软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2023,34(11):5029−5041 [doi: 10.13328/j.cnki.jos.006678]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

基于 GoGCN 的软件系统类交互关系预测*

何 鹏[1,3], 卫 操[1], 吕晟凯[1], 曾 诚[1,3], 李 兵[2]

[1](湖北大学 计算机与信息工程学院, 湖北 武汉 430062)

[2](武汉大学 计算机学院, 湖北 武汉 430072)

[3](湖北省软件工程技术研究中心, 湖北 武汉 430062)

通信作者: 何鹏, E-mail: penghe@hubu.edu.cn

摘 要: 软件系统是一个复杂的人工制品, 类之间的交互关系对软件质量有着潜在影响, 如软件缺陷的级联传播效应就是一个典型. 如何准确预测软件系统中类之间合理关系, 优化设计结构是软件质量保障的一个开放问题. 从软件网络观的视角, 综合考虑软件系统中类与类之间关系 (外部图), 以及每个类内部方法之间关系 (内部图), 将软件系统抽象成一个图中图结构的软件网络, 并在此基础上提出一种基于图中图卷积神经网络的类交互关系预测方法. 首先对每个类内部图进行卷积得到类节点的初始特征, 再通过外部图的卷积更新类节点的表征向量, 最后通过计算类节点对的评估值进行交互预测. 根据在 6 个 Java 开源项目上的实验结果显示, 图中图结构有助于提高软件系统结构的表征能力, 且所提方法与常规网络嵌入方法相比, AUC 值和 AP 值的平均增长率超过 5.5%. 与此同时, 和两种同行方法相比, AUC 值和 AP 值的平均增长率分别在 9.36% 和 5.22% 以上.

关键词: 软件网络; 图中图; 图神经网络; 链路预测; 软件质量

中图法分类号: TP311

中文引用格式: 何鹏, 卫操, 吕晟凯, 曾诚, 李兵. 基于GoGCN的软件系统类交互关系预测. 软件学报, 2023, 34(11): 5029–5041. http://www.jos.org.cn/1000-9825/6678.htm

英文引用格式: He P, Wei C, Lü SK, Zeng C, Li B. GoGCN for Interaction Prediction Between Classes in Software System. Ruan Jian Xue Bao/Journal of Software, 2023, 34(11): 5029–5041 (in Chinese). http://www.jos.org.cn/1000-9825/6678.htm

GoGCN for Interaction Prediction Between Classes in Software System

HE Peng[1,3], WEI Cao[1], LÜ Sheng-Kai[1], ZENG Cheng[1,3], LI Bing[2]

[1](School of Computer and Information Engineering, Hubei University, Wuhan 430062, China)

[2](School of Computer Science, Wuhan University, Wuhan 430072, China)

[3](Software Engineering and Technology Research Center of Hubei, Wuhan 430062, China)

Abstract: As a software system is a complex artifact, the interaction between classes exerts a potential impact on software quality, with the cascading propagation effect of software defects as a typical case. How to accurately predict the reasonable relationship between classes in the software system and optimize the design structure is still an open problem in software quality assurance. From the perspective of software network, this study comprehensively considers the interactions between classes in a software system (class external graph, CEG), and those between internal methods of each class (class internal graph, CIG). The software system is abstracted into a software network with a graph of graphs structure. As a result, a class interaction prediction method based on the graph of graphs convolutional network is proposed. Firstly, the initial characteristics of class nodes are obtained through the convolution of each CIG. Then the representation vector of class nodes is updated through the convolution of CEG, and finally, the evaluation values between class nodes are calculated for interaction prediction. The experimental results on six Java open source projects show that the graph of graphs structure is helpful to improve the representation of software system structure. The average growth rates of the area under the curve (AUC) and

* 基金项目: 国家自然科学基金 (62102136, 61902114, 61977021); 湖北省重点研发项目 (2021BAA184, 2021BAA188); 湖北省科技创新计划 (2019ACA144, 2020AEA008)
收稿时间: 2021-08-17; 修改时间: 2021-11-08, 2021-12-23; 采用时间: 2022-03-15; jos 在线出版时间: 2023-04-27
CNKI 网络首发时间: 2023-04-28

average precision (AP) of the proposed method are more than 5.5% compared with those of the conventional network embedding methods. In addition, the average growth rates of AUC and AP are more than 9.36% and 5.22%, respectively compared with those of the two peer methods.
Key words: software network; graph of graphs (GoG); graph neural network (GNN); link prediction; software quality

1 引 言

在软件工程领域, 软件系统的质量与寿命很大程度上取决于其内部结构. 类作为软件系统中粒度较为适中的一个元素, 它们之间的关系常被用于刻画软件系统的结构, 并且作为“高内聚, 低耦合”设计原则与重构依据[1]. 随着软件系统规模的增大, 类文件之间的交互关系也愈加复杂, 例如新功能的添加, 软件中类之间可能会产生新的交互关系. 若不对软件系统全局结构进行适当调整, 有可能导致开发偏离最初设计[2], 从而严重影响软件质量, 这种现象被称为软件衰退或者软件设计偏移[3], 导致维护成本很高[4,5]. 因此, 如何准确预测软件系统中类之间缺失的关系成为软件维护面临的一个挑战.

幸运的是, 软件系统中类之间的交互关系可用图结构进行简化表示, 即以类为节点, 类之间的交互关系为连边, 抽象成一个类交互图. 于是, 上述问题可变为图数据的学习问题, 其中类交互关系预测可映射为复杂网络中的链路预测 (link prediction) 问题. 链路预测的任务是指通过对已知网络结构的分析, 来评估尚未连接的两个节点之间建立连接的可能性. Xia 等人[6]对 build 配置文件 (如 Make file) 进行逆向解析, 将其中的代码文件与目标之间依赖关系构建为一个依赖图, 通过链路预测方法实现缺失依赖关系的挖掘. 在此基础上, Zhou 等人[7]进一步考虑源代码之间的依赖关系, 通过丰富依赖图从而提高缺失依赖关系预测性能. 鉴于很多设计问题都是因模块间多余的依赖导致, Tommasel 等人[8]则根据软件模块之间的依赖关系, 结合社会网络分析方法和链路预测技术, 实现软件架构臭味 (smell) 的预测. 另外, Pan 等人[9]根据类中方法之间的调用关系网络, 利用社区探测方法指导类之间的重构, 从而优化软件内部结构.

软件系统具有多粒度特性, 从而可从不同粒度进行软件结构的建模[10], 每一种粒度的软件网络模型都可视为一种平面网络, 每一层的网络都包含有大量的有效信息. 如图 1 所示, 假设外部的矩形大节点代表一个类, 边代表类与类之间的各种交互关系, 每个矩形内部的圆形节点为该类中对应的方法或属性, 边对应为它们之间的关联, 形成一个图中图结构. 其中外部的类图 CEG (class external graph) 可以反映整个软件系统中每个类在拓扑结构上的重要性差异, 而每个类的内部图 CIG (class internal graph) 则可用于反映其自身的复杂性.

不难发现, 已有工作存在一定的局限性: (1) 考虑的多为单层结构的 CEG, 即只考虑了类之间的交互关系, 而忽略了每个类内部方法之间的关系; (2) 主要采用基于复杂网络理论的链路预测技术, 而尚未采用深度学习模型进行处理. 众所周知, 图神经网络的兴起, 为图结构数据的学习提供了新的视角. 为填补以上两个空缺, 本文综合考虑类的 CIG 和 CEG, 在得到的图中图 (graph of graphs, GoG) 模型 (即为 CIG 和 CEG 内外组成的网络) 上进行双重图卷积操作, 获取类的深层表征, 进而完成类之间交互关系预测. 本文相关简写对照如表 1 所示. 本文的主要贡献概括如下.

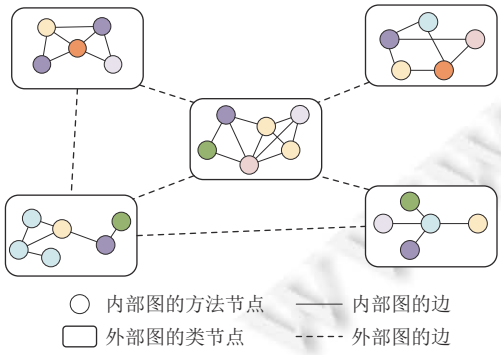


图 1 软件系统图中图结构示意图

表 1 相关简写对照表

描述	简写
外部的类图网络结构	CEG (class external graph)
类的内部图网络结构	CIG (class internal graph)
图中图网络结构	GoG (graph of graphs)
图中图卷积神经网络	GoGCN (graph of graphs convolutional neural network)

(1) 构建软件系统的图中图结构, 提出一种图中图卷积神经网络模型 (graph of graphs convolutional neural network, GoGCN). 为了获得软件系统中更多的类信息, 将 CEG 中单个类节点再扩展成一个 CIG 子图, 协同两种图结构进行信息挖掘.

(2) 在 6 个开源软件项目上, 从横向和纵向两个方面验证了本文所提方法的有效性, 预测性能 AUC 和 AP 值均有提高.

本文第 2 节归纳了目前相关工作已取得的一些成果. 第 3 节定义了本文的研究问题及研究基础. 第 4 节给出了本文方法的详细介绍. 第 5 节为实验设计与结果分析. 第 6 节主要讨论了实验中涉及的一些处理问题、应用价值与不足之处. 最后为全文总结.

2 相关工作

2.1 软件网络

复杂网络是将现实世界中的实体和实体间关系映射成网络的表示形式. 软件网络 (software network)[11,12]作为一种常规的复杂网络, 对软件工程领域相关问题的解决起着至关重要的作用. 有研究者从模块、包、类等粒度对软件系统进行建模, 利用复杂网络理论度量软件系统的复杂性, 分析网络的拓扑结构和形成机理以及演化规律.

近年来, 软件网络常被用于描述软件项目实际中出现的各种问题, 例如研究人员提出基于软件网络复杂性的一些度量指标, 以此评价软件的质量[12]. Gu 等人[13]基于软件网络模型, 提出用内聚度指标度量软件系统中类的连通性. Zhang 等人[14]通过分析许多软件系统, 发现复杂网络中的参数可以用来表示软件结构的属性, 并介绍了基于复杂网络中基本参数的一些有效度量和测量方法. Ma 等人[15]根据耦合和内聚性提出了一套分层的度量指标, 发现所提度量指标可以很好地补充传统软件度量. Pan 等人[16]利用软件网络的度量指标以衡量软件系统的稳定性.

正如前面介绍, 软件系统可从多粒度进行网络建模, Pan 等人[17]从包、类、方法 3 个粒度上, 利用复杂网络度量指标分析了 Azure 软件的演化情况. 何鹏等人[18]围绕 Lehman 演化定律, 从包、类、方法 3 个粒度, 分析了多个软件系统的演化情况.

对于多网络模型, Liu 等人[19]为了全面收集用户的各项行为, 将不同层网络的边加权投影到同层网络空间再进行特征提取. 同样, 在生物分子领域, Wang 等人[20]为了更加突出药物的特征, 将药物分子网络建模为一个由内部的原子网络和外部的药物分子网络构成的图中图. 然而, 在软件工程领域, 虽然有研究者从不同粒度对软件系统开展过建模分析, 但每层网络之间并没有信息交互, 尚不属于前面提到的图中图结构.

2.2 软件工程领域的链路预测

软件作为一种人工的复杂系统, 软件内各元素之间交互关系的识别对于工程师分析维护系统至关重要.

Barros 等人[21]指出当软件在整个生命周期中施加违反设计时架构意图的更改时, 软件体系结构就会降级. 这种现象被称为架构侵蚀. 并针对 Apach Ant 各个迭代的版本进行探索性研究, 将软件工程问题重新定义为模块搜索与优化问题. Pan 等人[17]提出了一种修复面向对象软件系统类结构的方法. 它使用属性-方法网络和方法-方法网络来表示属性、方法以及它们之间的依赖关系, 采用社区探测的方式通过类之间的关系预测, 进行软件重构指导.

为了解决依赖目标和源代码文件之间的依赖关系挖掘问题, Xia 等人[6]首先将构建配置文件 (例如 Make file) 通过逆向工程转换为依赖图, 其中图中的节点对应于配置文件中的实体, 而图中的边对应于这些实体之间的关系, 然后将该问题映射为预测依赖图中缺失边 (链接) 的预测问题, 作者对比分析了 9 种常规链路预测算法的效果.

与 Xia 等人类似, Zhou 等人[7]把重点放在制作构建工具上, 目标是进一步提高类间缺失依赖关系预测的有效性. Diaz-Pace 等人[8]利用系统以前版本的结构特征, 引入线性规划技术, 通过机器学习成功预测了下一个版本的模块依赖性.

不难发现, 该领域已有预测工作主要还停留在依托以往机器学习技术. 在深度学习的发展下, 图神经网络常被

用于图数据的学习, 其强大的网络结构表征能力, 为探索该方向的链路预测问题提供了新视角.

3 研究基础

3.1 问题定义

本文主要针对软件系统中类之间的交互开展预测. 在 UML 建模中, 类之间的交互关系主要包括泛化 (generalization)、实现 (realization)、组合 (composition)、聚合 (aggregation)、关联 (association) 和依赖 (dependency) 等. 本文对上述交互关系暂不进行区分, 即两个类若存在以上任何一种关系则它们之间存在一条交互连边.

本文将重点围绕以下两个研究问题进行探讨.

RQ1: 图中图结构建模能否提高软件系统的表征能力?

软件建模过程中, 除了类之间的交互信息之外, 类内部元素的交互信息, 如内部方法之间的关系, 是否有助于提高对软件系统的表征学习?

RQ2: 本文提出的 GoGCN 模型对软件类之间交互预测是否有效?

图中图模型在复杂网络研究领域被证明有效, 相比之下, 在软件工程领域, 结合图神经网络学习, 该模型对软件系统中类之间交互预测是否仍然有效?

3.2 图中图结构

本文将类之间交互关系组成的图称为类的外部图 (CEG), 记为 $\mathcal{G}=(\mathcal{V},\mathcal{R})$, 其中, $\mathcal{V}$ 是节点集, 每个节点代表一个类, $\mathcal{R}$ 是 CEG 中所有的边集. 实际上, 类之间的交互是建立在它们内部元素之间的交互基础上形成, 因此, 每个类内部的方法或属性之间构成的图称为类的内部图 (CIG), 用 $G_i=(V_i,R_i)$ 表示类 i 的内部图 ($G_i\in\mathcal{V}$), $|V_i|=n$ 表示类 i 中方法或属性节点的个数, 为了便于描述, 我们将方法和属性节点统称为特征节点, R_i 表示类 i 的局部图 G_i 中所有的边 (具体对照关系见表 1). 对于给定的两个类 i,j, 本文的目标是预测它们的内部图 G_i 与 G_j 间的连接概率. 图 2 给出了两个类文件对应的简单图中图构建示例, 类 A 与类 B 的交互是由于类 B 中的方法 b1() 调用了类 A 中的方法 a3().

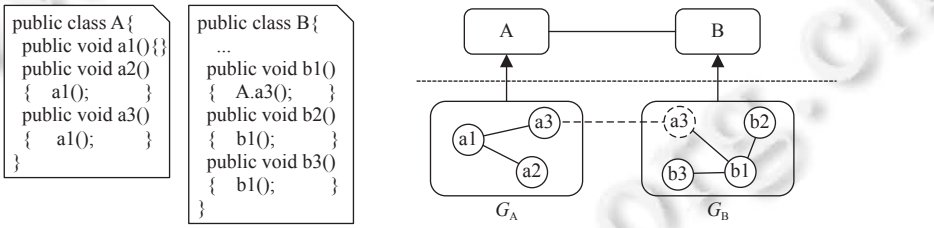


图 2 构建图中图的一个简单示例 (B 类中 A.a3() 方法为关联关系)

4 研究方法

本节将详细介绍图中图神经网络模型, 主要框架如图 3 所示, 包括以方法特征为输入的类内部图神经网络和生成用于预测任务的类外部图神经网络. 每个类 CIG 图神经网络得到的隐含特征将作为该类在 CEG 图神经网络学习的初始输入, 而 CEG 图神经网络上的特征聚合将进一步提高类 CIG 的表征能力.

4.1 CIG 图神经网络模型

方法代表类可以做什么, 是类的功能体现. 如果模型可以很好地识别类中的这些方法, 并用这些方法及方法间的关系来表示类, 则该模型可以更准确地学到类的表征. 因此, 本文设计了 CIG 图神经网络模型.

同时, 在类的 CIG 中, 不难发现不同节点之间的抱团效果存在差异, 即通常表现出一定的社区分布结构 (功能模块化)[22], 如图 3 中的内部图, 按照社区划分策略, 6 个节点划分为两个明显的子社区, 而这种社区结构从拓扑上

也一定程度可反映类之间的差异. 为此, 正如 Xu 等人[23]的工作所证明, 单个通用图卷积层只能聚合节点及其近邻的特征, 为了得到类的多尺度子结构特征, 我们对输入图进行了多层图卷积运算和自注意力图池化处理.

$$M_{(l+1)} = GCN_l(A, M_l) \tag{1}$$

$$GCN_l(A, M_l) = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}M_l W_l\right) \tag{2}$$

其中, $M_l \in \mathbb{R}^{n\times d}$ 为内部图 G 第 l 层的隐含特征矩阵, $\tilde{A} = A + I$ 是内部图 G 带自连接的邻接矩阵, $\tilde{D}$ 是 $\tilde{A}$ 的对角度矩阵, 包含每个节点的度. W_l 是权重矩阵, σ 为激活函数, 本文选择 $ReLU(x) = \max(0, x)$ 作为激活函数.

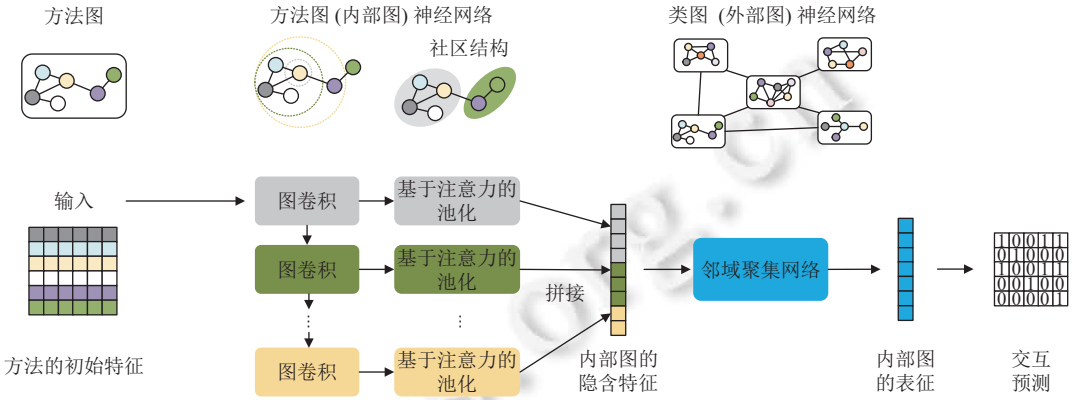


图 3 本文图中图卷积神经网络模型框架

如图 3 所示, 自注意力图池化层以每个图卷积层的输出作为输入, 通过学习该层内部图 G 中每个方法节点的自注意力得分 $s_l = \mathbb{R}^{n\times 1}$, 从而从中选择最具代表性的子结构 (社区).

$$s_l = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}M_l W_{\text{att}}^l\right) \tag{3}$$

其中, $W_{\text{att}}^l \in \mathbb{R}^{d\times 1}$ 是池化层的注意力权重矩阵. 为了选择有代表性的子结构, 图的池化层计算内部图中每个方法的注意力分值, 并返回得分最高的 top-k 个方法.

$$idx = top(\boldsymbol{s}_l, k) \tag{4}$$

$$\boldsymbol{s}_{\text{mask}} = \boldsymbol{s}_{idx} \tag{5}$$

$$M_{\text{sel}} = M \odot \boldsymbol{s}_{\text{mask}} \tag{6}$$

其中, $top()$ 函数返回注意力得分最高的前 k 个方法节点的索引[24], $\boldsymbol{s}_{\text{mask}} \in \{0,1\}^{n\times 1}$ 是由注意力分值决定的 mask 向量, $\odot$表示 mask 的内积, M_{sel} 是内部图 CIG 中选定的方法节点的特征矩阵. 对于选中的方法 M_{sel} 进行求和平均池化操作, 得到该层卷积操作的内部图隐含特征. 经过多层图卷积和自注意力池化层后, 将各层得到的图隐含特征进行拼接, 形成一个 CIG 的隐含特征向量 $\boldsymbol{x}_G$.

由于采用分层图池化结构, 使得图表示方法能有效地保留子结构信息. 可以识别在类的交互中起关键作用的功能子模块, 并用这些子模块来表示类.

4.2 CEG 图神经网络模型

通过上述 CIG 图神经网络模型学习可以很好地得到每个类自身内部在方法层的表征. 但是在一个软件系统中, 类不是孤立存在的, 而是通过类之间的各种交互来实现系统功能. 换句话说, CEG 中的信息对预测类之间的交互同样至关重要, 它使模型能够获取更高阶的交互信息, 同时提升捕获类中代表性子结构的能力.

类间相互作用的类型取决于所设计的内部方法的类型, 尽管本文不考虑具体交互的类型, 但如前文所述, 系统的功能是通过类的交互, 也即为方法/属性的相互调用来实现的. CEG 的邻居聚合可以收集邻居信息, 帮助总结与目标类相交互的类的相关信息.

如第 3.2 节介绍, 类的 CEG 表示为 $\mathcal{G} = (\mathcal{V}, \mathcal{R})$, 单个类的内部图 $G_i \in \mathcal{V}$. 为了更好地提取 CEG 中节点全局结构的隐藏特征, 有必要将每个节点的特征与图中其他关联节点的特征相结合, 换而言之, CEG 中每一个节点在迭

代学习过程中都会聚合其邻居节点在上一层得到的嵌入向量, 并与自身在上一层的嵌入向量相结合, 具体表示如下:

$$\boldsymbol{x}_{G_i}^{(l+1)}=\sigma\left(\boldsymbol{x}_{G_i}^{(l)}+\sum_{G_j\in N(G_i)}\boldsymbol{x}_{G_j}^{(l)}\right) \tag{7}$$

其中, $\boldsymbol{x}_{G_i}^{(l)}\in\mathbb{R}^d$ 表示类 i 的内部图 G_i 在模型的第 l 层的隐含嵌入向量, d 为嵌入向量的维度, 初始化时, 令 $\boldsymbol{x}_{G_i}^{(0)}=\boldsymbol{x}_{G_i}$. $N(G_i)$ 表示 G_i 在外部图中的邻居, σ 表示的是非线性激活函数, 本文使用的是 $ReLU(x)=\max(0,x)$ 函数.

4.3 GoGCN 模型训练

由 CIG 和 CEG 组合构成 GoGCN 模型, 我们使用特定于目标任务的损失函数来优化参数. 在类间交互关系预测中, 本文尚未考虑交互的类型, 所以将类间交互关系预测视为一个链路预测问题. 用两个类节点表示的点积作为两个节点链接的概率:

$$p_{ij}=\sigma(\boldsymbol{x}_{G_i}\cdot\boldsymbol{x}_{G_j}) \tag{8}$$

其中, σ 是激活函数, 使用的是 *Softmax* 函数. 训练中将无链接视为负链接, 并且为了使模型训练过程中能较好地学习到外部图中的链接情况, 我们通过负抽样来优化训练. 对于每个正边对 (G_i,G_j), 通过随机选择一个分子图 G_m, 采样一个随机的负边 (G_i,G_m). 同时使用以下交叉熵损失函数来优化模型:

$$L_{\mathrm{CEG}}=\sum_{(G_i,G_j)\in G_{\mathrm{CEG}}}-\log(p_{ij})-E_{m\sim p_j}\log(1-p_{im}) \tag{9}$$

5 实验分析

5.1 数据集

本文数据集选取依据主要参考了公开 PROMISE 软件工程库 (http://promise.site.uottawa.ca/SERepository/) 中的项目[8], 选取了 Apache 开源平台上 6 个项目, 分别为 JMeter、Commons-codec、Ivy、Ant-1.3、Ant-1.5 和 Maven. JMeter 是一款用于测试 Java 程序的工具, 最初只用于 Java Web 应用程序的测试, 后来推广到所有的 Java 应用程序; Commons-codec 是 Apache 开源组织提供的用于摘要运算、编码的包; Ivy 是一个免费基于 Java 的依赖管理器, 它提供了一些强大的功能包括依赖传递、Ant 集成、 Maven 存储库兼容等; Ant 是 Java 的一个编译工具, 选取了 1.3 和 1.5 这两个版本; Maven 是一款服务于 Java 平台自动化构建的工具.

针对每个软件项目, 首先对下载的源代码使用 DependencyFinder (https://depfind.sourceforge.io/) 进行类之间依赖关系解析, 并保存成 XML 文件, 以获取类间的交互关系和特征节点间的交互关系, 而后对类内的节点进行抽取, 并对该类所有特征节点进行排序编码, 依据该项目具体的规模大小转化为 n 位二进制向量表示, n 的大小由特征节点个数的最大值决定, 最后构建整个软件的图中图模型. 如表 2 所示, 6 个项目 CEG 的节点规模从 57–3 958 个, 连边从 60–10 811 条, CIG 的平均节点规模最大为 21.18, 连边为 14.11. 所选取的 6 个数据集在类内网络 CIG 上规模相近, 在类间结构 CEG 上呈现递增效果, 贴合实际生产环境中不同规模的软件, 能较好代表现实生活中不同功能需求的软件.

表 2 本文实验数据集信息统计

数据集	版本号	外部图CEG		内部图CIG	
		节点数	边数	平均节点数	平均边数
Commons-codec	1.1	57	60	16.49	13.95
Ant	1.3	291	778	21.18	14.11
	1.5	903	3 315	16.26	11.00
Ivy	2.5.0	589	1 888	15.11	11.49
JMeter	5.4.1	1 572	5 706	20.69	12.74
Maven	3.6.3	3 958	10 811	12.98	9.42

5.2 实验设置

以所选取数据集的真实软件项目代码为基础, 构建出图中图软件网络, 在模型训练过程中, 对于外部图网络 $\mathcal{G}=(\mathcal{V},\mathcal{R})$, 将已知的连边 $\mathcal{R}$ 分为训练集 $\mathcal{R}_t$ 和测试集 $\mathcal{R}_v$ 两部分. 从数据集中将 $\mathcal{R}_v$ 这部分连边删去, 由数据集中剩余的 $\mathcal{R}_t$ 部分进行训练, 实验预测任务为找出预先删除的连边.

本文实验是在 Linux 系统下用 Python 语言完成, 使用 Adam 优化器[25]在 PyTorch 框架实现, 训练集、验证集和测试集按 6:2:2 的比例进行划分. 主要参数设置信息如表 3 所示.

表 3 模型参数设置

参数名称	参数值
学习率 γ	0.001
隐藏层特征维度	384
外部图表示维度	128
池化率	0.5
丢失率	0.2
迭代次数	30
批训练大小	100

5.3 评价指标与基准方法

本文选择 AUC 和平均精度 AP (average precision) 作为评价指标. AUC 又称为 ROC 曲线下方所覆盖的面积, AUC 值越大说明分类器的性能越好. AP 则为以召回率 Recall 为横轴, 精确率 Precision 为纵轴所绘制的曲线下方面积, AP 值越高, 表示效果越好.

为了验证在 GoG 上本文双重图卷积神经网络的有效性, 选取了 5 个常用网络嵌入学习方法做纵向对比: DeepWalk、node2vec、struc2vec、LINE 和 SDNE.

DeepWalk[26]是较早提出的一种效果较好的网络嵌入方法, 采用神经网络模型 Skip-Gram 进行图嵌入, 把图的表示学习作为一种处理链路预测的方法. 该方法分为随机游走和生成表示向量两部分, 随机游走部分从图中提取节点序列, 然后利用 Word2Vec 将每个节点表示为一个固定维度的向量.

node2vec[27]是一种综合考虑深度优先遍历和广度优先遍历邻域的图嵌入方法, 是一种学习网络节点连续特征的框架, 同时将节点映射到一个低维特征空间, 以保护节点的网络邻域.

struc2vec[28]是从空间结构相似性的角度来定义节点相似度, 其思想是具有相同度数的节点结构相似, 若各自邻接节点仍然具有相同度数, 则它们的相似度就更高.

LINE[29]是一种基于邻域相似假设的方法, 并且使用广度优先遍历构造邻域, 适用于任意类型的信息网络. 该方法设计的目标函数保留了局部和全局网络结构, 且提出的边采样算法解决了经典随机梯度下降的局限性.

SDNE[30]是一个由多层非线形函数组成的深度学习模型来捕捉网络结构, 同时联合优化节点的一阶和二阶邻域网络结构, 用有监督的部分捕捉网络局部结构, 无监督的部分捕捉网络全局结构, 是一个半监督的模型.

另外, 为了进一步验证 GoGCN 的有效性, 我们引入两个已有的基于类交互图的关系预测方法, 进行同工作的横向对比.

在文献 [6] 中, 作者分析了 9 种链路预测算法对源代码文件之间缺失依赖关系的预测, 发现基于偏好连接 (preferential attachment, PA) 的效果最好, 为此, 本文引入此方法作为第 1 个对比对象, 标记为 PA. 另外, 考虑到除软件系统源代码之间的交互外, 软件系统的配置文件中也常隐含各种交互关系, 于是提出了 BuildPredictor 方法[7], 文本将该方法作为第 2 个对比对象, 标记为 BuildPredictor.

5.4 实验结果

RQ1: 图中图结构建模能否提高软件系统的表征能力?

我们先仅在 CEG 上采用 5 种基准网络嵌入方法进行学习, 同时我们采用第 4.2 节中介绍的 CEG 图神经网络模型学习, 由于没有 CIG 层的学习, 所以对每个类节点进行 one-hot 编码方式初始化, 标记为 CEGCN. 然后, 再在 GoG 上, 分别运行上述 5 种方法和本文的 GoGCN 方法, 其中前 5 种方法我们采取先对每个类的 CIG 进行网络嵌入学习, 通过对类中每个方法的嵌入向量进行取平均值, 得到每个类的 CIG 嵌入向量$\boldsymbol{x}_G$, 然后再在 CEG 上继续学习, 更新并得到每个类的最终表征向量$\boldsymbol{x}'_G$.

表 4 结果显示, 整体上, 无论使用哪种网络嵌入学习方法, 在 GoG 结构上的预测结果普遍比在单层的 CEG 上要好, 表现为取得更好的 AUC 值和 AP 值. 例如在 Ant-1.3 中, 使用本文提出的 GoGCN 方法取得的 AUC 值和 AP 值分别为 0.924 和 0.917, 而 CEGCN 的 AUC 值和 AP 值分别为 0.839 和 0.846. 另外, 从表中加粗标记的结果来看, 无论是 CEGCN 还是 GoGCN, 都比相应的网络嵌入方法的预测效果要高, 紧接着是 DeepWalk 和 node2vec 两种方法. 此外, 如图 4 所示, 不难发现, 规模越大的软件系统, 预测效果倾向于会更好, 说明本文方法适合大型复杂软件系统结构信息的表征学习.

表 4 仅在外部图 CEG 上学习结果的 AUC 值和 AP 值

结构	方法	Ant-1.3		Ant-1.5		Codec		Ivy		JMeter		Maven	
		AUC	AP	AUC	AP	AUC	AP	AUC	AP	AUC	AP	AUC	AP
CEG	DeepWalk	0.823	0.814	0.846	0.846	0.792	0.736	0.844	0.845	0.863	0.867	0.872	0.875
	node2vec	0.803	0.795	0.798	0.787	0.799	0.765	0.885	0.88	0.907	0.906	0.913	0.916
	struc2vec	0.497	0.499	0.515	0.504	0.361	0.488	0.564	0.561	0.628	0.61	0.679	0.696
	LINE	0.550	0.530	0.551	0.531	0.413	0.566	0.637	0.663	0.758	0.769	0.786	0.792
	SDNE	0.609	0.632	0.603	0.63	0.396	0.581	0.717	0.723	0.767	0.776	0.793	0.799
	GCN	**0.839**	**0.846**	**0.875**	**0.869**	**0.829**	**0.822**	**0.913**	**0.901**	**0.922**	**0.917**	**0.925**	**0.921**
GoG	DeepWalk	0.836	0.827	0.857	0.863	0.868	0.879	0.853	0.848	0.916	0.907	0.923	0.928
	node2vec	0.828	0.817	0.851	0.848	0.847	0.86	0.893	0.882	0.937	0.926	0.911	0.923
	struc2vec	0.539	0.528	0.553	0.541	0.556	0.558	0.563	0.558	0.623	0.613	0.654	0.640
	LINE	0.576	0.552	0.664	0.660	0.486	0.609	0.651	0.657	0.767	0.814	0.744	0.796
	SDNE	0.633	0.637	0.706	0.703	0.406	0.456	0.734	0.746	0.764	0.779	0.791	0.799
	GCN	**0.924**	**0.917**	**0.928**	**0.921**	**0.904**	**0.906**	**0.943**	**0.941**	**0.951**	**0.950**	**0.958**	**0.960**

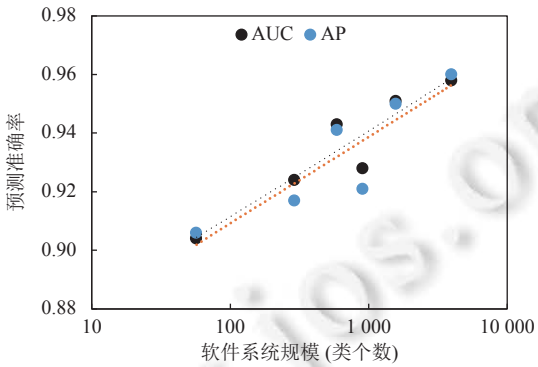


图 4 软件系统规模与预测性能关系

表 5 和表 6 分别给出了 GoG 和 CEG 情境下 AUC 值和 AP 值的增长率. 针对 AUC 指标, 相比单层的 CEG 情境, 前 3 个项目 (Ant-1.3、Ant-1.5 和 Codec) 在 GoG 情境下预测性能都有所提高, 表现为增长率均为正值, 其中它们分别在采用本文方法、LINE 和 struct2vec 时 AUC 增长率最大, 对应为 10.13%、20.51% 和 54.02%. 然而, 后 3 个项目 (Ivy、JMeter 和 Maven) 则有所不同, GoG 情境下不同模型的预测性能有增也有减, 其中 Maven 上表现尤为明显, 除 DeepWalk 和本文方法外, 使用其他 4 个网络嵌入方法结果都不理想. 尽管如此, 各模型的整体效果仍表现为, 在 GoG 情境下比 CEG 情境下的预测性能要好.

表 5 GoG vs. CEG 情境下的 AUC 值增长率 (%)

方法	Ant-1.3	Ant-1.5	Codec	Ivy	JMeter	Maven	平均增长率
DeepWalk	1.58	1.30	9.60	1.07	**6.14**	**5.85**	4.23
node2vec	3.11	6.64	6.01	0.90	3.31	−0.22	3.17
struc2vec	8.45	7.38	**54.02**	−0.18	−0.80	−3.68	**7.52**
LINE	4.73	**20.51**	17.68	2.20	1.19	−5.34	5.22
SDNE	3.94	17.08	2.53	2.37	−0.39	−0.25	3.84
GCN	**10.13**	6.06	9.05	**3.29**	3.15	3.57	5.75

表 6 GoG vs. CEG 情境下的 AP 值增长率 (%)

方法	Ant-1.3	Ant-1.5	Codec	Ivy	JMeter	Maven	平均增长率
DeepWalk	1.60	2.01	**19.43**	0.36	4.61	**6.06**	5.40
node2vec	2.77	7.75	12.42	0.23	2.21	0.76	4.10
struc2vec	5.81	7.34	14.34	−0.53	0.49	−8.05	2.38
LINE	4.15	**24.29**	7.60	−0.90	**5.85**	0.51	**6.15**
SDNE	0.79	11.59	−21.51	3.18	0.39	0.00	−0.51
GCN	**8.39**	5.98	10.22	**4.44**	3.60	4.23	6.05

针对 AP 指标, 实验结果的整体趋势与 AUC 指标大体一致, 表现为在 GoG 情境下比 CEG 情境下的预测性能要好. 有所不同的是, Ant-1.3、Ant-1.5 和 JMeter 这 3 个项目在 GoG 情境下预测性能的增长率均为正, 而对于 Codec、Ivy 和 Maven 项目的预测性能有增也有减.

另外, 图 5 给出了同在 GoG 情境下, 本文方法相比已有 5 种网络嵌入方法在 AUC 值和 AP 值上的平均增长率情况. 相比 DeepWalk 和 node2vec 方法, 两个评价指标的增长率均超过 6%, 而相比其他方法, 增长率幅度较大, 尤其是 struc2vec 方法, 增长率超过 60%.

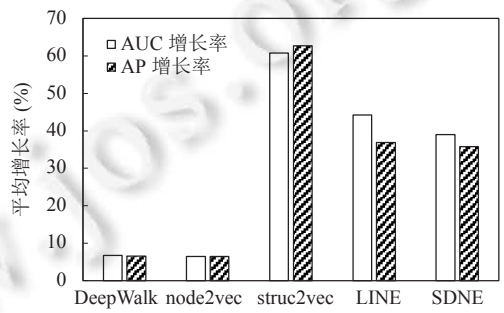


图 5 GoG 情境下本文方法的平均增长情况

综上所述, 采用软件系统中类的图中图结构比只考虑类之间交互关系的外部图进行类的表征学习效果更好, 即综合考虑类中方法间的关系 (CIG) 与类之间的关系 (CEG) 确实能提高软件系统的表征能力.

RQ2: 本文提出的 GoGCN 模型对软件类之间交互预测是否有效?

前面实验已从纵向上验证了本文方法的有效性, 为了进一步验证 GoGCN 的有效性, 接下来进行同行工作的横向对比.

根据表 7 和图 6 的结果显示, 本文提出的 GoGCN 方法比已有两个同行方法效果都好, 其中相比于 PA 方法, AUC 的增长率最大达到 13.79%, 最小也接近 5%, 平均达到 9.47%; 而 AP 的增长率最大达到 13.11%, 最小也超了 4%, 平均达到 9.36%. 相较于 BuildPredictor 方法, 趋势一致只是优势变小, AUC 和 AP 的平均增长率分别为 5.22% 和 5.41%. 另外, 从实验结果不难看出, 在后两个项目上, 本文 GoGCN 方法的优势一般, 其中在 Maven 上, GoGCN 方法比 BuildPredictor 方法的 AP 值只提高了 0.007, 增长率为 0.73%.

整体上, 相比已有两个同行方法, 本文提出的 GoGCN 模型对软件类之间的交互预测在性能上有所提高.

表 7 GoGCN 与已有方法的结果对比

Data	Ant-1.3		Ant-1.5		Codec		Ivy		JMeter		Maven		平均值	
	AUC	AP	AUC	AP	AUC	AP	AUC	AP	AUC	AP	AUC	AP	AUC	AP
PA	0.812	0.812	0.837	0.834	0.796	0.801	0.863	0.854	0.902	0.892	0.913	0.923	0.854	0.853
BuildPredictor	0.839	0.846	0.875	0.869	0.829	0.822	0.913	0.901	0.932	0.917	0.942	0.953	0.888	0.885
GoGCN	0.924	0.917	0.928	0.921	0.904	0.906	0.943	0.941	0.951	0.950	0.958	0.960	0.935	0.933

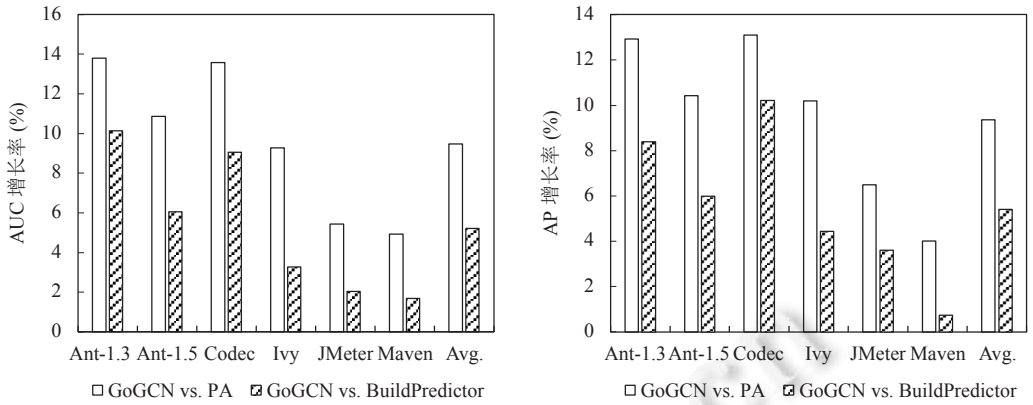


图 6 本文方法 AUC 值和 AP 值的提升幅度

5.5 灵敏度分析

本节主要对 GoGCN 模型中几个关键参数的影响进行分析.

由前文实验结果分析可知, 本文方法预测性能与项目的规模一定程度上成正相关, 我们选择项目规模相对适中, 预测效果较为稳定的 Ivy 数据集进行灵敏度分析, 并选取 AUC 作为评价指标.

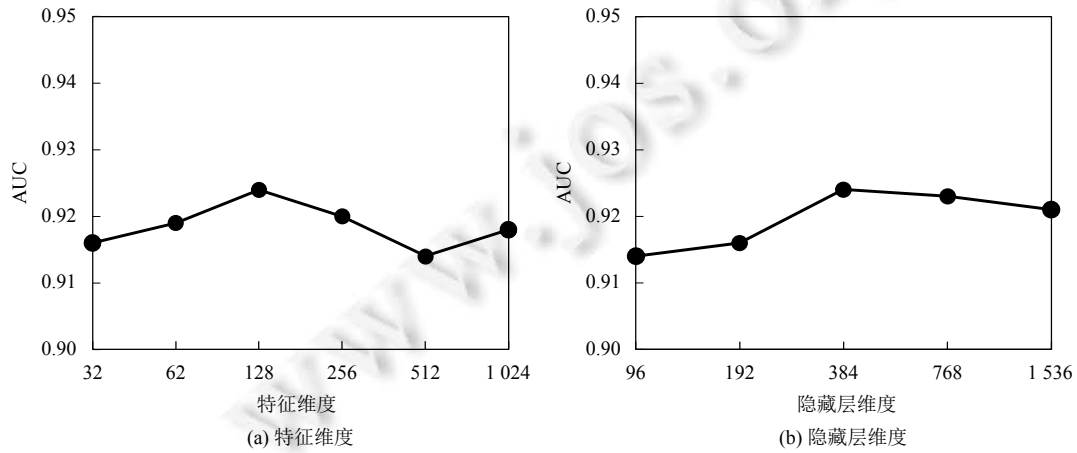
在保持其他参数与第 4.2 节中一致的基础上, 通过不断改变测试目标参数的设置, 本节依次分析了 4 个超参的影响: 输出表示和隐藏特征的维度、学习率和池化率.

如后文图 7 所示, 本文模型在外部图的输出表示维数为 128 时效果最佳, 隐藏特征的最优维度为 384. 至于学习率和池化率, 最佳点分别为 0.001 和 0.5. 从整体而言, 超参数的变化对模型性能影响较为有限, AUC 值的波动范围保持在 0.01 以内, 即本模型具有相对较好的鲁棒性, 对参数的依赖性有限.

6 讨 论

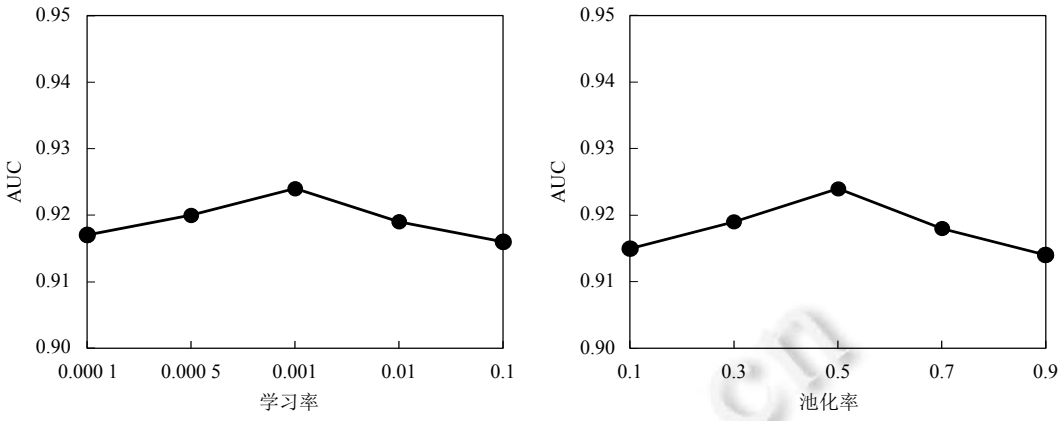
本节主要讨论文中涉及的一些实验处理问题、应用价值与不足之处.

在回答 RQ1 的实验中, 为验证软件系统的 GoG 模型的作用, 采取了与 CEG 对比, 而不是 CIG. 主要是因为本文目标是围绕软件系统中类之间的交互预测, 而 CIG 是对类自身的一个内在刻画, 对类的 CIG 学习更算是对类的属性学习, 其输出结果作为 CEG 学习时类的初始向量. 所以, 其实最后还是等价于基于 CEG 的类交互预测.



(a) 特征维度 (b) 隐藏层维度

图 7 参数影响结果分析



(c) 学习率 (d) 池化率

图 7 参数影响结果分析 (续)

整体上, 实验结果较为有力地证实了对软件系统进行 GoG 建模的有用性, 尤其是在本文的图神经网络学习情境下. 然而, 从前文表 5 和表 6 中, 也发现在一些常规网络嵌入学习方法下, 采用 GoG 模型甚至不如 CEG 模型, 说明对于一些软件项目 (尤其是当软件项目规模偏大时) 细化类的内部结构信息未必一定有效. 参照前文表 2 给出的内部图 CIG 平均规模信息, Maven 项目规模最大, 一种可能的解释是规模越大的项目, 在开发过程中会倾向于更专业, 会更注重“高内聚低耦合”以及设计模式的作用, 因而类设计的相对更小巧, 越小的类在 CIG 学习时可提取的信息越少, 从而导致效果越有限.

同时, 我们也考虑过将类内结构 CIG 和类间结构 CEG 合并起来作为初始向量这一研究思路, 本文方法在于提取社区子结构来表征类节点, 具体的意义便在于区分不同的类. 相较于将类内结构 CIG 和类间结构 CEG 合并起来同时输入这种思路, 在类的差异化体现上, 本文方法是要优于广泛的将类内结构和类间结构合并起来作为初始输入的处理, 并且这种差异化的体现也贴合数据集特点, 对于本文中所提到的 6 个常用数据集, 详细信息如表 2 所示, 可以看出, 随着数据集体量的增大 (节点数和边数的增加), 类 (内部图 CIG) 的平均节点数和平均边数其实较为接近, 在一个较小的范围内波动, 符合软件演化中一个类不会有过多或过少的方法节点规律, 即类内网络结构差异化不够明显. 因此, 本文未考虑该基线实验.

在回答 RQ2 的实验中, 我们选取了同领域的两个代表性方法进行比较, 其中 PA 方法是基于复杂网络中的偏好连接理论, 即节点 x 将作为新链路端点之一的概率与该节点当前邻居数成正比, 换句话说, 节点的度越大, 该节点与其他节点建立新链接的可能性越大, 属于一种简单的注意力机制. BuildPredictor 方法算是在本文基于源代码的 CEG 基础上, 通过其他信息 (如配置文件、co-change 关系等) 来完善类之间存在的一些交互关系, 是对 CEG 结构的升级, 而本文则是在不改变常规 CEG 的基础上, 通过对 CIG 的学习来优化 CEG 中每个类节点在图数据学习时的初始属性.

在灵敏度分析中, 本文只给出了 Ivy 项目上的 AUC 实验结果, 不同项目上模型的各参数效果确实不一样, 但趋势基本一致, 这也是我们只选择一个项目分析的原因之一. 其次, 由前文实验结果分析可知, 本文方法预测性能与项目的规模成正相关, 而 Ivy 项目的规模相对适中, 运行效率也适中, 具有较好的代表性. 另外, 该部分也只采用 AUC 作为评价指标, 原因是前面实验结果中, AUC 和 AP 两个指标的评价效果也基本保持一致.

本文提出的 GoGCN 在预测结果上取得了一定效果, 但仍然存在一些挑战: (1) 本文的图中图结构还只是对软件系统最为基本的建模, 在 CIG 和 CEG 中都未考虑边的方向和权重, 而实际中, 方向和权重也是非常有价值的信息, 考虑后相信效果会要好. (2) 在对 CEG 进行图卷积学习时, 如公式 (7) 所示, 在迭代学习过程中采取的是直接聚合邻居节点在上一层迭代得到的嵌入向量, 并与自身在上一层迭代得到的嵌入向量进行简单线性结合, 未考虑邻居节点的差异以及邻居节点聚合向量与节点自身向量的比重. (3) 目前只在 Java 开发的项目上进行了验证分析, 后续还需进一步扩展到更多开发语言的项目中去, 验证方法的通用性. (4) 在软件工程这种特殊的领域中, 由于软件是不断迭代的, 同一个数据集通常是在原始版本上不断迭代产生的结果, 这对于测试集和验证集的选取有着得

天独厚的优势, 这种数据集更能证明模型的可用性, 但同样这对于数据集的选取较为苛刻, 在后续的版本中我们会尝试去构建这种数据集, 力求为后续研究以及软件工程领域的工作者们做出自己应有的贡献.

7 结 论

本文从软件网络的视角, 从两个粒度将软件系统抽象为一个图中图结构, 并引入图卷积神经网络学习模型, 提出了一种基于图中图神经网络的软件系统中类交互预测方法 GoGCN. 所提方法在 5 个软件项目上进行了实证分析, 实验结果表明, 相比仅在 CEG 情境下, 本文所提出方法的 AUC 和 AP 值平均增长率达到 5.75% 和 6.05%, 同时在 GoG 情境下, 本文方法相比其他网络嵌入方法, AUC 和 AP 值的平均增长率在 6%–60% 范围, 证实了 GoG 模型的有用性, 以及 GoGCN 的优势. 另外, 与同行方法对比, 进一步证实了本文方法的有效性.

References:

[1] Diaz-Pace JA, Tommasel A, Godoy D. Can network analysis techniques help to predict design dependencies? An initial study. In: Proc. of the 2018 IEEE Int'l Conf. on Software Architecture Companion (ICSA-C). Seattle: IEEE, 2018. 64–67. [doi: 10.1109/ICSA-C.2018.00025]

[2] Mens T, Tourwe T. A survey of software refactoring. IEEE Trans. on Software Engineering, 2004, 30(2): 126–139. [doi: 10.1109/TSE.2004.1265817]

[3] Fowler M. Refactoring: Improving the design of existing code. In: Proc. of the 2nd XP Universe and the 1st Agile Universe Conf. on Extreme Programming and Agile Methods. Chicago: Springer, 2002. 256–256.

[4] Coleman D, Ash D, Lowther B, Oman P. Using metrics to evaluate software system maintainability. Computer, 1994, 27(8): 44–49. [doi: 10.1109/2.303623]

[5] Guimaraes T. Managing application program maintenance expenditures. Communications of the ACM, 1983, 26(10): 739–746. [doi: 10.1145/358413.358421]

[6] Xin X, Lo D, Wang XY, Zhou B. Build system analysis with link prediction. In: Proc. of the 29th Annual ACM Symp. on Applied Computing. Gyeongju: ACM, 2014. 1184–1186. [doi: 10.1145/2554850.2555134]

[7] Zhou B, Xia X, Lo D, Wang XY. Build predictor: More accurate missed dependency prediction in build configuration files. In: Proc. of the 38th IEEE Annual Computer Software and Applications Conf. Vasteras: IEEE, 2014. 53–58. [doi: 10.1109/COMPSAC.2014.12]

[8] Díaz-Pace JA, Tommasel A, Godoy D. Towards anticipation of architectural smells using link prediction techniques. In: Proc. of the 18th IEEE Int'l Working Conf. on Source Code Analysis and Manipulation. Madrid: IEEE, 2018. 62–71. [doi: 10.1109/SCAM.2018.00015]

[9] Pan WF, Li B, Ma YT, Liu J, Qin YY. Class structure refactoring of object-oriented softwares using community detection in dependency networks. Frontiers of Computer Science, 2009, 3(3): 396–404. [doi: 10.1007/s11704-009-0054-y]

[10] Bing L , Pan W , Lu J . Multi-granularity dynamic analysis of complex software networks. In: Proc. of the 2011 IEEE Int'l Symp. on Circuits & Systems. Rio de Janeiro: IEEE, 2011. 2119–2124. [doi: 10.1109/ISCAS.2011.5938017]

[11] Ma YT, He KQ, Li B, Liu J. Empirical study on the characteristics of complex networks in networked software. Ruan Jian Xue Bao/Journal of Software, 2011, 22(3): 381–407 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/3934.htm [doi: 10.3724/SP.J.1001.2011.03934]

[12] Myers CR. Software systems as complex networks: Structure, function, and evolvability of software collaboration graphs. Physical Review E, 2003, 68(4): 046116. [doi: 10.1103/PhysRevE.68.046116]

[13] Gu AH, Zhou XF, Li ZH, Li QF, Li L. Measuring object-oriented class cohesion based on complex networks. Arabian Journal for Science and Engineering, 2017, 42(8): 3551–3561. [doi: 10.1007/s13369-017-2588-x]

[14] Zhang HH, Feng WJ, Wu LJ. The static structural complexity metrics for large-scale software system. Applied Mechanics and Materials, 2010, 44–47: 3548–3552. [doi: 10.4028/www.scientific.net/AMM.44-47.3548]

[15] Ma YT, He KQ, Li B, Liu J, Zhou XY. A hybrid set of complexity metrics for large-scale object-oriented software systems. Journal of Computer Science and Technology, 2010, 25(6): 1184–1201. [doi: 10.1007/s11390-010-9398-x]

[16] Pan WF, Chai CL. Measuring software stability based on complex networks in software. Cluster Computing, 2019, 22(S2): 2589–2598. [doi: 10.1007/s10586-017-1353-y]

[17] Pan WF, Li B, Ma YT, Liu J. Multi-granularity evolution analysis of software using complex network theory. Journal of Systems Science and Complexity, 2011, 24(6): 1068–1082. [doi: 10.1007/s11424-011-0319-z]

[18] He P, Wang P, Li B, Hu SW. An evolution analysis of software system based on multi-granularity software network. Acta Electronica

Sinica, 2018, 46(2): 257–262 (in Chinese with English abstract). [doi: 10.3969/j.issn.0372-2112.2018.02.001]

[19] Liu WY, Chen PY, Yeung S, Suzumura T, Chen LL. Principled multilayer network embedding. In: Proc. of the 2017 IEEE Int'l Conf. on Data Mining Workshops. New Orleans: IEEE, 2017. 134–141. [doi: 10.1109/ICDMW.2017.23]

[20] Wang HC, Lian DF, Zhang Y, Qin Y, Lin XM. GoGNN: Graph of graphs neural network for predicting structured entity interactions. In: Proc. of the 29th Int'l Joint Conf. on Artificial Intelligence. Yokohama: IJCAI, 2020. 317–1323. [doi: 10.24963/ijcai.2020/183]

[21] Barros MDO, Farzat FDA, Travassos GH. Learning from optimization: A case study with Apache Ant. Information & Software Technology, 2015, 57: 684–704. [doi: 10.1016/j.infsof.2014.07.015]

[22] Shen HW. Community Structure of Complex Networks. Berlin: Springer. [doi: 10.1007/978-3-642-31821-4]

[23] Xu K, Li CT, Tian YL, Sonobe T, Kawarabayashi KI, Jegelka S. Representation learning on graphs with jumping knowledge networks. In: Proc. of the 35th Int'l Conf. on Machine Learning (ICML). Stockholmsmässan: PMLR, 2018. 5449–5458.

[24] Gao HY, Ji SW. Graph U-Nets. In: Proc. of the 36th Int'l Conf. on Machine Learning. Long Beach: PMLR, 2019. 2083–2092.

[25] Kingma DP, Ba J. Adam: A method for stochastic optimization. In: Proc. of the 3rd Int'l Conf. on Learning Representations. San Diego: ICLR, 2015.

[26] Cai LJ, Xu YB, He TQ, Meng T, Liu HM. PROD: A new algorithm of DeepWalk based on probability. Journal of Physics: Conf. Series, 2019, 1069: 012130. [doi: 10.1088/1742-6596/1069/1/012130]

[27] Grover A, Leskovec J. node2vec: Scalable feature learning for network. In: Proc. of the 22nd ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. San Francisco: ACM, 2016. 855–864. [doi: 10.1145/2939672.2939754]

[28] Ribeiro LFR, Saverese PHP, Figueiredo DR. struc2vec: Learning node representations from structural identity. In: Proc. the 23rd ACM SIGKDD Int'l Conf. Halifax: ACM, 2017. 385–394. [doi: 10.1145/3097983.3098061]

[29] Jian T, Meng Q, Wang MZ, Yan J, Mei QZ. LINE: Large-scale information network embedding. In: Proc. of the 24th Int'l Conf. on World Wide Web. Florence: ACM, 2015. 1067–1077. [doi: 10.1145/2736277.2741093]

[30] Wang DX, Cui P, Zhu WW. Structural deep network embedding. In: Proc. of the 22nd ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. San Francisco: ACM, 2016. 1225–1234. [doi: 10.1145/2939672.2939753]

附中文参考文献:

[11] 马于涛, 何克清, 李兵, 刘婧. 网络化软件的复杂网络特性实证. 软件学报, 2011, 22(3): 381–407. http://www.jos.org.cn/1000-9825/3934.htm [doi: 10.3724/SP.J.1001.2011.03934]

[18] 何鹏, 王鹏, 李兵, 胡思文. 基于多粒度软件网络模型的软件系统演化分析. 电子学报, 2018, 46(2): 257–262. [doi: 10.3969/j.issn.0372-2112.2018.02.001]

何鹏(1988－), 男, 博士, CCF 专业会员, 主要研究领域为面向服务的软件工程, 软件质量分析, 缺陷预测.

曾诚(1976－), 男, 博士, CCF 专业会员, 主要研究领域为服务计算, 机器学习, 软件工程.

卫操(1998－), 男, 硕士生, 主要研究领域为深度学习, 软件交互关系预测.

李兵(1969－), 男, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为软件工程, 云计算, 复杂网络.

吕晟凯(2001－), 男, 本科生, 主要研究领域为深度学习, 个性化推荐系统.