Uso de lógica difusa no Portugol Studio para avaliar a necessidade de ajuda do usuário em programação

Adson M. da S. Esteves1, Alisson S. Henrique1, Augusto C. Pluschkat1

1Ciência da Computação – Universidade do Vale do Itajaí (UNIVALI)  
CEP 88302-202 – Itajaí – SC – Brazil

{shinadson,ali.steffens}@gmail.com, acpluschkat@hotmail.com

**Resumo.** Este artigo tem por objetivo o desenvolvimento de um sistema para avaliar a necessidade de ajuda de um usuário da IDE Portugol Studio enquanto o mesmo programa, utilizando a lógica difusa e formulando regras para defuzzificar variáveis que contam os erros, avisos, número de linhas, de variáveis e funções que o usuário pode fazer no código. Após realizar testes com diferentes códigos com diferentes problemas e complexidades, o sistema foi validado. Ainda há melhoras a serem feitas como uma pesquisa melhor sobre as variáveis utilizadas e seus limites e detalhes de tempo de análise, porém o uso da lógica difusa foi no Portugol Studio foi aceita dentro dos parâmetros estabelecidos.

# 1. Introdução

O Portugol Studio é uma ferramenta em expansão, desenvolvida com o propósito principal de ajudar iniciantes no curso de ciência da computação na compreensão de princípios de lógica e programação. A literatura aponta dificuldades nessa compreensão como base para afirmar que a programação introdutória é um assunto de alta complexidade para se conseguir aprender e de se ensinar [Sheard et al 2009]. As taxas de reprovação nesta disciplina, que podem chegar em até 40%, de acordo com Beauboef e Mason (2005), ajudam a fundamentar essa afirmação.

Estudos têm investigado tais dificuldades sob diferentes aspectos, como os de Castro et al (2002) e de Raabe e Silva (2005). Dado tal problema, muitos alunos iniciantes de cursos de programação apresentam diferentes níveis de dificuldade de aprendizado e por isso é necessária uma maneira de mensurar suas dificuldades para que seja mais simples de identificar alunos que necessitam de maior ajuda no processo de aprendizagem. Portanto, propõe-se uma modificação no Portugol Studio através de uma abordagem por lógica difusa. Pretende-se definir quantitativamente as necessidades de ajuda de cada aluno a partir da coleta de informações obtidas durante seu uso do Portugol Studio.

Este artigo tem como foco apresentar a implementação dessa modificação do Portugol Studio, tendo suas seções organizadas da seguinte forma: a seção 2 explicará a fundamentação teórica que deu base para a construção da solução apresentada; a seção 3 irá descrever um apanhado geral da metodologia aplicada na abordagem do problema; a seção 4 focará em apresentar as particularidade da implementação em detalhes; na seção 5 serão exibidos os resultados obtidos através de testes; e por fim, a seção 6 será focada nas conclusões obtidas com base nos resultados e apresentará sugestões para o que deve ser aperfeiçoado.

# 2. Fundamentação Teórica

Neste capítulo serão discutidos os principais temas de interesse deste artigo, sendo eles: o cenário atual da informática na educação, linguagens de programação para iniciantes e logica difusa.

## 2.1 Neataptic

O ensino de algoritmos e programação é fundamental para as disciplinas iniciais dos cursos de computação como primeira etapa do desenvolvimento do raciocínio lógico e para a prática da programação [RAABE e DA SILVA, 2005]. Eficiência em algoritmos e estruturas de dados são os principais requisitos para a programação. Devido a este fato, tais assuntos estão quase sempre presentes nas matrizes de cursos de Ciência da Computação [SCHEFFLER, 2008].

Porém, de acordo com Lahtien et al (2005), programação não é uma disciplina fácil de aprender, visto a necessidade da correta compreensão de inúmeros conceitos abstratos dos quais sua própria natureza acaba por levar ao aluno, certo grau de dificuldade. Os grandes e heterogêneos grupos de estudantes dificultam a concepção de métodos de ensino genéricos o suficiente para serem benéficos para todos. Além disso, em muitos casos, a falta recursos acaba dificultando a instrução individual e aumentando ainda mais tal heterogeneidade.

Kinnunen e Malmi (2006) reforçam esta dificuldade afirmando que independentemente do fato de ser conhecida a importância em aprender programação introdutória, grande parte das instituições de ensino assumem deficiências em tal aprendizado por parte dos estudantes. Afirmando ainda que um equívoco ou mau entendimento de conceitos básicos referentes a estudos iniciais, dificulta o avanço na continuidade da disciplina, e este mau aprendizado acaba refletindo-se nas instituições por meio das altas taxas de evasão de estudantes nesta matéria.

## 2.2 Redes Neurais Artificiais

Usando neurônios criados artificialmente, a ideia de redes neurais é simular o processo de pensamento do um ser humano, ao fazer com que cada neurônio artificial se comunique uns com os outros enviando sinais, como se fossem sinapses de um cérebro humano normal. Esse sistema foi feito para a identificação de padrões, quando não se sabe exatamente o problema ocorrente, resolvendo-o a partir de uma generalização dos dados com que é treinado.

Geralmente as redes neurais feitas, são multicamadas, onde cada camada de neurônios recebe entradas de uma camada anterior a ela, sendo assim, o sinal original entra pela primeira camada, passando por cada neurônio dela e liberando sinais para camadas inferiores, até se ter uma saída na última camada.

Foi primeiramente proposta e realizada em 1943 quando Warren McCulloch do o Instituto Tecnológico de Massachusetts e Walter Pitts da Universidade de Illinois criaram um modelo de redes neurais com algoritmos e matemática a partir de eletrônicos, chamado de Lógica de limite (Threshold logic), simulando um comportamento de neurônios. A ideia trazia que a saída de um neurônio era uma função da soma de diversas entradas [MACCULLOCH e PITTS, 1943].

Em 1958 Frank Rosenblatt crio o Perceptron, um algoritmo de redes neurais que utilizava apenas uma camada de neurônios[ROSENBLATT, 1985], porém após pesquisas de Marvin Minsky e Seymour Paper em 1969, as pesquisas no campo de redes neurais ficaram estagnadas, pois de acordo com o artigo deles haviam 2 problemas com o processamento de redes neurais, perceptrons básicos não conseguiam processar XOR e computadores da época não tinham poder de processamento suficiente para executar algoritmos de redes neurais.

Assim, Redes Neurais voltou em 1975, com o algoritmo de Backpropagation

Sendo assim, a lógica difusa reflete a maneira como as pessoas pensam, possibilitando a capacidade de modelar o seu senso de palavras, tomada de decisões e senso comum [NORVIG e Russell, 2004]. E por isso, a utilização de tal logica tem sido parte de pesquisas que vem de encontro a sistemas cada vez mais inteligentes e adequados a realidade humana, principalmente em sistemas embarcados por permitir um resultado humanamente aceitável com menos processamento.

Conforme pode ser visualizado na Figura 3, é possível, por exemplo, definir a sensação térmica de uma pessoa além do quente e frio, permeando estados intermediários desta sensibilidade, através de variáveis difusas.

# 3. Metodologia

Definiu-se como foco a coleta de informações geradas ao se programar na plataforma do Portugol Studio para definir a necessidade de ajuda do usuário, sendo gerados a partir dessa coleta um resultado quantitativo através da aplicação da lógica difusa que varia de zero até quatro, variando, portanto, entre nenhuma necessidade até alta necessidade de ajuda.

As variáveis de entrada definidas para serem coletadas foram os erros e avisos gerados durante a programação, que são utilizadas para definir o nível de dificuldade em codificar do usuário; e a quantidade de variáveis, funções e linhas de código, que por sua vez são utilizadas para definir a complexidade do código escrito. Ao definir a complexidade, é possível balancear a quantidade de erros e avisos esperados, pois conforme a complexidade aumenta, é esperado que a dificuldade aumente de maneira proporcional. Portanto, um código menos complexo é esperado que tenha menos problemas como padrão do que um código mais complexo.

Serão realizados 4 testes para validação do sistema, onde 4 códigos em Portugol com diferentes complexidades, erros e avisos serão analisados, e será verificado se o sistema apresenta resultados satisfatórios com as entradas.

# 4. Desenvolvimento

Definiu-se como base da implementação da modelagem da lógica difusa o uso da biblioteca jFuzzyLogic, que é uma biblioteca open source criada em Java, que visa através do uso da Fuzzy Control Language(FCL) especificação IEC 61131 parte 7, implementar e simplificar o uso da lógica difusa em sistemas Java, como é o caso da plataforma alvo da solução proposta, o Portugol Studio.

## 4.1 Captura de Dados de Entrada

O Portugol Studio, possui ferramentas que permitem a obtenção de dados sobre o código do usuário através de sua árvore sintática abstrata, possibilitando quantificar e categorizar os erros e avisos gerados pelo código, assim como contabilizar o número de variáveis e funções geradas. Com base nessas ferramentas, foram obtidas as variáveis de entrada indicadas na Tabela 1.

Tabela 1. Variáveis de entrada e suas descrições



As variáveis de dificuldade são obtidas durante a digitação de um código no editor do Portugol Studio, sendo que a cada 1 segundo de ociosidade, uma compilação de análise do código é realizada, onde tais variáveis são atualizadas. Dentre as variáveis escolhidas, apenas as de erro são cumulativas, pois não é possível executar o código enquanto houver um erro sintático ou semântico, ou seja, o usuário obrigatoriamente tem que resolvê-los para continuar programando. As variáveis de complexidade, por sua vez, são obtidas na compilação do programa.

A cada execução do código, os dados são enviados e calculados no jFuzzyLogic, que retorna os gráficos difusos através de sua API.

## 4.2 Seleção das Regras de Defuzzificação

Para a utilização dos dados dentro da lógica difusa, foi necessário a transformação das variáveis quantitativas em variáveis linguísticas, para isso foram definidos números limites para cada variável linguística através da experiência dos membros especialistas em Portugol da equipe. As variáveis linguísticas foram Nenhum, Baixo e Alto, para erros e avisos, e Poucas, Médias e Muitas para variáveis, funções e linhas (veja Tabela 2 e Tabela 3).

Tabela 2. Variáveis quantitativas de número de Erros e Avisos, com suas respectivas variáveis linguísticas



Tabela 3. Variáveis quantitativas de número de Variáveis, Funções e Linhas com suas respectivas variáveis linguísticas.



Houve a necessidade da criação de cinco tabelas para definição das regras de inferência difusa por causa do número de variáveis trabalhadas, sendo, portanto, as seguintes tabelas: tabela com variáveis de erros; tabela com variáveis de avisos; tabela de problemas, que utiliza os valores gerados pelas tabelas de erros e avisos; tabela com variáveis de complexidade; e por fim, a tabela de nível de necessidade de ajuda do usuário, que utiliza os resultados das tabelas de complexidade e de problemas.

Tabela 4. Regras de defuzzificação das variáveis linguísticas de Erro Semântico e Sintático



Tabela 5. Regras de defuzzificação das variáveis linguísticas de Aviso de Truncação, Conversão e Ocultação



Tabela 6. Regras de defuzzificação das variáveis linguísticas de Erros e Avisos para variaveis linguísticas de problemas



Tabela 7. Regras de defuzzificação das variáveis linguísticas de Variáveis, Linhas e Funções para variáveis linguísticas de complexidade



Tabela 8. Regras de defuzzificação das variáveis linguísticas de Complexidade e Problemas para variáveis linguísticas de Necessidade



# 5. Testes e Resultados

Foram realizados 4 testes para a validação do sistema, em cada teste, um código em Portugol diferente foi utilizado, como também, todos contêm números diferentes de problemas e complexidade.

Tabela 9. Variáveis de problemas e complexidade do código testado 1



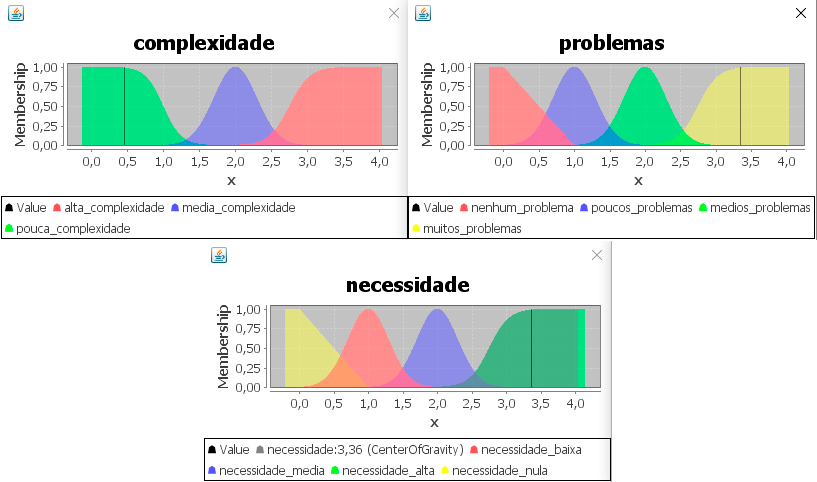


Figura 3. Gráficos gerados pelo jFuzzyLogic a partir do código testado 1

No primeiro teste, foi selecionado um código com baixa complexidade, porém com muitos problemas. O resultado gerado foi que o usuário tem uma alta necessidade de ajuda, pois códigos pouco complexos, não deveriam causar muitos erros.

Tabela 10. Variáveis de problemas e complexidade do código testado 2



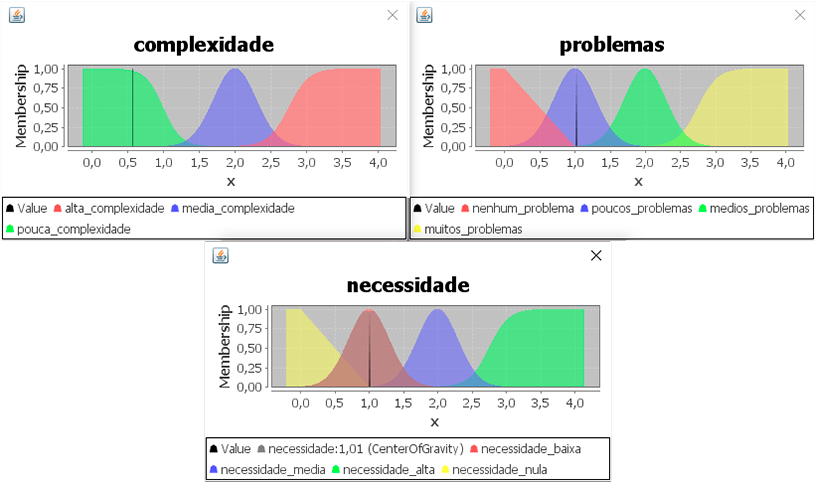


Figura 4. Gráficos gerados pelo jFuzzyLogic a partir do código testado 2

No segundo teste, foi selecionado um código de baixa complexidade, e com poucos problemas também. O resultado gerado foi que o usuário tem baixa necessidade de ajuda, pois códigos em pouco complexos é normal haver poucos erros.

Tabela 11. Variáveis de problemas e complexidade do código testado 3



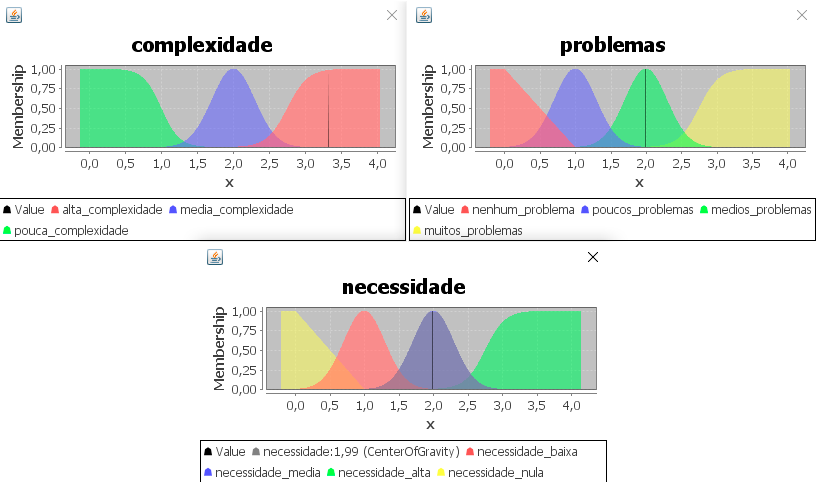


Figura 5. Gráficos gerados pelo jFuzzyLogic a partir do código testado 3

No terceiro teste, um código de alta complexidade, sem erros, porém com muitos avisos, foi selecionado. O resultado gerado foi que o usuário tem necessidade média de ajuda, pois avisos não comprometem a execução do código, mas ainda significam problemas, como também, em um código de alta complexidade, é normal ocorrer mais erros e avisos que códigos menos complexos.

Tabela 12. Variáveis de problemas e complexidade do código testado 4



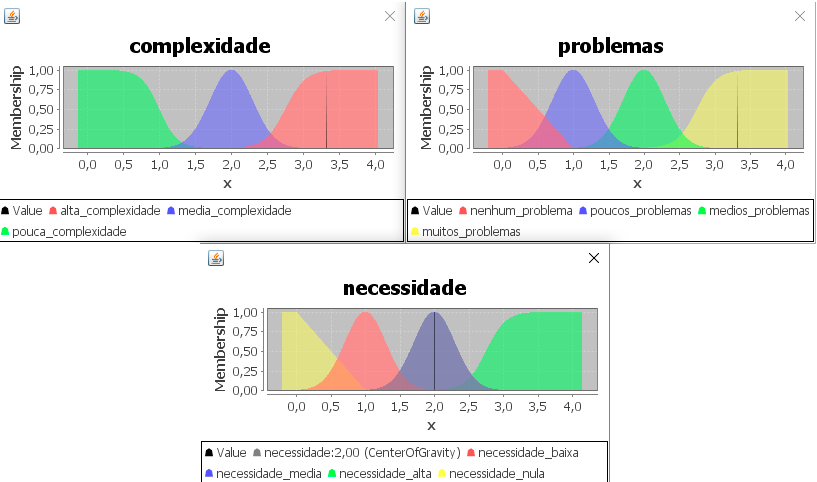


Figura 6. Gráficos gerados pelo jFuzzyLogic a partir do código testado 4

No quarto código testado, um código com alta complexidade, muitos erros, porém sem avisos, foi selecionado. O resultado gerado foi que o usuário tem necessidade média de ajuda, pois mesmo com muitos erros no código, o código contém alta complexidade, então é normal conter mais erro, além de que, um pequeno erro sintático pode ocasionar diversos outros erros por todo o código.

# 6. Conclusões

A partir dos testes, obteve-se, variáveis finais quantitativas, satisfatórias com o que se esperava com o artigo. Com os dados que se pode obter com o sistema, pode-se propor uma diferente forma de avaliação de um aluno de programação, aonde em tempo real, as dificuldades de cada aluno poderiam ser analisadas e dar atenção necessária a quem precisa, como também, acompanhar a evolução do aluno, no aprendizado da programação.

Há a possibilidade de melhorias no sistema, a reverificação da transformação das variáveis quantitativas para qualitativas na Tabela 2 e Tabela 3, ou seja, uma pesquisa mais profunda sobre quantos erros/avisos e variáveis/funções/linhas podem significar o quanto o aluno sabe sobre lógica e programação e se pode existir alguma outra variável que pode ajudar a definir a necessidade do aluno.

No Portugol Studio, uma melhora da contagem de informações do código, pois com o atual, pode ocorrer que em códigos mais complexos, um pequeno erro sintático, pode fazer diversas partes do código apresentarem erros, fazendo serem gravados mais erros do que deveria.

Uma reavaliação, do tempo de compilação de analise, após a digitação de algum código no Portugol Studio. Atualmente a cada 1 segundo de ociosidade, gera uma compilação para analise, que atualizaria as variáveis de informações do código, porém pode acontecer, de o usuário fazer uma pausa em meio a sua programação, podendo gerar erros que não foram causados por desconhecimento do usuário sobre a linguagem ou programação.

# Referencias

BEAUBOUEF, Theresa; MASON, John (2005). Why the high attrition rate for computer science students: some thoughts and observations. In: ACM SIGCSE Bulletin, USA, v. 37, n. 2, p. 103-106, jun. 2005.

CASTRO, T. H. C. (2002) et al. “Arquitetura SAAP - Sistema de Apoio à Aprendizagem de Programação”. XXII Congresso da Sociedade Brasileira de Computação, volume 5 - VIII Workshop de Informática na Escola, Florianópolis.

KINNUNEN, P. and MALMI, L. (2006) “Why students drop out CS1 course?” In Proceedings of the Second international Workshop on Computing Education Research (Canterbury, United Kingdom).ICER '06.ACM, New York, NY, 97-108.

LAHTINEN, E.; ALA-MUTKA, K.; JÄRVINEN, H.-M (2005). A study of the difficulties of novice programmers. Annual Sigcse Conference On Innovation And Technology In Computer Science Education, v. 37, n. 3, p. 14–18, Caparica, Portugal.

LOFTI A. Zedah. (1964). Fuzzy Sets. Information and Control. http://www.sciencedirect.com/science/article/pii/S001999586590241X , November.

LOFTI A. Zedah. (1964). A computational approach to fuzzy quantifiers in natural languages. Computers & Mathematics with Applications. http://www.sciencedirect.com/science/article/pii/0898122183900135, November.

MANNES, Paula (2013). Integração do Portugol Core com o Bipide. http://siaibib01.univali.br/pdf/Paula%20Mannes.pdf, November.

NORVIG, Peter; RUSSELL, Stuart Jonathan (2004). Inteligência artificial. editora Campus.

NOSCHANG, Luiz Fernando (2012). Adaptação Do Portugol Core Para Permitir A Integração Com Outras Ferramentas. Universidade do Vale do Itajaí.

NOSCHANG, Luiz Fernando; PELZ, Fillipi; JESUS, Elieser A. de Jesus; RAABE, André L. A.(2014). Uma IDE para Iniciantes em Programação. In: Workshop de Informática na Escola, No prelo.

OLIVEIRA, B. C. d. S.; WANG, M., GIBBONS, J. (2008). The visitor pattern as a reusable, generic, type-safe componente OOPSLA ’08 Proceedings of the 23rd ACM SIGPLAN conference on Object-oriented programming system languages and applications.

PELZ, Fillipi.(2011). Correção Automática de Algorítmos no Ensino Introdutório de Programação. http://lite.acad.univali.br/portugol/resources/pdfs/tra02.pdf, June.

PELZ, Fillipi (2014). Um Gerador de Dicas para Guiar Novatos na Aprendizagem de Programação. http://lite.acad.univali.br/portugol/resources/pdfs/dis01.pdf, February.

RAABE, André L. A.; SILVA, Júlia M. Carvalho da. Um ambiente para atendimento as dificuldades de aprendizagem de algoritmos. In: CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO, 25., São Leopoldo, Rio Grande do Sul. Anais... RS:SBC, 2005.

SCHEFFLER, P. (2008). Teaching Algorithmics – Theory and Pratice. In: Proc. 2nd Intern. Sc. Conf. “Informatics in the Scientific Knowledge.” Varna, pp. 259-269.

SHEARD (2009), Judy et al. Analysis of research into the teaching and learning of programming. In: INTERNATIONAL COMPUTING EDUCATION RESEARCH WORKSHOP, 5., 2009, Berkeley, USA. Proceedings… USA: ACM, 2009. p. 93-104.