计算机科学
COMPUTER SCIENCE

基于图嵌入的代码相似性度量

梁瑶，谢春丽，王文捷

引用本文

梁瑶，谢春丽，王文捷.基于图嵌入的代码相似性度量[J]．计算机科学, 2022, 49(11A): 211000186-6.

LIANG Yao, XIE Chun-li, WANG Wen-jie. Code Similarity Measurement Based on Graph Embedding[J]. Computer Science, 2022, 49(11A): 211000186-6.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.211000186

基于图嵌入的代码相似性度量

梁 瑶 谢春丽 王文捷
江苏师范大学计算机科学与技术学院 江苏 徐州 221116
(2396769801@qq.com)

摘 要 近年来,代码相似性检测一直是软件工程领域的热点问题,它可以帮助代码克隆检测、代码缺陷预测等,降低软件维护成本。目前流行的代码相似性检测方法大多是借用自然语言处理方法从符号(Token)、抽象语法树(Abstract Syntax Tree,AST)等代码表征中提取源代码的文本、语法、结构等特征信息,将其映射为连续空间的实值向量,然后通过直接计算提取特征的欧氏距离、余弦值,或通过浅层神经网络模型获得代码的相似值,这些方法取得了优于传统程序静态分析的效果。但这些方法大多数是基于源代码语法层面的检测技术,未充分利用源代码的语义信息。Doc2Vec 和 Word2Vec 虽然能够挖掘代码的词汇语义信息,但对代码的执行语义信息无能为力,针对这一问题,提出了使用控制流程图(Control Flow Graph,CFG)来表示代码的执行语义,并使用基于随机游走(Random Walk)的图嵌入方法来学习和推理代码的语义信息,进而判断源代码的功能相似性。实验结果表明,和 Doc2Vec 以及 Word2Vec 方法相比,该模型能够较精确地检测出源代码的功能相似性,其 F1 值相较于 Doc2Vec 和 Word2Vec 方法分别提高了 16.01%和 18.72%。

关键词:控制流程图;图嵌入;随机游走;代码相似性检测
中图法分类号 TP311

Code Similarity Measurement Based on Graph Embedding

LIANG Yao,XIE Chun-li and WANG Wen-jie
School of Computer Science and Technology,Jiangsu Normal University,Xuzhou,Jiangsu 221116,China

Abstract In recent years,code similarity detection has been a hot topic in the field of software engineering,which can help code clone detection,code defect prediction,and reduce the cost of software maintenance. At present,most popular code similarity detection methods build language processing model to extract the text,syntax,structure and other feature information of source code from tokens,AST and other code representations,and map them to real value vectors in continuous space. Then,obtain the similar value of the code comparison by calculating the Euclidean distance and cosine value of the extracted features or by the shallow neural network model. These methods have achieved better results than the traditional static analysis program. However,most of these methods are based on the grammar level of source code,which can not make full use of the semantic information of source code. Although Doc2Vec and Word2Vec can extract the lexical semantic information of code,they are powerless to handle the execution semantic information of code. To solve this problem,control flow graph(CFG) is proposed to represent the execution semantics of code,and the graph embedding method based on random walk is used to learn and reason the semantic information of the code,and then judge the functional similarity of the source code. Compared with Doc2Vec and Word2Vec methods,experimental results show that the model can accurately detect the functional similarity of source code,and its F1 value improves by 16.01% and 18.72% compared with Doc2Vec and Word2Vec methods,respectively.

Keywords Control flow graph,Graph embedding,Random walk,Code similarity detection

1 引言

随着代码开源潮流的发展,GitHub 和阿里云 Code 等开源代码网站存在大量不同程序语言实现的相同功能的源代码。代码相似性检测是度量某一段代码和其他代码段在语法、语义、功能上的相似程度,是代码克隆、代码剽窃等具体任务的技术手段。

基于对源代码的语法信息和语义信息的不同利用情况,现有的代码克隆检测方法可归类为基于文本、词汇、语法和语义 4 个层次。基于文本表征的检测方法有 Dup 方法[1]、Duploc 方法[2]以及 NICAD 方法[3]等,虽然这些方法具有成本低和计算开销小等优点,但仅仅适用于完全克隆类型的简单克隆检测,无法完成复杂类型检测。基于词汇(Token)的检测技术有 CCFinder[4]、D-CCFinder[5]、CP-Miner[6]和 CCLearner[7]等方法,虽然这些方法对源代码信息的利用程度有所提高,但在检测过程中仍然会忽略掉源代码的结构信息。基于语法树的方法和基于指标的方法是基于语法检测技术的主要表征方式。知名的基于树的检测方法有 CloneDR[8],

基金项目:国家自然科学基金(61773185,61877030);江苏省研究生科研与实践创新计划项目(2021XKT1392)
This work was supported by the National Natural Science Foundation of China(61773185,61877030) and Postgraduate Research & Practice Innovation Program of Jiangsu Province(2021XKT1392).
通信作者:谢春丽(6020030132@jsnu.edu.cn)

Deckard[9]和 CDLH[10]等，基于指标的检测方法有 Mayrand 等[11]的工作和 Kontogiannis 等[12]的工作。这些检测方法在提供检测准确率的同时，也带来了不可避免的问题，例如基于树的检测方法由于需要遍历树结构，所以开销较大，而基于指标的检测方法无法保证较高的精确度。

基于语义表征的检测技术与前 3 种检测技术相比，能够在更深入的层次上利用源代码的结构、语法以及一定程度的语义信息。目前基于语义表征的检测技术分别是基于图的检测技术和混合技术。知名的基于图的检测技术有 Komondoor 等提出的方法[13]和 Duplix 方法[14]，知名的混合技术有 ConQAT 方法[15]。近年来，基于语义的深度学习方法被广泛研究与应用，具有代表性的有 Code2Vec[16]，Tree-CNN[17]和 Func2Vec[18]等。从目前的研究结果看，基于语义表征的检测技术对 4 种克隆类型都有较好的检测效果，但也存在一定程度上的缺陷，如 AST 表征方式适用任务的范围较小，难以迁移到其他任务中；构造程序依赖图(Program Dependence Graph，PDG)和同构子图的代价较大，并且随着程序规模的扩大，检测方法的时间复杂度和空间复杂度也在不断上升。为解决这些问题，进一步利用源代码的动态语义信息，本文提出使用控制流程图(CFG)来表示代码的动态语义，然后使用基于随机游走(Random Walk)的图嵌入方法[19]学习并推理代码的语义特征向量，进而判断源代码的功能是否相似。

本文第 2 节介绍代码相似性检测领域的相关研究背景；第 3 节介绍本文提出的方法；第 4 节给出实验及其相关分析讨论；最后总结全文并展望未来。

2 相关工作

2.1 代码相似性

代码相似性检测流程可以分为代码格式转换和相似度确定两个阶段，具体可以细分为代码预处理、中间表示转换、比较单元生成和度量算法匹配 4 个部分，检测流程如图 1 所示。

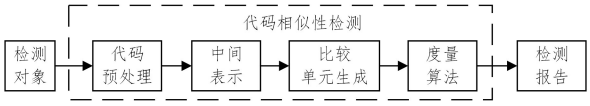


图 1 代码相似性检测流程

Fig. 1 Code similarity detection process

目前，随着深度学习在自然语言和图像识别领域的成功应用，各种神经语言模型被提出，并应用到代码克隆检测领域。文献[20]提出了一种新的基于 AST 的神经网络(AST-NN)。该模型将大型的抽象语法树分割成小的语句树序列，并通过获取语句树的词法句法知识，将语句树编码为对应的向量；然后将这些向量作为输入，采用双向循环神经网络来生成代码片段的向量表示形式；最后使用对称的网络结构，对比两个向量的相似度并得出结论。Wang 等[21]提出了一种模块化的树网络，它根据输入的 AST 动态地将不同的神经网络单元组合成树结构神经网络模型，模块化树网络可以捕获 AST 子结构类型之间的语义差异。实验表明，该模型对 Tpye4 类克隆具有良好的检测结果。Infercode 模型[22]将自监督学习思想从自然语言处理引入到代码的 AST 上，通过预测从 AST 上下文中自动识别的子树来训练代码表示，AST 中的子树被视为训练代码表示的标签，无需人工标记或昂贵的图构造开销表示，不再绑定到任何特定的下游任务或代码单元。将 Infercode 模型应用于代码克隆检测任务上比一般基准有更好的检测效果。

2.2 图嵌入

图嵌入技术实质上是一种图表示学习方法，它将图数据映射为低维稠密向量，同时最大化地保留原始图的结构信息，向量空间中相近距离的节点具有相似的结构信息。表 1 列出了目前图嵌入的所有算法分类。

表 1 图嵌入算法分类

Table 1 Classification of graph embedding algorithm

矩阵分解	随机游走	深度学习	其他
LE	DeepWalk	SDNE	LINE
CGE	Node2vec	DNGR	NEU
Isomap	Metapath2vec	GraphSAGE	CDK
LLE	—	—	DeepCas
GraPep	—	—	TransE
HoPE	—	—	DPMQ

本文采用的图嵌入方法是随机游走算法中的 DeepWalk 算法[23]。该方法将网络中随机游走得到的节点序列看作一个句子，将网络中的节点看作句子中的单词。由于随机游走得到的句子序列组成的语料库和自然语言处理生成的语料库都遵循相似的幂律分布，因此 DeepWalk 算法的实现借鉴了自然语言处理中的 Word2Vec 方法[24]。Word2Vec 方法是用向量表示单词，然后对单词组成的语句进行学习，而 DeepWalk 是利用随机游走思想对图中节点采样组成的节点序列进行学习。

3 基于图嵌入的代码相似性检测模型

由于大多数图分析方法的时间和空间代价都很高，因此本文引入了图嵌入方法，运用图嵌入技术对代码相似性检测进行研究，构造了相应的代码相似性检测模型，如图 2 所示，将实现过程分为预处理层、表征层、相似度计算层和可视化层。下面将详细介绍基于图嵌入的代码相似性模型的实现。

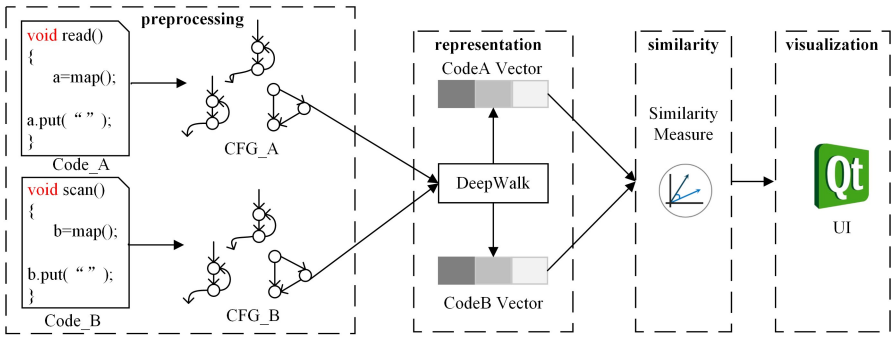


图 2 基于图嵌入的代码相似性度量模型

Fig. 2 Code similarity measurement model based on graph embedding

3.1　预处理层

预处理层的主要功能是对源代码进行处理,以提高检测过程的效率和检测结果的准确率,在本文中是对收集到的C++源代码文件去除无关信息后,将其转换成CFG文件。具体实施过程如算法1所示。算法1描述了从指定源文件到控制流程图的构造过程,基本思想是首先调用parse_file()函数读取源代码文件,接着调用CFGASTVistor()方法遍历AST,访问AST中的文件类型节点、函数类型节点、复合语句类型节点、If语句类型节点、验证函数类型节点以及While语句类型节点,之后调用make_cfg_from_AST()函数生成CFG,最后获取CFG的节点和边,并将其可视化。

算法1　控制流程图生成算法

```
输入:c_file 源代码文件
输出:cfg:控制流程图表示
1. AST=parse_file(c_file)
2. AST_visitor=cfg_AST_visitor. CFGASTVistor(){
      vist()
      vist_FileAST(nodes)
      vist_FuncDef(nodes)
      vist_Compound(nodes)
      vist_If(nodes)
      vist_FuncCall(nodes)
      vist_ While(nodes)
      }
3. entry_nodes=AST_vistor. make_cfg_from_AST(AST)
4. cfg_get_nodes()
5. cfg_get_edges()
6. cfg_txt=save_cfg()
7. cfg_dot=cfg_txt. call_dot()
8. cfg=nx. draw(cfg_get_nodes(),cfg_get_edges())
```

3.2　表征层

表征层是将预处理得到的CFG文件输入到DeepWalk模型中得到代码向量。本模型采用静态分析生成方法构建源程序的CFG,不用编译和运行任何需要分析的代码。

算法2描述了DeepWalk算法的实现过程。该算法主要由两部分组成,分别是随机游走生成和使用Skip-Gram模型学习表达向量。具体可以分为以下4个步骤:

(1)读取:读入代码片段的CFG文件并格式化为后缀名为.gml的文件,然后使用networkx构造有向图。

(2)采样:通过DeepWalk算法对有向图中的节点进行采样,得到一个指定步长的节点序列列表。

(3)训练Skip-Gram模型:将采样得到的节点序列作为Skip-Gram模型的输入,模型的输出是相邻节点的概率,通过最大化相邻节点概率值训练和学习周围节点。

(4)计算嵌入:即得到Skip-Gram隐藏层中每个嵌入节点的向量表示,然后使用加权平均得到代码的向量表示。

算法2　DeepWalk(G,w,d,γ,t)

```
输入:图结构数据为G(V,E);窗口大小为w;节点嵌入向量维度为d;
      步长为γ;总步数为t
输出:节点的向量表示矩阵 Φ=ℝ^{|V|×d}
1. 初始化,从U^{|V|×d}中初始化Φ
2. 从V中建立一个二叉树
3. for i=0 to γ do
4. O=Shuffle(V)
5. for each v_i∈ O do
6. W_vi=RandomWalk(G,v_i,t)
7. SkipGram(Φ,W_vi,w)
8. end for
```

3.3　相似度计算层

本层采用余弦相似度度量代码片段之间的相似值,如式(1)所示:

$$similarity=\frac{\boldsymbol{CodeA\_Vector}\cdot\boldsymbol{CodeB\_Vector}}{|\boldsymbol{CodeA\_Vector}\|\boldsymbol{CodeB\_Vector}|}$$

其中,$\boldsymbol{CodeA_Vector}$与$\boldsymbol{CodeB_Vector}$是两个代码片段$CodeA$和$CodeB$的嵌入向量。

3.4　可视化层

本模型可视化层的图形用户界面设计采用PyQt5框架,设计工具是Qt-Designer。本模型的可视化层UI界面主要功能有3个:

(1)自定义从本地文件夹选择任意两个C++源文件,读取并显示文件内容到代码文本框;

(2)根据步骤(1)上传的源文件显示其文本和图像CFG;

(3)根据步骤(2)生成的CFG调用DeepWalk和余弦相似度得到两个源代码的相似度结果。

4　实验与分析

DeepWalk算法是近年来比较知名的图嵌入方法,为研究DeepWalk算法对代码相似性检测的效果,本文将DeepWalk算法与Word2Vec算法和Doc2Vec算法在同样的实验条件下进行了对比,并对结果进行了分析。本节主要描述了实验中使用的数据集、实验过程和实验结果。下面是对实验的具体描述与分析。

4.1　数据集

本文实验使用了从江苏师范大学Online Judge网站[1)]中收集到的C/C++源代码文件数据集。该数据集约含有9000多个源代码文件,其实现的功能为30种,实现同一个功能的不同源码虽然语法结构不同,但语义是相似的。本文进行的所有实验均以8:2的比例将数据集划分为训练集和测试集。

4.2　实验过程

本文做的所有实验,其过程包含4部分:数据收集、模型构建、结果评估和界面测试。如图3所示。本文实验的代码可从相关网站[2)]获取。

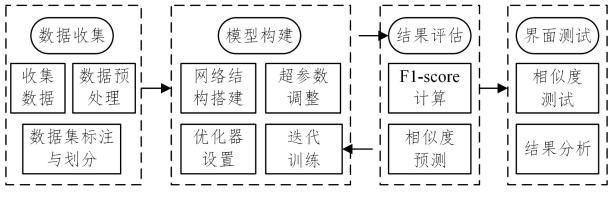


图3　实验实施过程

Fig. 3　Experiment implementation process

4.2.1　数据处理

3个实验的数据处理过程基本一致,首先对源代码进行

1) http://cstlab.jsnu.edu.cn

2) https://github.com/dabaier/code.git

预处理，然后使用 Pycparser 库生成源代码对应的 AST。接着对数据集进行标注与划分，有相似功能的源代码标注为 1，反之标注为 0。本实验源码中使用 data_split()方法划分数据集，其中训练集和测试集的比例为 8:2。

4.2.2 算法描述

(1)DeepWalk 模型

本模型的构建过程为：首先输入经过数据预处理生成的控制流程图，然后使用 DeepWalk 算法随机选取图中的一个节点，对生成的控制流程图进行深度优先遍历以生成随机游走产生的节点序列列表集合，接着将生成的节点序列列表集合放入 Skip-Gram 模型中进行迭代训练，最后保存每张控制流程图所含节点的向量并使用加权平均计算整张图的向量。通过调整参数对模型进行训练得到模型最优结果时，随机游走路径数量(num_walks)设置为 100，随机游走路径长度(walk_lengenth)为 50，嵌入维度(dimension)为 64，滑动窗口大小(window)为 10，词频(min_count)为 1，降采样(sample)为 0.001，迭代次数(iter)为 10。本实验中，由于选取的 Skip-Gram 模型的学习率(learning rate)是变速的，因此初始值设置为 0.025，且模型加速采用分层 Softmax(Hierarchical Softmax)和负样本采样(Negative Sampling)算法。

(2)Word2Vec 模型

本实验模型的构建过程与 DeepWalk 模型构建过程类似。首先输入预处理阶段生成的语料库，然后使用 Word2Vec 训练词向量，并保存每个词的向量，最后使用加权平均计算整张图的向量。当该模型达到最优结果时选取参数的值分别为：嵌入维度为 64，滑动窗口大小为 5，词频为 1，降采样为 0.001。

(3)Doc2Vec 模型

本模型构建过程与上述两个实验模型构建过程类似。首先输入预处理阶段生成的语料库，然后使用 Doc2Vec 训练并保存句向量。当本模型达到最优结果时选取的参数值分别为：嵌入维度为 64，滑动窗口大小为 10，词频为 3，降采样为 0.001，迭代轮次为 20。

4.2.3 模型评估与测试

本文实验进行了两种测试，分别是相似功能测试和不相似功能测试。对于相似功能测试，从收集的数据集中随机抽取 5 个源代码文件，对其进行排列组合后得到 10 组功能相同的测试用例，送入上述的 3 个模型中进行测试，测试结果即相似度对比结果如表 2 所列。对于不相似功能测试，同样从收集到的数据集中随机抽取 5 个实现不同功能的源代码文件，排列组合后得到 10 组功能不相同的测试用例，其相似度对比结果如表 3 所列。从代码相似功能测试实验结果来看，当涉及到源代码文件 Greatestcommondvisior_4.cpp 时，与 Word2Vec 模型结果相比，DeepWalk 模型结果的相似度值较低。通过观察所抽取的源代码发现，这些代码的功能均是求解最大公约数，而源代码文件 Greatestcommondvisior_4.cpp 与其他文件最主要的不同是其对于某个数的初始化赋值没有经过条件判断语句。本文提出的基于图嵌入的代码相似性度量是使用 CFG 来表示代码的执行语义，并使用基于随机游走的图嵌入方法学习和推理代码的语义信息，进而判断源代码的功能相似性，因此源代码文件 Greatestcommon dvisior_4.cpp 的 CFG 与其他代码的 CFG 在某些节点上有所不同，从而导致对这些 CFG 进行表征嵌入，计算出的余弦相似度结果较低。

表 2 代码相似功能测试实验的相似度结果

Table 2 Similarity results of code similarity function test experiment

(单位：%)

序号	源代码 1	源代码 2	DeepWalk 模型	Word2Vec 模型	Doc2Vec 模型
1	Greatestcommondvisior_1.cpp	Greatestcommondvisior_2.cpp	98.93	87.93	45.93
2	Greatestcommondvisior_1.cpp	Greatestcommondvisior_3.cpp	97.78	75.78	37.78
3	Greatestcommondvisior_1.cpp	Greatestcommondvisior_4.cpp	50.59	99.59	42.59
4	Greatestcommondvisior_1.cpp	Greatestcommondvisior_5.cpp	88.85	68.85	66.85
5	Greatestcommondvisior_2.cpp	Greatestcommondvisior_3.cpp	99.16	87.16	29.16
6	Greatestcommondvisior_2.cpp	Greatestcommondvisior_4.cpp	40.78	83.78	46.78
7	Greatestcommondvisior_2.cpp	Greatestcommondvisior_5.cpp	86.99	86.09	39.99
8	Greatestcommondvisior_3.cpp	Greatestcommondvisior_4.cpp	38.03	71.03	38.03
9	Greatestcommondvisior_3.cpp	Greatestcommondvisior_5.cpp	82.96	88.96	48.96
10	Greatestcommondvisior_4.cpp	Greatestcommondvisior_5.cpp	32.09	92.09	45.09

表 3 代码不相似功能测试实验的相似度结果

Table 3 Similarity results of code dissimilar function test experiment

(单位：%)

序号	源代码 1	源代码 2	DeepWalk 模型	Word2Vec 模型	Doc2Vec 模型
1	Greatestcommondvisior_1.cpp	Euclid-1.cpp	16.08	66.08	16.08
2	Greatestcommondvisior_1.cpp	Fibonaccisequence-1.cpp	34.13	54.13	3.13
3	Greatestcommondvisior_1.cpp	Sumofoddsquares-1.cpp	22.04	38.04	23.04
4	Greatestcommondvisior_1.cpp	Factorialsum-1.cpp	39.09	66.09	21.09
5	Euclid-1.cpp	Fibonaccisequence-1.cpp	49.19	74.19	18.19
6	Euclid-1.cpp	Sumofoddsquares-1.cpp	36.61	57.61	14.61
7	Euclid-1.cpp	Factorialsum-1.cpp	66.95	57.95	6.95
8	Fibonaccisequence-1.cpp	Sumofoddsquares-1.cpp	72.83	67.83	4.83
9	Fibonaccisequence-1.cpp	Factorialsum-1.cpp	47.16	40.16	13.16
10	Sumofoddsquares-1.cpp	Factorialsum-1.cpp	39.70	65.70	2.70

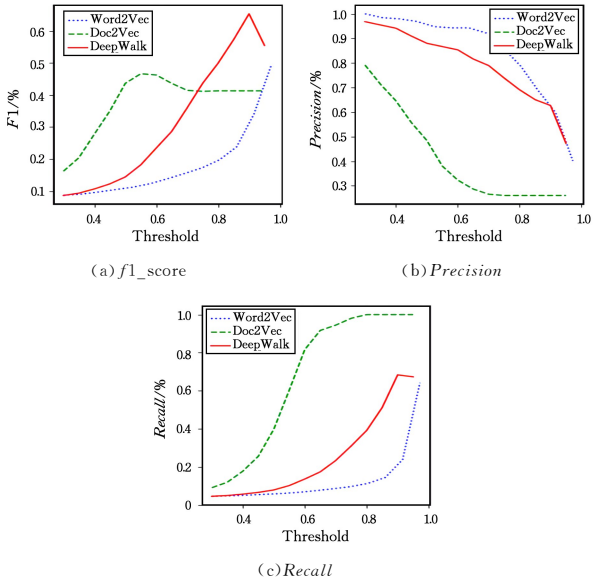
4.3　实验结果与分析

本文3个实验得到的具体评测指标对比结果如表4所列，取得的最优结果对比如图4所示。其中，横坐标threshold为代码相似度阈值。

表4　3个实验指标对比结果

Table 4　Comparison results of 3 experimental indexes

	TP	*TN*	*FP*	*FN*	*Precision*	*Recall*	*F1*
Doc2Vec	146	9520	238	96	0.3802	0.6033	0.4665
Word2Vec	154	9530	230	86	0.4010	0.6417	0.4936
DeepWalk	235	9317	140	109	**0.6267**	**0.6831**	**0.6537**



(a) *f*1_score　(b) *Precision*

(c) *Recall*

图4　各指标随阈值变化的对比

Fig. 4　Comparison of indicators vary with thresholds

从表4中可以看出，DeepWalk算法整体上优于其他两种计算方法，其精确率（Precision）分别提高了24.65%和22.57%，查全率（Recall）分别提高了7.98%和4.14%，F1值分别提高了18.72%和16.01%。为了更加直观和准确地观察3种计算方法的性能对比，我们对相似性阈值设定从0开始按照步长为0.05的大小增加到1，测试结果如图4所示，不同线型所代表的含义见图注。从图中可以看出，当阈值达到0.9左右时，DeepWalk算法的F1值达到最高，虽然之后有所下降，但仍比Word2Vec算法和Doc2Vec算法的F1值最大值更大。从精确率来看，DeepWalk算法前期的表现稍差于Word2Vec算法，但当阈值达到0.9后，两种算法的精确率基本一致，Doc2Vec算法精确率一直小于其他两种算法。而对于查全率来说，Word2Vec算法表现最差，Doc2Vec算法表现最好，DeepWalk算法表现适中。因此，从总体上看，DeepWalk算法更兼顾Precision和Recall值。

结束语　本文首次提出将控制流程图结合基于随机游走的图嵌入方法应用于代码相似性度量任务，通过代码功能相似性实验证明了代码语义信息结合自然语言处理的方法应用在代码相似性度量任务中的优越性，其F1值在65%左右，相较其他方法提高了16%～19%左右，但仍存在一些可能提升模型性能的因素未被考虑：

（1）数据集较小，代码实现功能单一。本文使用的数据集的数量和功能种类相比其他数据集较少，为进一步验证本文提出的方法在大规模数据集中的性能，应继续加大训练数据集，提高本文模型的泛化能力。

（2）源代码的控制流程图生成算法可以继续优化。本文使用的控制流程图是从现有的Pycparser生成的AST结果中遍历获取，由于该AST对某些代码片段考虑粒度较大，后续工作应继续考虑展开这些粒度较大的基本块。

（3）普适性不高。本文仅针对收集到的C/C++源代码进行相似性度量，后续可以考虑其他程序设计语言的静态分析结果，将其处理成可供已有模型输入的规范化中间表示形式。

（4）由于深度学习技术在多个领域取得了优异表现，因此可以考虑基于深度学习的图嵌入算法是否可以在代码相似性度量任务上取得更优的性能。

参 考 文 献

[1] BAKER B S. On finding duplication and near-duplication in large software systems[C]// Proceedings of 2nd Working Conference on Reverse Engineering. IEEE, 1995: 86-95.

[2] DUCASSE S, RIEGER M, DEMEYER S. A language independent approach for detecting duplicated code[C] // Proceedings IEEE International Conference on Software Maintenance-1999 (ICSM'99)[C]// Software Maintenance for Business Change. IEEE, 1999: 109-118.

[3] ROY C K, CORDY J R. NICAD: Accurate detection of near-miss intentional clones using flexible pretty-printing and code normalization[C] // 2008 16th iEEE International Conference on Program Comprehension. IEEE, 2008: 172-181.

[4] KAMIYA T, KUSUMOTO S, INOUE K. CCFinder: A multilinguistic token-based code clone detection system for large scale source code[J]. IEEE Transactions on Software Engineering, 2002, 28(7): 654-670.

[5] LIVIERI S, HIGO Y, MATUSHITA M, et al. Very-large scale code clone analysis and visualization of open source programs using distributed CCFinder: D-CCFinder[C]// 29th International Conference on Software Engineering (ICSE'07). IEEE, 2007: 106-115.

[6] LI Z, LU S, MYAGMAR S, et al. CP-Miner: Finding copy-paste and related bugs in large-scale software code[J]. IEEE Transactions on Software Engineering, 2006, 32(3): 176-192.

[7] LI L, FENG H, ZHUANG W, et al. Cclearner: A deep learning-based clone detection approach[C]// 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME). IEEE, 2017: 249-260.

[8] BAXTER I D, YAHIN A, MOURA L, et al. Clone detection using abstract syntax trees[C]// Proceedings. International Conference on Software Maintenance. IEEE, 1998: 368-377.

[9] JIANG L, MISHERGHI G, SU Z, et al. Deckard: Scalable and accurate tree-based detection of code clones[C]// 29th International Conference on Software Engineering (ICSE'07). IEEE, 2007: 96-105.

[10] WEI H, LI M. Supervised Deep Features for Software Functional Clone Detection by Exploiting Lexical and Syntactical Information in Source Code[C]// IJCAI. 2017: 3034-3040.

[11] MAYRAND J, LEBLANC C, MERLO E M. Experiment on the automatic detection of function clones in a software system using metrics[C] // International Conference on Software Mainte-

nance. IEEE, 1996: 244-253.
[12] KONTOGIANNIS K, GALLER M, DEMORI R. Detecting code similarity using patterns[C]// Working Notes of 3rd Workshop on AI and Software Engineering. 1995: 68-73.
[13] KOMONDOOR R, HORWITZ S. Using slicing to identify duplication in source code[C]// International static analysis symposium. Springer, Berlin, Heidelberg, 2001: 40-56.
[14] KRINKE J. Identifying similar code with program dependence graphs[C]// Proceedings Eighth Working Conference on Reverse Engineering. IEEE, 2001: 301-309.
[15] HUMMEL B, JUERGENS E, HEINEMANN L, et al. Index-based code clone detection: incremental, distributed, scalable [C]// 2010 IEEE International Conference on Software Maintenance. IEEE, 2010: 1-9.
[16] ALON U, ZILBERSTEIN M, LEVY O, et al. code2vec: Learning distributed representations of code[J]. arXiv: 1803, 09473, 2019.
[17] ROY D, PANDA P, ROY K. Tree-cnn: A deep convolutional neural network for lifelong learning[J]. arXiv: 1802. 05800, 2018.
[18] DEFREEZ D, THAKUR A V, RUBIO-GONZÁLEZ C. Path-based function embedding and its application to specification mining[J]. arXiv: 1802. 07779, 2018.
[19] PEROZZI B, AL-RFOU R, SKIENA S. Deepwalk: Online learning of social representations[C]// Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2014: 701-710.
[20] ZHANG J, WANG X, ZHANG H, et al. A novel neural source code representation based on abstract syntax tree[C]// 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE). IEEE, 2019: 783-794.
[21] WANG W, LI G, SHEN S, et al. Modular tree network for source code representation learning[J]. ACM Transactions on Software Engineering and Methodology (TOSEM), 2020, 29(4): 1-23.
[22] BUI N D Q, YU Y, JIANG L. Infercode: Self-supervised learning of code representations by predicting subtrees[C]// 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE). IEEE, 2021: 1186-1197.
[23] LE Q, MIKOLOV T. Distributed representations of sentences and documents[C]// International Conference on Machine Learning. PMLR, 2014: 1188-1196.
[24] MIKOLOV T, SUTSKEVER I, CHEN K, et al. Distributed Representations of Words and Phrases and their Compositionality[C]// Advances in Neural Information Processing Systems. 2013: 3111-3119.
[25] ZHAO G, HUANG J. Deepsim: deep learning code functional similarity[C]// Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2018: 141-151.
[26] WANG W, LI G, MA B, et al. Detecting code clones with graph neural network and flow-augmented abstract syntax tree[C]// 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER). IEEE, 2020: 261-271.
[27] KARNALIM O. Syntax trees and information retrieval to improve code similarity detection[C]// Proceedings of the Twenty-Second Australasian Computing Education Conference. 2020: 48-55.
[28] HAAS R, NIEDERMAYR R, RÖHM T, et al. Recommending Unnecessary Source Code Based on Static Analysis[C]// 2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion). IEEE, 2019: 274-275.
[29] ALON U, ZILBERSTEIN M, LEVY O, et al. A general path-based representation for predicting program properties[J]. ACM SIGPLAN Notices, 2018, 53(4): 404-419.
[30] ZHANG J, WANG X, ZHANG H, et al. A novel neural source code representation based on abstract syntax tree[C]// 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE). IEEE, 2019: 783-794.
[31] CHEN Q Y, LI S P, YAN M, et al. Code clone detection: A literature review[J]. Ruan Jian Xue Bao/Journal of Software, 2019, 30(4): 962-980.

LIANG Yao, born in 1997, postgraduate, is a member of China Computer Federation. Her main research interests include code analysis and so on.

XIE Chun-li, born in 1979, Ph. D, associate professor, is a member of China Computer Federation. Her main research interests include software reliability analysis and deep learning.