Classificação de Códigos C usando medidas de similaridade para apoio ao Ensino em Programação

José Carlos Campana Filho[1], **Elias Oliveira**[1], **Márcia Gonçalves de Oliveira**[1]

[1]Programa de Pós Graduação em Informática
Universidade Federal do Espírito Santo (UFES) – Vitória – ES – Brazil

zeccfilho@gmail.com, elias@acm.org, clickmarcia@gmail.com

Abstract. *The increase in computer programming courses in distance learning platforms (EAD) has generated a lot of questions and codes. This knowledge base is not always organized in a suitable form to be reused. In order to assist teachers in the generation of the database by grouping programming code by theme, or kind, of the problem, we propose an improvement in the classification code approach for the C language based on similarity measures. The contribution of this classification process is the generation of a base of questions associated to the code solutions that can be used as a source for research and for automatic correction of program issues.*

Resumo. *O aumento de cursos de programação de computadores em plataformas de ensino a distância (EAD) tem gerado uma grande quantidade de questões e códigos. Esta base de conhecimento nem sempre está organizada de forma adequada para ser reaproveitada. Com o objetivo de auxiliar professores na geração dessa base, propomos uma melhoria em uma abordagem de classificação de códigos em linguagem C baseada em medidas de similaridade para agrupar códigos de programação por tema ou tipo de problema. A contribuição desse processo de classificação é a geração de uma base de questões com códigos de soluções associados, que pode ser utilizada como fonte de pesquisa ou para correção automática de questões de programação.*

1. Introdução

A motivação desse artigo surgiu a partir de um problema que professores do Programa de Educação Tutorial (PET) da Universidade Federal do Espírito Santo(UFES)[Valentim et al. 2014] enfrentaram.

O PET/UFES possui um legado de atividades cuja soluções foram escritas em códigos C e submetidos por seus alunos usando a plataforma *Moodle*. Todavia, os enunciados das atividades foram desvinculadas de seus códigos ao longo do tempo. Portanto, esses códigos ficaram sem suas respectivas associações aos seus enunciados. Agora, o que queremos é reconstruir essa associação entre códigos e seus respectivos enunciados.

Nossa proposta de solução é a utilização de um processo de classificação desses códigos em que, os enunciados, passam a representar as classes almejadas. Essa estratégia, portanto, visa apoiar nessa tarefa de reconstrução de uma base de dados de cerca de 100 atividades e mais de 3000 códigos que foram submetidos ao longo do programa do PET/UFES.

DOI: 10.5753/cbie.sbie.2016.1156

Esta base de dados gerada pode ser usada como gabarito de questões, servindo de referência para professores em uma correção manual, ou ser usada como dados de treino para tecnologias desenvolvidas que realizam correção automática de questões de programação [Moreira and Favero 2009] e para sistema de apoio à prática assistida de programação por execução em massa e análise de programas [Oliveira et al. 2015].

Para classificarmos os códigos, usamos como base da nossa solução a abordagem de classificação proposta por [Baby et al. 2014], que classifica códigos em linguagem C que tratam a mesma classe de atividade, isto é, agrupa códigos que tratam um mesmo tema (estrutura de árvore, fila, matriz, etc). Assume-se que os códigos podem ser classificados de acordo com a ocorrência de determinados termos, sejam eles identificadores, palavras-chave, operadores, etc. Os códigos são decompostos em vetores de termos e a classificação é baseada no cálculo de similaridades dos vetores dado por um conjunto de 36 métricas de distância.

Como contribuição de melhoria ao processo proposto em [Baby et al. 2014], foi realizada uma análise sintática e semântica [Cha 2007] do conjunto das 36 medidas de similaridade, com o objetivo de reduzir este conjunto, visando melhorar o desempenho do processo.

Este trabalho está organizado conforme a ordem a seguir. Na Seção 2, apresentamos os trabalhos relacionados. Na Seção 3, explicamos a metodologia utilizada para classificar os códigos. Na Seção 4, apresentamos a análise feita nas métricas de similaridade em busca de melhoria de desempenho. Na Seção 5 relatamos os experimentos. Na Seção 6, temos as considerações finais e trabalhos futuros.

2. Trabalhos Relacionados

Muito trabalho já foi feito em relação a similaridade de códigos de programação, seja para detecção de plágio ou mesmo classificação. Muitas ferramentas estão disponíveis para realizar a análise de similaridade entre códigos. Podemos citar o JPlag [Prechelt et al. 2002], MOSS [MOSS 2014] e Sherlock [Pike 2012]. A proposta usada neste artigo se difere destes trabalhos principalmente no tocante ao cálculo de distância entre os arquivos para determinar a similaridade. Enquanto nos trabalhos relacionados o grau de similaridade é dado pelo cálculo de distância de uma medida (cosseno ou euclidiana, normalmente), neste artigo ele é dado pelo cálculo de distâncias de um conjunto de medidas.

JPlag é uma ferramenta que encontra semelhanças entre conjuntos de arquivos de código fonte. JPlag considera a sintaxe da linguagem de programação e estrutura do programa ao compará-los e, portanto, é muito bom para detectar tentativas de disfarce de semelhanças entre os arquivos plagiados. JPlag atualmente suporta Java, C#, C, C++, Scheme e texto de linguagem natural. JPlag já desempenhou papel em casos de propriedade intelectual onde ele tem sido usado com sucesso por peritos.

O *Measure of Software Similarity* (MOSS) é um sistema automático para a determinação de similaridade de programas. A principal aplicação do MOSS foi na detecção de plágio em aulas de programação. Desde o seu desenvolvimento em 1994, MOSS tem sido muito eficaz neste papel. O MOSS é capaz de analisar similaridade de códigos em várias linguagens, entre elas C, C++, Java, C#, Python, Visual Basic, etc.

O Sherlock é um programa que verifica similaridade entre documentos de texto. Ele gera uma assinatura digital (*hash*) para uma palavra ou sequência de palavras e compara para detectar semelhanças. A assinatura digital é um número que representa um conjunto de palavras convertidas em uma série de bits. O Sherlock é implementado em C e possui código aberto, podendo ser aprimorado pela comunidade. Como esta ferramenta compara códigos como documento de texto, é interessante que se faça uma preparação do arquivo do código antes de submeter ao Sherlock.

Um trabalho muito interessante foi o de [Baby et al. 2014], que tratou a análise de similaridade um pouco diferente. A similaridade entre arquivos é determinada por um conjunto de medidas de distância. Por ser um tópico no estudo de similaridade ainda não muito explorado, fizemos nosso trabalho a partir dele. Falaremos mais detalhadamente dessa abordagem na próxima seção. Continuando o trabalho de [Baby et al. 2014], visando uma performance melhor, usamos os conceitos de [Cha 2007] para reduzir o conjunto de medidas utilizadas. Em [Cha 2007] é feita uma análise sintática e semântica de várias medidas de similaridade e elas são agrupadas por estas características. A melhoria nesse processo irá contribuir na geração da base de enunciados e códigos desejada.

3. Metodologia

Nessa seção descrevemos o método usado neste artigo, proposto em [Baby et al. 2014], a figura 1 ilustra todo o processo:

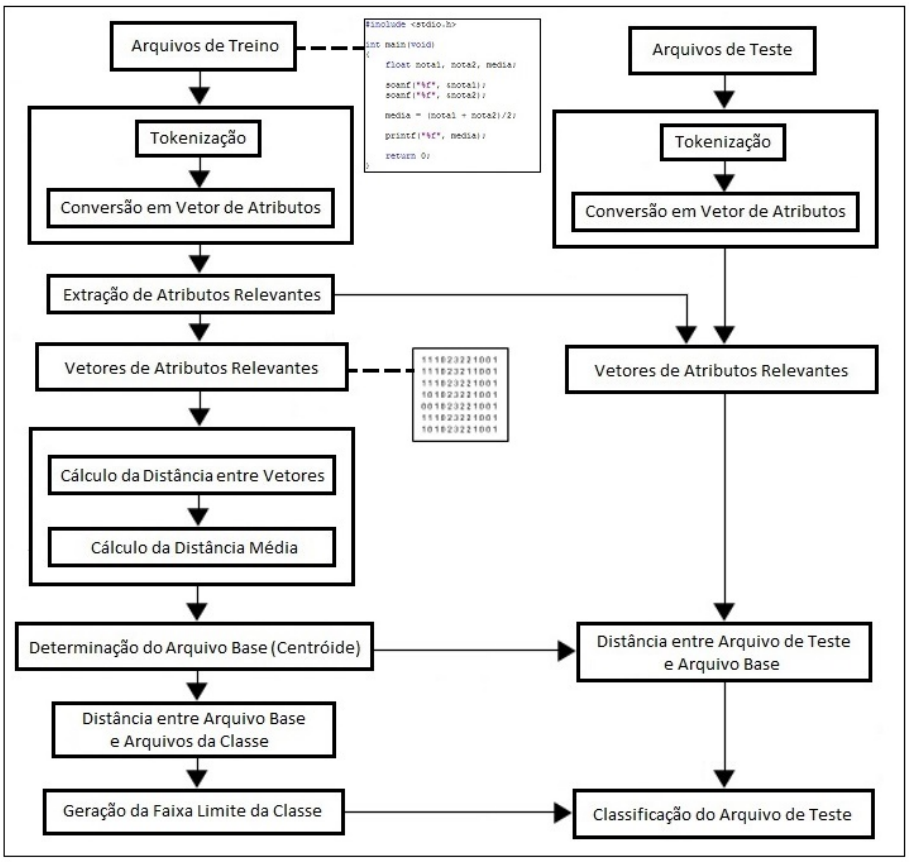

Figura 1. Método proposto

Primeiramente os dados são divididos em treino e teste. O modelo é avaliado comparando as predições com o conjunto de teste.

Uma lista de *tokens* é criada contendo as funções nativas e operadores usados na linguagem C. Cada *token* é associado a um código inteiro único.

Na conversão, o código C é transformado em um vetor de ocorrência de termos. Um termo pode ser uma palavra-chave, um operador, um literal, uma constante, etc. Cada termo é convertido no código inteiro equivalente, e representa uma posição no vetor de frequência. O valor na posição referente ao termo é a sua frequência no código. A tabela 1 mostra um exemplo simples de conversão de código C em vetor de frequência dos termos:

Tabela 1. Exemplo de conversão de código em vetor de frequências de termos

Código	*Token*	Posição do *token* no vetor	Vetor de frequências
	int	1	
	main	2	
int main {	{	3	
int x = 10;	literal (x e y)	4	
int y = x;	=	5	v = {3,1,1,3,2,2,3,1,1}
return 0;	constante (10 e 0)	6	
}	;	7	
	return	8	
	}	9	

No exemplo, os valores do vetor v correspondem a frequência dos termos ”int”, ”main”, ”{”, ”literal (x e y)”, ”=”, ”constante (10 e 0)”, ”;”, ”return”e ”}”respectivamente. A similaridade entre 2 códigos é dada, então, com o cálculo da distância entre 2 vetores, onde *Pi* é o *i-ésimo* termo do primeiro vetor e *Qi* é o *i-ésimo* termo do segundo vetor.

Nem todo termo é necessário para representar as características da classe. Somente os termos presentes em pelo menos 40% dos arquivos da classe são considerados relevantes para representar a classe, os outros são descartados. Em uma classe que contém 26 programas, por exemplo, termos presentes em mais de 40% dos 26 arquivos (ou seja, presentes em mais de 10 arquivos) são selecionados como atributos relevantes.

A distância entre os vetores que representam os arquivos da classe é calculada para encontrar similaridade. Quanto menor a distância significa mais similar os arquivos e vice versa. 36 medidas de distância foram usadas para verificar a similaridade entre os arquivos. A tabela 2 mostra o conjunto de medidas.

Para cada medida é gerada uma matriz de distâncias nXn onde n é o número de programas na classe. Em seguida, é gerada outra matriz nX36 com as médias de distância para cada arquivo e medida.

A partir dessa matriz, é selecionado um arquivo base, centróide da classe, para cada medida. O centróide é o arquivo com a menor distância em determinada medida. O centróide pode ser ou não o mesmo arquivo para cada uma das 36 medidas de distância.

Para gerar a faixa limite da classe para cada medida de distância, é calculada a distância entre os arquivos base de cada medida de distância e os arquivos da família.

Tabela 2. Conjunto das 36 medidas de similaridade usadas na abordagem original

Additive Symmetric	*Average*	*Bhattacharrya*	*Canberra*
Chebyshev	*City Block*	*Clark*	*Czekanowski*
Dice	*Divergence*	*Euclidean*	*Gower*
Harmonic Mean	*Hellinger*	*Intersection*	*Jeffreys*
Jensen Difference	*Jensen Shannon*	*Kdivergence*	*Kulczynski*
KullbackIiebler	*Kumar Johnson*	*Lorentzian*	*Matusita*
Neyman	*Probabilistic Symmetric*	*Ruzicka*	*Soergel*
Sorensen	*Squared*	*Squared Euclidean*	*Squared Chord*
Taneja	*Tanimoto*	*Topsoe*	*Wave Hedges*

Se o número de arquivos que possuem valor de distância menor que a distância média é maior que o número de arquivos com distância maior que a distância média, então a faixa limite é a menor distância até a média, caso contrário, a faixa é a distância média até a maior distância.

Para classificar arquivos de teste, é avaliada a distância entre o centróide e o teste, e se o valor está dentro da faixa limite em mais de 50% das medidas, o arquivo é dito como pertencente à classe.

4. Análise do conjunto de métricas

Para tentar melhorar o desempenho na classificação, focamos em alterar o conjunto de medidas propostas em [Baby et al. 2014]. Para as fórmulas apresentadas nessa seção, considere *d* o número de termos do vetor, *Pi* o *i-ésimo* termo do primeiro vetor e *Qi* o *i-ésimo* termo do segundo vetor.

Observando as 36 métricas utilizadas, e considerando as análises semânticas e sintáticas de [Cha 2007], pode-se perceber que algumas métricas são equivalentes, ou seja, produzem resultados idênticos, como *Soergel* e *Tanimoto*, equações (1) e (2).

$$d_{Soergel} = \frac{\sum_{i=1}^{d} |P_i - Q_i|}{\sum_{i=1}^{d} \max(P_i, Q_i)} \tag{1}$$

$$d_{Tanimoto} = \frac{\sum_{i=1}^{d} \max(P_i, Q_i) - \sum_{i=1}^{d} \min(P_i, Q_i)}{\sum_{i=1}^{d} \max(P_i, Q_i)} \tag{2}$$

Outras métricas são proporcionais, como *City Block* e *Gower*, equações (3) e (4).

$$d_{CityBlock} = \sum_{i=1}^{d} |P_i - Q_i| \tag{3}$$

$$d_{Gower} = \frac{1}{d}\sum_{i=1}^{d}|P_i - Q_i|, \quad d_{Gower} = \frac{1}{d}d_{CityBlock} \tag{4}$$

Ainda temos métricas que são assimétricas, ou seja, a distância entre *P* e *Q*, *d*(*P*,*Q*) é diferente de *d*(*Q*,*P*), como a métrica *Kdivergence*, equação (5). A métrica *Topsoe* é uma versão simétrica da *Kdivergence*, equação (6).

$$d_{Kdivergence} = \sum_{i=1}^{d} P_i \ln \frac{2P_i}{P_i + Q_i} \tag{5}$$

$$d_{Topsoe} = \sum_{i=1}^{d} \left[P_i \ln \left(\frac{2P_i}{P_i + Q_i} \right) + Q_i \ln \left(\frac{2Q_i}{P_i + Q_i} \right) \right] \tag{6}$$

Tanto métricas idênticas como proporcionais não trazem novas informações sobre a similaridade de arquivos, portanto foram escolhidas apenas uma de cada métrica idêntica ou proporcional. As métricas assimétricas foram descartadas, mantendo apenas as suas versões simétricas.

Com isso o número de métricas passou de 36 para 26. Foi adicionada a este novo conjunto a métrica cosseno, pois é bastante citada na literatura [Baeza and Ribeiro 99] e não fazia parte do conjunto inicial. O novo conjunto passou a ter 27 métricas (tabela 3).

Tabela 3. Novo conjunto com 27 medidas de similaridade

Additive Symmetric	*Average*	*Bhattacharrya*	*Canberra*
Chebyshev	*City Block*	*Clark*	-
Dice	*Divergence*	*Euclidean*	-
Harmonic Mean	-	*Intersection*	*Jeffreys*
Jensen Difference	*Jensen Shannon*	-	*Kulczynski*
-	*Kumar Johnson*	*Lorentzian*	*Matusita*
-	-	*Ruzicka*	-
-	*Squared*	*Squared Euclidean*	*Squared Chord*
Taneja	*Tanimoto*	-	*Wave Hedges*
Cosseno			

5. Experimentos

Para o experimento, foram usadas duas bases de dados, a base utilizada em [Baby et al. 2014], e uma base de uma turma de alunos do PET/UFES. O desempenho de classificação foi analisado com a métrica acurácia.

Para a validação do modelo foi usada a técnica de validação cruzada, o método usado foi o *leave-one-out*. O método *leave-one-out* consiste em retirar 1 elemento da base para teste e usar o restante como treino. Este processo é repetido para todos os elementos. No final das iteração calcula-se a acurácia sobre as predições.

5.1. Base do artigo de referência

A base utilizada em [Baby et al. 2014] está dividida em 4 classes, Btree, Crque, Dqueue e Matrix, que agrupam códigos referentes a árvore B, fila circular, fila duplamente encadeada e matriz, respectivamente. Cada classe contém 26 arquivos.

O processo de classificação foi realizado utilizando o conjunto inicial de medidas e depois com o novo conjunto. A tabela 4 mostra o resultado obtido. As predições obtidas para as classes Btree e Matrix foram muito boas. As classes Crque e Dqueue não tiveram a mesma qualidade de predição. Para todas as classes alguns arquivos passaram a ser classificados na classe correta somente quando executado com as 27 medidas, o que nos revela uma melhora na classificação.

Tabela 4. Acurácia obtida com conjuntos de 36 e 27 medidas de similaridade para a base do artigo de referência

Classe	Total de arquivos	Qtde acertos 36 medidas	Qtde acertos 27 medidas	% de acerto 36 medidas	% de acerto 27 medidas
Btree	26	22	23	84.61	88.46
Crque	26	11	11	42.30	42.30
Dqueue	26	11	14	42.30	53.84
Matrix	26	21	22	80.77	84.61
Total	**104**		**Média**	**62.50**	**67.30**

5.2. Base de exercícios do PET/UFES

A base de exercícios do PET/UFES utilizada foi obtida de uma turma de alunos de introdução à programação em linguagem C, do projeto Introcomp [Meneses et al. 2015]. Esta base tem 13 classes, que são atividades, cada uma com quantidade de arquivos variados num total de 591 arquivos.

A tabela 5 mostra a quantidade de arquivos classificados corretamente para os dois conjuntos de medidas para a base do PET.

Ao executar o processo de classificação verificamos que alguns arquivos passaram a ser classificados na classe correta somente quando executado com as 27 medidas, como havia ocorrido na outra base.

Alguns arquivos foram classificados em mais de uma classe. Isso era esperado e se deve ao fato de algumas atividades cobrarem conceitos parecidos de programação. Neste caso, alunos submetem códigos de programas com conjunto de *tokens* parecidos. Situações como essa devem ser tratadas com a supervisão do professor para a indicar a correta classificação.

6. Considerações Finais

Considerando o desempenho da classificação de códigos C dos 2 conjuntos de métricas, vemos que o novo conjunto obteve uma acurácia média de arquivos classificados corretamente melhor que o primeiro, para as duas bases utilizadas. 67.30% contra 62.50% na primeira base e 74.78% contra 71.84% na base do PET. Com isso, conclui-se que, com a

Tabela 5. Acurácia obtida com conjuntos de 36 e 27 medidas de similaridade para a base PET

Classe	Total de arquivos	Qtde acertos 36 medidas	Qtde acertos 27 medidas	% de acerto 36 medidas	% de acerto 27 medidas
Questão 1	22	18	18	81.82	81.82
Questão 2	23	20	19	86.96	82.61
Questão 3	26	21	22	80.77	84.62
Questão 4	31	14	18	45.16	58.06
Questão 5	50	42	44	84.00	88.00
Questão 6	50	38	39	76.00	78.00
Questão 7	52	34	36	65.38	69.23
Questão 8	53	36	35	67.92	66.04
Questão 9	53	40	44	75.47	83.02
Questão 10	56	20	20	35.71	35.71
Questão 11	58	47	47	81.03	81.03
Questão 12	58	39	45	67.24	77.59
Questão 13	59	51	51	86.44	86.44
Total	**591**		**Média**	**71.84**	**74.78**

retirada das medidas idênticas, proporcionais e assimétricas, e a inclusão da medida cosseno, evidenciamos melhor a similaridade entre os códigos, e melhoramos o desempenho do processo.

Com esses resultados, o processo proposto se mostra capaz solucionar o problema tratado nesse artigo, isto é, gerar uma base com uma grande quantidade de códigos associados a enunciados de questões e disponibilizar para professores do PET, ou outros professores com este mesmo problema. Essa base pode ser usada, por exemplo, como um repositório de pesquisa para alunos e professores ou como base de treino para realização de correções automáticas.

Como trabalho futuro, temos a análise de outras medidas e outros pontos que podem ser alterados na abordagem inicial, como a tokenização, extração de atributos relevantes e a faixa de corte das classes. Também como trabalho futuro, pode-se usar esta abordagem de similaridade para detecção de possíveis plágios de códigos.

Referências

Baby, J., Kannan, T., Vinod, P. and Gopal, V. (2014). Distance indices for the detection of similarity in C programs. In *Computation of Power, Energy, Information and Communication (ICCPEIC)*, Chennai. IEEE. DOI:10.1109/ICCPEIC.2014.6915408.

Baeza-Yates R. A., Ribeiro-Neto B. (1999). *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA

Cha, S.-H.(2007). Comprehensive survey on distance/similarity measures between probability density functions. In *International Journal of Mathematical Models And Methods In Applied Sciences*, Volume 1, pp.300-307, 2007.

Meneses, L. F., Mai, L. F. F., Rosario, J., Oliveira, E., Gomes, R. L. (2015). IntroComp: Atraindo alunos do ensino médio para uma instigante experiência com a programação.

In *Anais do XXIII Workshop sobre Educação em Computação (WEI 2015)*, Recife, PE, SBC.

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

MOSS. (2014). *MOSS (Measure Of Software Similarity) plagiarism detection system*. Univ. California, Berkeley. Disponível em "http://theory.stanford.edu/∼aiken/moss/", Acesso em: 21 de março de 2016.

Oliveira, M.;Nogueira, M. A. and Oliveira, E. (2015). Sistema de apoio à prática assistida de programação por execução em massa e análise de programas. In *Anais do XXIII Workshop sobre Educação em Computação (WEI 2015)*, Recife, PE, SBC.

Pike, R. (2012) *The Sherlock Plagiarism Detector*. Disponível em: http://sydney.edu.au/engineering/it/∼scilect/sherlock/, Acesso em: 21 de março de 2016.

Prechelt, L.; Malpohl, G. and Phlippsen, M. (2002). Finding plagiarisms among a set of programs with JPlag. In *J. UCS Journal of Universal Computer Science*.

Valentim, R., Meneses, L., Carvalho, T. M., Penha, R. E., de Oliveira, P., dos Santos, G. A., Rodrigues, L. P., Leite, L. B., Rodrigues, D. P., Gomes, R. L. and Oliveira, E. (2014). Em busca de uma metodologia para a disseminação em massa do ensino de programação. In *Seminário Nacional de Inclusão Digital (SENID)*, Passo Fundo, RS, SBC.