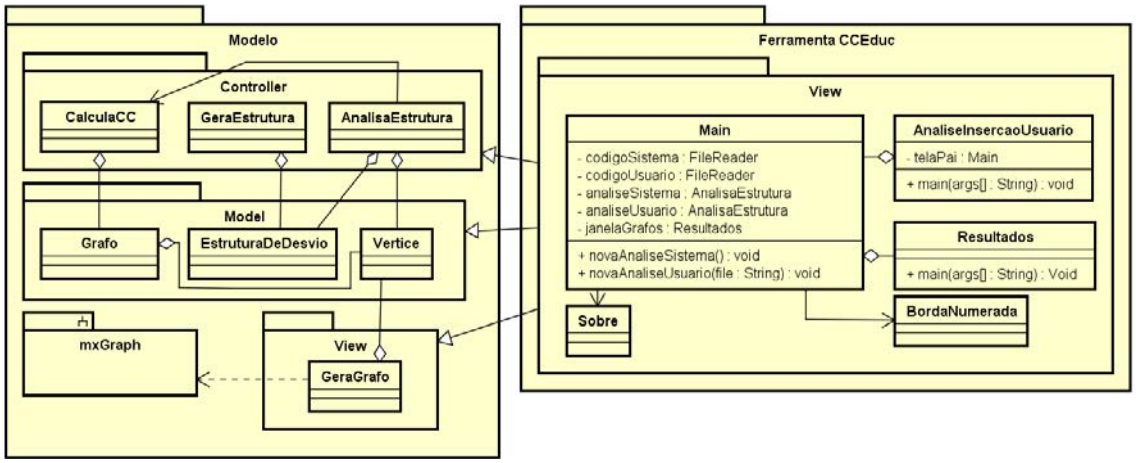

Figura 5. Estrutura de classes da ferramenta CCEduc

A Figura 6 apresenta a interface inicial fornecida para entrada da solução do aluno.

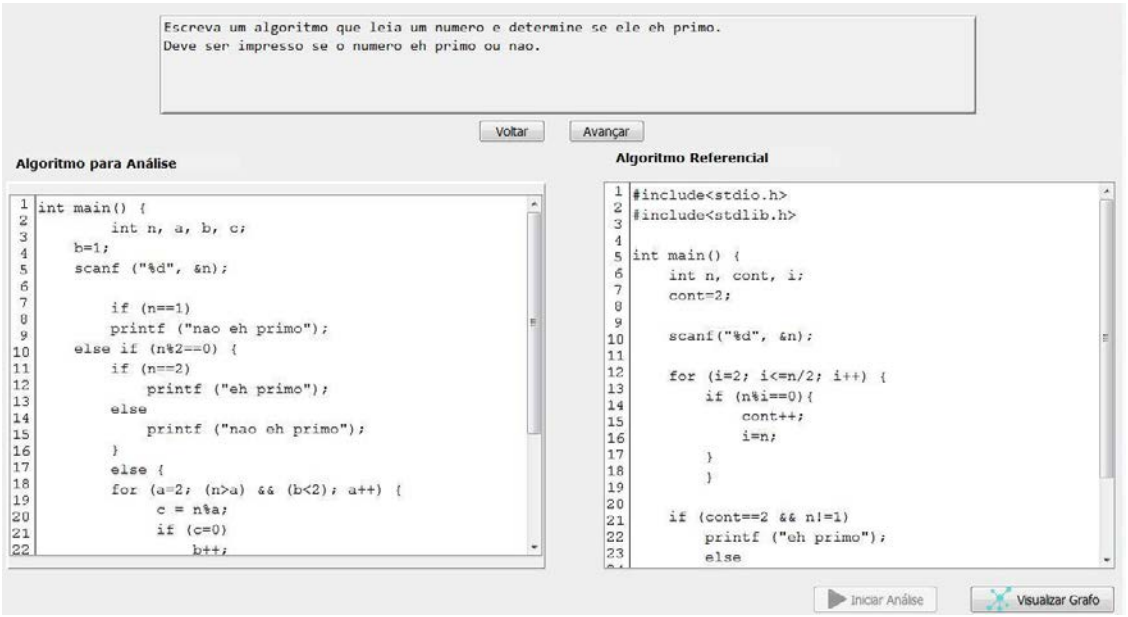

Figura 6. Interface inicial com os algoritmos a serem comparados

Uma vez que o aluno tenha fornecido a sua solução e clique no botão para iniciar a análise é apresentado para ele o algoritmo referencial. Em seguida é habilitado o botão para visualização do grafo de Complexidade Ciclomática. Quando clicado nesse botão é apresentada a tela da Figura 7. Nessa tela podem-se observar os resultados obtidos a partir da análise comparativa entre a solução do aluno e a solução referencial (retirada do livro "Algoritmos: Teoria e Prática", Cormen 2004). A partir da análise da saída da ferramenta pode-se observar que existem diferenças entre a solução proposta pelo aluno e a solução considerada referencial. Ambas alcançam o mesmo objetivo. Porém, a solução proposta pelo aluno demanda maior esforço, usando de quatro comandos de desvio a mais do que a solução referencial.

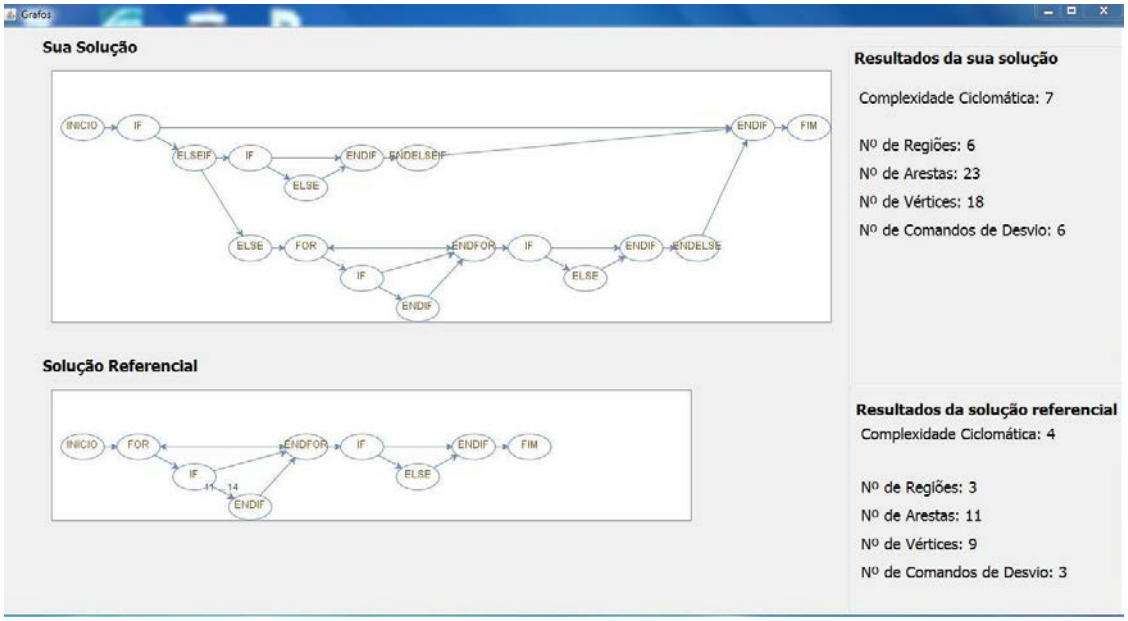

Figura 7. Resultados fornecidos pela ferramenta

Uma das principais vantagens de visualização dos grafos de fluxo e indicadores é facilitar o entendimento do código e identificar os caminhos de execução. Ao comparar as duas soluções o aluno perceberá que a sua solução quebra a execução do fluxo principal e a solução referencial a preserva. Assim os grafos e os indicadores propõem uma reflexão do aluno quanto a sua solução.

5. Considerações Finais

O modelo desenvolvido neste trabalho teve como proposta usar análise da complexidade ciclomática para comparar soluções algorítmicas, buscando uma reflexão maior do aluno iniciante em programação quanto ao seu desenvolvimento de soluções de problemas. Esta reflexão foi o ponto chave desta pesquisa, pois provoca o aluno a analisar por intermédio de dados visuais e quantitativos diferentes soluções, abstraindo conhecimentos avançados em complexidade computacional.

O modelo proposto foi materializado em uma ferramenta chamada de CCEduc. A avaliação dessa ferramenta foi realizada a partir da análise qualitativa em relação a 10 diferentes algoritmos. Em todos os testes executados obtiveram-se informações corretas em relação a análise ciclomática favorecendo o uso da ferramenta em contextos reais de ensino. Para aferir sobre o processo cognitivo da formação do pensamento algorítmico na prática, a ferramenta está sendo disponibilizada para uso de acadêmicos da disciplina de Algoritmos e Programação.

Como trabalhos futuros estão sendo realizadas adaptações em termos de interface visando contemplar recursos de acessibilidade. Além disso, há possibilidades de extensão do modelo de forma a permitir que a comparação ocorra não apenas entre o algoritmo do aluno e o algoritmo referencial, mas também pela comparação entre todos os algoritmos gerados pela turma de aprendizes.

Por fim, uma sugestão interessante seria a criação de um sistema onde não existam soluções referencias, e sim o próprio sistema enumera as melhores soluções submetidas pelos usuários, baseando-se na complexidade ciclomática das soluções. Dessa forma, enfatizando a aprendizagem pelas melhores e não melhores soluções. Esta abordagem usaria da estrutura do modelo apresentado neste trabalho, apenas introduzindo novas funções e buscando objetivos semelhantes. Seria considerada uma evolução do modelo.

Referências Bibliográficas

Allen F. E. (1970). Control flow analysis, Proceedings of a symposium on Compileroptimization, Urbana-Champaign, Illinois, p. 1-19.

Bombasar, James; Raabe, André; Miranda, Elisangela Maschio de; Santiago, Rafael (2015). Ferramentas para o Ensino-Aprendizagem do Pensamento Computacional: onde está Alan Turing? In: Anais do XXVI Simpósio Brasileiro de Informática na Educação (SBIE), pg. 71-80. SBC: Maceio.

Cormen, Thomas H. Cormen; Leiserson, Charles E.; Rivest, Ronald L.; et al. (2004). Algoritmos: Teoria e Prática. Ed: Campus.

Cunha, Karine (2006). Software para Cálculo da Complexidade Ciclomática em Código-fonte PL/SQL. Universidade Regional De Blumenau. Blumenau - SC – Brasil.

Cuny, J., Snyder, L., & Wing, J. (2010). Demystifying Computacional Thinking for Non Computer Scientists. work in progress. EUA.

Faêda, Felippe Moreira (2012). COMPLEXGRAPH: Uma Ferramenta para Geração do Grafo da Complexidade Ciclomática com Foco em Teste de Software. Faculdade Governador Ozanam Coelho – FAGOC. MG – Brasil.

McCabe, Thomas J. (1976). A Complexity Measure. Department of Defense, National Security Agency – EUA.

Pressman, R. (2006). Engenharia de Software. 6. ed. Rio de Janeiro: McGraw-Hill.

Resnick, M. Brennan (2012). New frameworks for studying and assessing the development of computational thinking. American Educational Research Association meeting. Vancouver – BC – Canadá.

Seibt, Patrícia Ramos (2001). Ferramenta para Cálculo de Métrica em Softwares Orientados a Objetos. Universidade Regional de Blumenau. Blumenau – SC – Brasil

Silva, T. R. da, Medeiros, T., Medeiros, H., Lopes, R., & Aranha, E. (2015). Ensino-aprendizagem de programação: uma revisão sistemática da literatura. *Revista Brasileira de Informática na Educação*, *23*(01), 182.

Sommerville (2003), I. Engenharia de software. 6. ed. São Paulo: Addison Wesley.

Souza, J. F. de, & Coelho, S. A. (2015). Uma biblioteca gráfica para aprendizagem de algoritmos e estruturas de dados. *Revista Brasileira de Informática na Educação*, *23*(01), 110.

Wing, J. (2014). Computational Thinking Benefits Society. Social Issues in Computing. New York: Academic Press.