计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.210100200

基于图神经网络的软件系统中关键类的识别

张健雄[1] 宋 坤[1] 何 鹏[1,2] 李 兵[3]
1 湖北大学计算机与信息工程学院 武汉 430062
2 应用数学湖北省重点实验室 武汉 430062
3 武汉大学计算机学院 武汉 430072
(930065661@qq.com)

摘 要 软件系统中通常存在一些在拓扑结构上处于核心位置的关键类,这些类上的缺陷往往会给系统带来极大的安全隐患,识别关键类对工程师理解或维护一个软件系统至关重要。针对这一问题,提出一种基于图神经网络的关键类识别方法。首先利用复杂网络理论,将软件系统抽象为软件网络;其次结合无监督网络节点嵌入学习以及邻域聚合的方式,构建一个编码-解码(encoder-decoder)框架,提取软件系统中类节点的表征向量;最后利用 Pairwise 排序学习实现网络中节点的重要性排序,从而实现软件系统中关键类的识别。为验证所提方法的有效性,选取 4 个 Java 开源软件作为实验对象,并与常用的 5 种节点重要性度量方法以及 2 个已有工作进行对比分析。实验结果表明:与介数中心性、K-core、接近中心性、节点收缩法和 PageRank 等方法相比,该方法识别关键类的效果更好;另外,相比已有工作,在前 15%的关键类节点中,所提方法的召回率和准确率的提高幅度均在 10%以上。

关键词:软件网络;关键类识别;网络嵌入;图神经网络;排序学习
中图法分类号 TP311.53;TP183

Identification of Key Classes in Software Systems Based on Graph Neural Networks

ZHANG Jian-xiong[1], SONG Kun[1], HE Peng[1,2] and LI Bing[3]
1 School of Computer and Information Engineering, Hubei University, Wuhan 430062, China
2 Hubei Provincial Key Laboratory of Applied Mathematics, Wuhan 430062, China
3 School of Computer Science, Wuhan University, Wuhan 430072, China

Abstract There are usually some key classes which are in the core position in the topology structure of software systems. The defects in these classes will bring great security risks to the system. Therefore, it is very important to identify these key classes for engineers to understand or maintain an unfamiliar software system. To do this, the paper proposes a novel method of identifying key classes based on graph neural networks. Specifically, the software system is abstracted as software network by using complex network theory, and then by combining unsupervised network embedding learning and neighborhood aggregation mode, we construct an encoder-decoder framework to extract the representation vector of class nodes in software system. Finally, according to the obtained node representations, Pairwise learning-to-rank algorithm is adopted to realize the importance ranking of nodes, so as to achieve the identification of key classes in software system. In order to verify the effectiveness of our method, an empirical analysis of four object-oriented Java open-source software is done, and we compare it with five commonly used node importance measurement methods and two existing works. The experimental results show that, compared with node centrality, K-core and PageRank, the proposed method is more effective in identifying key classes from the perspective of network robustness. In addition, on the existing public labeled dataset, the recall and precision of this paper are better at the top 15% percent of nodes, and improved by more than 10%.

Keywords Software network, Key class identification, Network embedding, Graph neural network, Learning to rank

到稿日期:2021-01-26 返修日期:2021-05-09 本文已加入开放科学计划(OSID),请扫描上方二维码获取补充信息。
基金项目:国家重点研发计划(2018YFB1003801);国家自然科学基金(61832014, 61902114, 61977021);湖北省科技重大专项(2019ACA144);应用数学湖北省重点实验室开放基金(HBAM201901)
This work was supported by the National Key R & D Program of China (2018YFB1003801), National Natural Science Foundation of China (61832014, 61902114, 61977021), Science and Technology Innovation Program of Hubei Province (2019ACA144) and Open Foundation of Hubei Key Laboratory of Applied Mathematics (HBAM201901).
通讯作者:何鹏(penghe@hubu.edu.cn)

1 引言

随着网络规模的不断扩大，软件系统的复杂度不断提升，软件中存在缺陷是在所难免的。这些缺陷一旦在软件使用过程中暴露出来，影响的往往不仅是缺陷所在的模块本身，还有可能导致整个软件系统的崩溃。有研究[1]指出，软件系统中的缺陷与疾病的传播模型类似，会随着软件系统中各模块间的依赖关系，如方法调用和参数传递等，传播到其他没有缺陷的模块中，也会导致这些模块运行出错。研究表明，大部分软件缺陷仅对系统造成有限的影响[2]，但如果处于软件系统中关键位置的类存在缺陷，则造成系统出现重大故障的概率会更高。因此，有必要提前识别软件系统中的关键类，并对这些关键类给予关注和完善，降低它们出现缺陷和遭到恶意攻击的可能性，从而增加软件系统的稳定性与可靠性。

从 20 世纪 90 年代开始，人们发现网络结构非常适合表示现实世界中高度复杂的系统，将现实世界中存在的各种复杂关系抽象为网络而得到的模型被称为复杂网络(complex network)。通过分析复杂网络来对系统的复杂性进行量化，为科学研究提供了全新的角度[3]，这与软件工程对量化软件系统复杂度的需求不谋而合。

在复杂网络中，存在一些“特殊”节点[4]，与其他节点相比，它们包含更多的全局或局部网络信息。当网络中这部分节点失效时，将可能影响到整个网络。已有研究表明：在复杂网络中，只要 10%左右的重要节点受到攻击，就可以导致整个网络瘫痪[5]。因此，识别复杂网络中有影响力的节点具有重要的理论和现实意义[6]。软件网络是一种典型的人工复杂网络[4]，即以软件系统的元素(包括模块、包、类、方法等)为节点，元素之间的交互关系(包括依赖、调用等)为连边，并且同样具有复杂网络中“无标度”与“小世界”的特性。因此，将复杂网络中重要节点的度量方法引入软件网络中，对软件系统中关键类的识别具有极大的参考价值。

本文首先通过对源代码进行解析，将软件系统抽象为一个类粒度的加权软件网络；其次引入网络表征学习的特征学习优势，提取软件网络中每个类节点的结构特征(一个低维表征向量)；最后构建一个基于图自编码器的排序模型，将每个类节点解码为一个重要性评估值，并依据评估值进行排序，从而实现软件系统中关键类的识别。本文的主要贡献如下。

(1)提出一个改进的图自编码器(Graph Autoencoders)框架，进行面向软件系统全局结构的特征学习，克服已有工作中只考虑局部信息的不足。

(2)将图神经网络的学习结果应用于软件系统的关键类识别。已有关键类识别研究主要基于人工构建度量指标，还未有利用图神经网络来自动学习网络的全局结构信息，并将学习结果应用于软件工程实践中。

本文第 2 节概括了目前在相关领域的研究现状；第 3 节给出了理论基础；第 4 节对所提方法进行了详细介绍；第 5 节为实验部分，主要阐述了实验数据来源、实验步骤、实验结果分析和讨论；最后总结全文并展望未来。

2 相关工作

2.1 软件网络

2.1.1 复杂网络研究

复杂网络的研究最早可追溯到欧拉的“七桥问题”，Watts 等[7]的小世界网络模型以及 Barabási 等[8]的无标度网络模型为复杂网络的研究带来了突破性进展，揭示了复杂网络中重要的“小世界”和“无标度”特性[9]。复杂网络的研究概况为：网络建模原理、社区划分、网络结构度量以及网络行为分析与传播等[10]。

2.1.2 软件网络研究

软件工程领域研究者将复杂网络思路应用于软件系统，形成软件网络观，从而利用复杂网络理论研究软件系统的设计特性，验证软件系统同样具有复杂网络的特性[11]。

已有研究主要是从模块、包、类、方法等粒度对软件系统进行建模，构建软件网络(software network)[12-13]，度量软件系统的复杂性，分析软件网络的拓扑结构及其形成机理和演化规律。Valverde 等[14]最早将复杂网络理论引入软件工程领域，并将类作为节点，将类之间的依赖、继承等关系作为边，建立了类粒度级别上的无向软件网络模型，分析了软件的拓扑结构，验证了软件网络的“小世界”和“无标度”特性。

2.2 网络节点重要性排序研究

2.2.1 复杂网络节点重要性排序

现有的复杂网络节点重要性排序方法主要分为 4 类。第 1 类是根据节点属性或网络全局属性来判断节点重要性，如根据节点的度、介数、接近中心性等属性判断节点重要性[15]。第 2 类是根据节点位于网络中的位置决定其重要性，如 k-shell 分解法[16]。第 3 类是根据删除或收缩节点对整个网络连通性的影响力来评估目标节点的重要性，如节点删除法[17]和节点收缩法[5]。第 4 类是基于特征向量的节点重要性评估方法，如 PageRank[18]，HITS[19]等。

2.2.2 软件网络关键节点的识别

识别软件系统中的关键类，对于软件工程师有效理解或维护一个系统至关重要。软件网络是一种典型的人工复杂网络，软件系统中的关键类识别问题可转换为软件网络中重要节点的识别。Wang 等[20]提出了一种使用介数中心性和类节点之间的交互以及类自身的复杂性指标来识别软件系统中关键类的方法。Perin 等[21]将 PageRank 方法用于软件网络中重要类节点的度量，实验结果发现，度量结果与专家识别的重要类保持一致。Wang 等[22]将引文网络中衡量重要作者的 h 指数引入软件网络中，提出了一种基于 h 指数及其衍生指数的识别软件系统中关键类的方法。随后，Hu 等[23]改进了 h 指数的定义，在计算邻居节点对目标节点的影响力时考虑了权重的因素，使其可以用于加权软件网络的关键类识别。

此外，Wang 等[24]将网页重要性排序中的 HITS 算法引入软件网络研究中，提出了一种基于软件网络中类节点重要

性的集成测试序列生成方法。Li 等[25]分析了软件网络结构，提出了一种软件类重要性度量指标 IC，并使用仿真算法计算节点的 IC 值。Jiang 等[26]提出一种基于 UIO 序列的软件类重要性度量方法，将软件系统转化为以类为单位的有限自动机模型，取代常规的复杂网络模型，随后求解自动机的 UIO 序列并转化为状态转换树，通过计算状态转换树中节点的复杂度，实现对类的重要性排序。Wang 等[27]考虑类的位置及其对软件信息流的控制能力，并关注类与其邻居的交互，以及类自身的复杂性，进而提出了一种自动识别关键类的方法。

节点重要性识别本质上是一个排序问题，对于软件系统中的排序任务，Srinivasan 等[28]提出了一种基于软件核心构件的排序模型。Pan 等[29]构建了一个基于软件网络的新型模型 ElementRank，利用多层复杂网络对软件模块进行排序。此外，Pan 等[30]还基于有向加权软件网络，提出了一个使用广义 k 核分解的方法，识别面向对象软件中的关键类，并验证了该方法在目标系统上关键类识别的召回率大于 64%。

上述研究均表明，软件系统（软件网络）中存在对整体拥有相对更大影响力的少数重要类（节点）。虽然已有很多方法用于排序网络中的节点重要性，但现实网络中包含节点的规模大，而且其拓扑结构复杂，导致很多方法的实用性存在一定的缺陷。如基于网络中节点位置的排序方法计算复杂度往往过高，不适用于大规模复杂网络。在深度学习的推动下，图神经网络被用于处理大规模图数据的神经网络结构，其强大的图结构表征能力，为探究一种适用于大规模且有效的节点重要性排序方法提供了新的视角。

3 理论基础

3.1 软件网络建模

本文选择以 Java 语言开发的软件系统作为分析对象，通过 Dependencyfinder[1)] 工具对源代码编译后产生的.class 文件进行解析，并抽取产生类之间依赖关系。本文旨在对软件系统中的关键类进行识别，所以仅构建类粒度级别的软件网络，即以类（class）为节点，类之间的关系为连边，具体的软件网络模型定义如下。

类级别软件网络（Class-level Software Network，CSN）定义为一个无向网络 $CSN=(V,E,W)$，其中节点 v_i（$v_i \in V$）代表软件系统的一个类（Class）或者接口（Interface），如果它们之间存在依赖关系，则存在一条连边 e_{ij}（$e_{ij}=e_{ji} \in E$），连边权重 w_{ij}（$w_{ij} \in W$）表示连边 e_{ij} 的权值。在 CSN 的建模过程中，本文类节点的依赖关系主要考虑以下 3 种：

（1）继承。假如类 i 与类 j 之间存在继承或接口实现关系，则它们对应的类节点 v_i 和 v_j 之间存在一条连边 e_{ij}。

（2）聚合。假如类 i 包含类 j 的属性，则它们对应的类节点 v_i 和 v_j 之间存在一条连边 e_{ij}。

（3）参数。假如类 i 中的方法调用了类 j 的方法，则它们对应的类节点 v_i 和 v_j 之间存在一条连边 e_{ij}。

使用类之间的依赖次数作为连接两个类节点边的权重，两个类之间每存在以上 3 种依赖关系的任意 1 种，则连接两个类节点的边权重增加 1，类节点之间的连边权重并不直接决定节点的重要性，而是代表了两个类之间的紧密程度。

3.2 图神经网络

图神经网络（Graph Neural Network，GNN）起源于 2005 年，其概念最早由 Gori 等[31]提出，并得到 Scarselli 等[32]和 Gallicchio 等[33]的进一步阐述。最早的 GNN 是基于巴拿赫不动点理论，定义图中的每个节点和边都拥有自己的特征向量。与图嵌入（graph embedding）技术相似，GNN 的目标就是获得每个节点相对于全图结构的隐藏特征，因此 GNN 在计算节点的隐藏特征时，结合了邻居节点的信息。为了能让每一个节点都结合到全图所有节点的特征，GNN 采用了迭代式更新的方式，在每一次迭代中，图中每个节点都会将自己邻居节点的特征、邻居节点在上次迭代之后的隐藏特征、与邻居节点间连边的特征和自身的特征通过一个复杂函数 f 相结合，作为本节点在此次迭代中更新的隐藏特征，复杂函数 f 对全图每一个节点通用，GNN 通过一个前馈神经网络来拟合函数 f。

这种迭代方式一方面使得图中每个节点都结合了邻居节点的特征，另一方面也将节点自身的特征传播给了邻居节点，而在下一次迭代中，节点的特征又会作为邻居节点隐藏特征的一部分传播给邻居的邻居。因此只需要通过最多不超过图的直径次的迭代，就能将自身特征传遍全图的每一个节点。这种迭代思想在后来的各种图神经网络变种和改进中也同样存在。最新的图神经网络已开始尝试运用于图生成，如图变分自编码器[34]和图对抗神经网络[35]等。

3.3 排序学习

一般来说，排序学习（Learning to Rank，LtR）算法是使用机器学习技术来解决排名问题的方法[36]，该算法在信息检索、自然语言处理与数据挖掘等领域应用广泛。排序学习的核心是学习一个排序模型 $f(q,d)$，其中 q 表示查询，d 表示文档，利用排序模型，在给定查询 q 时给出一个相关文档的合适排序。它的目标是最小化一个排序学习的损失函数 $R(F)$。LtR 算法可分为 3 种：单文档方法（Pointwise）、文档对方法（Pairwise）和文档列表方法（Leastwise）[37]。这 3 种方法以不同的方式对 LtR 过程进行建模，定义了各种输入和输出空间。

Pairwise 方法主要是将排序问题转为预测一对文档之间的相对顺序。这种方法只考虑了一对文档之间的偏序关系。常用的 Pairwise 方法有 RankNet[38]，RankBoost[39] 和 LambdaMART[40]。

4 基于图神经网络的关键类识别方法

本文尝试运用图神经网络来学习软件网络中节点的重要性，从而实现对软件系统中关键类的识别，如图 1 所示。首先，利用网络嵌入学习方法，对加权软件网络中的节点进行学习，得到节点的初始嵌入向量。然后，构建基于图神经网络的排序模型，采用邻域聚合的方式设计一个编码器 Encoder 和一个多层感知机的解码器 Decoder。以网络嵌入学习得到的

1) http://depfind.sourceforge.net/

向量为输入，编码器 Encoder 利用网络结构将每个节点进一步编码成一个特征向量，该特征向量可捕获节点的重要结构信息。解码器 Decoder 再将每个节点的特征向量进一步转化为标量，然后利用 Pairwise 排序学习，实现对网络中节点重要性的排序。最后，将训练得到的结果用于软件系统的关键类识别验证，并采用网络鲁棒性指标来评价关键类识别的精度。

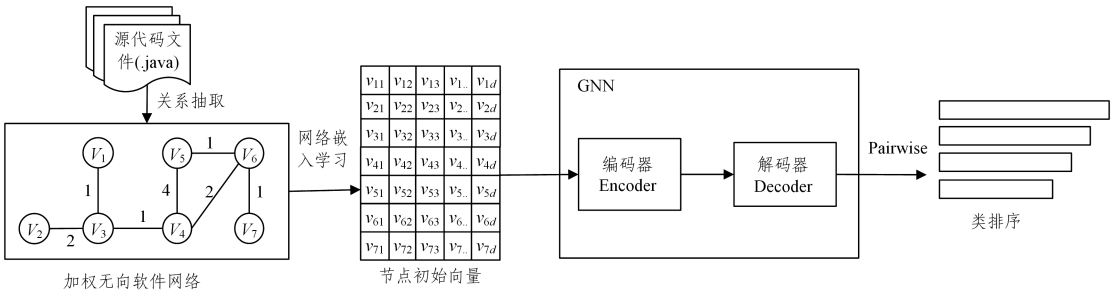


图 1 本文方法的整体框架

Fig. 1 Overall framework of the method in this paper

4.1 网络嵌入学习

图神经网络框架往往需要网络中的节点拥有自身的特征，如引文网络使用文章关键词作为节点特征、蛋白质网络使用原子性质作为节点特征等，本文讨论的软件网络的类节点本身不具备类似特征。网络嵌入(network embedding)可将一个网络映射到一个空间，并用一个低维的向量来表征网络中每个节点的结构信息。为了表征软件网络中节点的全局拓扑结构信息，本文引入网络嵌入学习对软件网络中的节点进行预处理，即将网络中每个节点 v 都转化为一个 d 维的表征向量 $\boldsymbol{X}_v(\boldsymbol{X}_v\in\mathbb{R}^d, d\ll|V|)$，将得到的表征向量作为初始特征来优化图神经网络模型效果。

在网络嵌入学习过程中，需要尽可能地保留网络全局结构信息。Node2vec 作为一个经典的网络嵌入学习方法，将网络中节点之间的连边关系通过随机游走转化为节点序列，并将这种序列类比为自然语言处理中的文本序列，然后利用 Skip-gram 方法来学习网络节点的表征向量。在游走过程中，引入参数返回概率 p 和远离概率 q 控制下一步游走的策略，令 u 为随机游走的起始节点，c_i 表示游走过程中第 i 步选择的游走节点，则有：

$$P(c_i=x\mid c_{i-1}=v\mid c_{i-2}=t)=\begin{cases}\dfrac{\alpha_{pq}(t,x)}{Z}, & (v,x)\in E\\ 0, & (v,x)\notin E\end{cases}\tag{1}$$

其中，Z 为标准化常数。$\alpha_{pq}(t,x)$为非标准化转移概率，通过参数 p 和 q 对随机游走进行引导，而$\alpha_{pq}(t,x)$定义为：

$$\alpha_{pq}(t,x)=\begin{cases}\dfrac{1}{p}, & d_{tx}=0\\ 1, & d_{tx}=1\\ \dfrac{1}{q}, & d_{tx}=2\end{cases}\tag{2}$$

其中，d_{tx}表示节点 t 和节点 x 的距离，$d_{tx}=0$ 表示节点 t 和节点 x 是同一个节点，$d_{tx}=1$ 表示节点 x 是节点 t 的邻居节点，$d_{tx}=2$ 表示节点 x 与节点 t 没有直接关系。当 p 较大且 q 较小时，随机游走偏向深度优先搜索；而当 p 较小且 q 较大时，随机游走偏向广度优先搜索。

网络嵌入学习的目标是给定所有节点，使相邻节点出现的概率最大，表示为：

$$\mathrm{Max}\Big(\sum_{u\in V}\log P(N(u)\mid\vec{\boldsymbol{u}})\Big)\tag{3}$$

其中，$N(u)\in V$ 为节点 u 的邻域节点集合，$\vec{\boldsymbol{u}}$ 为节点 u 的嵌入向量，由嵌入向量 $\vec{\boldsymbol{u}}$ 得到邻域 $N(u)$的概率为 $P(N(u)\mid\vec{\boldsymbol{u}})$，定义为：

$$P(N(u)\mid\vec{\boldsymbol{u}})=\prod_{n_i\in N(u)}P(n_i\mid\vec{\boldsymbol{u}})\tag{4}$$

在此基础上，对给定节点 u，生成节点n_i的条件概率可通过节点嵌入向量的内积进行 $SoftMax$ 运算得到：

$$P(n_i\mid\vec{\boldsymbol{u}})=\frac{\exp(\vec{\boldsymbol{n}_i}\cdot\vec{\boldsymbol{u}})}{\sum_{v\in V}\exp(\vec{\boldsymbol{v}}\cdot\vec{\boldsymbol{u}})}\tag{5}$$

最后可得到最终优化函数：

$$\mathrm{Max}\Big(\sum_{u\in V}\Big[-\log\sum_{v\in V}\exp(\vec{\boldsymbol{v}}\cdot\vec{\boldsymbol{u}})+\sum_{n_i\in N(u)}\vec{\boldsymbol{n}}_i\cdot\vec{\boldsymbol{u}}\Big]\Big)\tag{6}$$

通过 Node2vec 学习可得到每一个节点的嵌入向量，这些向量将作为图神经网络中编码器的输入。

4.2 基于图神经网络的排序模型

4.2.1 编码器建模

在 3.1 节软件网络 CSN 定义的基础上，令 $N(v)=\{u\in V\mid(v,u)\in E\}$表示节点 v 的直接邻居节点集，$\boldsymbol{X}_v\in\mathbb{R}^d$表示节点 v 通过网络嵌入学习获得的嵌入向量，d 为向量维度。$\boldsymbol{h}_v^{(k)}\in\mathbb{R}^d$表示目标节点 v 在模型第 k 层的隐含嵌入向量，d 为隐含嵌入向量的维度，初始时，令$\boldsymbol{h}_v^{(0)}=\boldsymbol{X}_v$。为了简化模型，本文保持不同层中隐含嵌入向量的维度 d 一样。

为了更好地提取网络中节点全局结构的隐藏特征，有必要将每个节点的特征与网络中其他关联节点的特征相结合，即网络中每个节点在编码迭代过程中都会聚合(aggregate)其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相结合(combine)，如图 2 所示，具体表示如下：

$$\boldsymbol{h}_{N(v)}^{(k)}=AGGREAGE(\{\boldsymbol{h}_u^{(k-1)};u\in N(v)\})\tag{7}$$

$$\boldsymbol{h}_v^{(k)}=\sigma COMBINE(\boldsymbol{h}_v^{(k-1)},\boldsymbol{h}_{N(v)}^{(k)})\tag{8}$$

其中，k 表示当前层；$\boldsymbol{h}_u^{(k-1)}$ 表示目标节点 v 的邻居节点 u 在 $k-1$ 层的嵌入向量；$\boldsymbol{h}_{N(v)}^{(k)}$表示目标节点 v 的所有邻居节点在 $k-1$ 层嵌入向量的聚合表示；$AGGREAGE$ 表示网络中局部节点的信息聚合函数；$\boldsymbol{h}_v^{(k)}$表示目标节点 v 在当前第 k 层的嵌入向量，是由目标节点 v 在 $k-1$ 层的嵌入向量$\boldsymbol{h}_v^{(k-1)}$和其邻居节点在当前层聚合向量$\boldsymbol{h}_{N(v)}^{(k)}$通过 $COMBINE$ 函数组合而成；σ 是一个激活函数，如 ReLU。

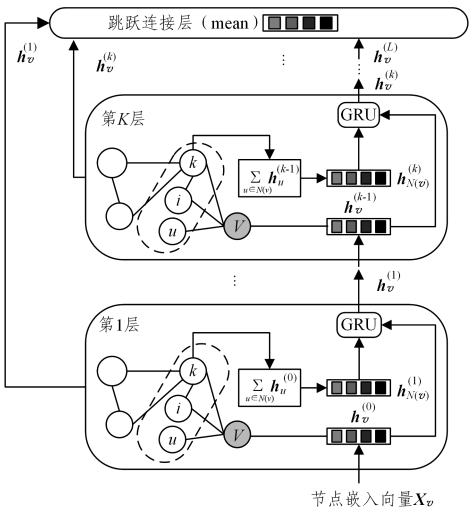


图 2 编码器结构

Fig. 2 Encoder structure

常见的 *AGGREAGE* 函数有求和（sum）[41]、平均值（mean）和最大值（max）[42]等，这些方法通常不考虑节点连边的权值，而本文使用的软件网络为加权网络，因此在邻居节点嵌入向量聚合过程中，本文提出了一种改进的加权平均方法来进行邻居节点的聚合处理，其定义为：

$$\boldsymbol{h}_{N(v)}^{(k)}=\sum_{u\in N(v)}\frac{\boldsymbol{w}_{vu}}{\sqrt{\hat{\boldsymbol{d}}_v}\sqrt{\hat{\boldsymbol{d}}_u}}\boldsymbol{h}_u^{(k-1)} \tag{9}$$

其中，$\hat{\boldsymbol{d}}_v$和$\hat{\boldsymbol{d}}_u$分别为节点 v 和 u 的加权度，即节点的所有连边的权值之和，表示为$\hat{\boldsymbol{d}}_v=\sum_{u\in N(v)}\boldsymbol{w}_{vu}$；$\boldsymbol{h}_u^{(k-1)}$为节点 v 的邻居节点 u 在第 $k-1$ 层的嵌入向量。使用这种聚合方法时，节点会聚合更多拥有高权值连边的邻居节点的特征，最终节点的特征向量更加接近其拥有高权值连边的邻居节点。这与本文对于软件网络边权的定义一致，即节点间的连边拥有高权值，这说明节点间联系的紧密程度更高。

COMBINE 函数负责组合目标节点的邻居节点在前一层嵌入向量的聚合结果与目标节点自身在前一层生成的嵌入向量，从而得到节点在本层的新嵌入向量。常用函数有求和[41]、拼接[42]和门控循环单位（Gated Recurrent Units，GRU）[43]等。在此本文采取 GRU 作为组合函数，即以节点 v 的邻居节点在第 k 层的聚合向量$\boldsymbol{h}_{N(v)}^{(k)}$为输入，而节点 v 自身在第 $k-1$ 层的嵌入向量$\boldsymbol{h}_v^{(k-1)}$为隐含状态，于是节点 v 在第 k 层的新嵌入向量可表示为：

$$\boldsymbol{h}_v^{(k)}=\sigma(\boldsymbol{W}\cdot CONCAT(\boldsymbol{h}_v^{(k-1)},\boldsymbol{h}_{N(v)}^{(k)})) \tag{10}$$

如图 2 所示，在编码器的最顶端设置了一个跳跃连接层[44]，将模型前 L 层中每一层产生的特征向量都连接到跳跃连接层进行聚合，聚合方式可以选择最大池化、拼接或递归神经网络 LSTM 等。本文在此选择对每一个节点在每一层产生的特征向量取均值作为节点最终的特征向量，因此跳跃连接层相当于一个平均池化操作的聚合层，该操作简单且不会引入任何参数学习任务。设置跳跃连接层的主要目的是缓解由节点在网络中位置的不同导致邻域聚合范围差异对邻域聚合效果造成的影响。因此，最后节点 v 的特征向量可表示为：

$$\boldsymbol{z}_v=mean(\boldsymbol{h}_v^{(1)},\boldsymbol{h}_v^{(2)},\boldsymbol{h}_v^{(3)},\cdots,\boldsymbol{h}_v^{(L)}) \tag{11}$$

详细的编码过程如算法 1 所示。

算法 1 编码器编码过程

输入：软件网络 CSN=(V,E)，节点的初始特征向量$\mathbf{X}_v\in\mathbb{R}^d$，迭代层数 L，权重矩阵 $\mathbf{W}\in\mathbb{R}^{d\times d}$。

输出：每个节点的特征向量 $\mathbf{Z}$。

1. 初始化节点的嵌入向量$\mathbf{h}_v^{(0)}=\mathbf{X}_v$；
2. 如果 k 小于或等于 L；
3. 对每个节点 v 进行式(9)的邻居节点聚合处理；
4. 对每个节点 v 进行式(10)的组合处理；
5. 如果 k 大于 L；
6. 对每个节点 v 进行式(11)的平均池化处理，得到节点的最终嵌入向量$\mathbf{z}_v$；
7. 输出$\mathbf{z}_v$。

4.2.2 解码器建模

解码器 Decoder 的主要功能是将每个节点的最终特征向量进一步转化为标量，其由一个多层感知机构成。多层感知机（Multilayer Perceptron，MLP）也称人工神经网络，一个 MLP 中除了最顶层的输入层和最后一层的输出层外，中间可以有多个隐藏层。本文采用包含一个隐藏层的简单 MLP，即三层网络结构，如图 3 所示。

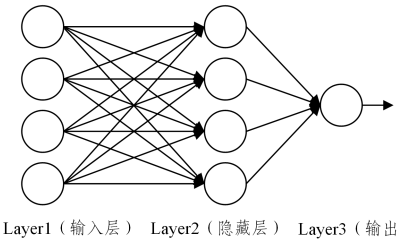


图 3 三层感知机

Fig. 3 Three layers of perceptron

以编码器输出的节点嵌入向量 $\boldsymbol{Z}$ 为输入，隐藏层与输入层通过全连接，输出为 $f(\boldsymbol{WZ}+\boldsymbol{b})$，$\boldsymbol{W}$ 为连接系数，$\boldsymbol{b}$ 是偏置，函数 f 可以是常用的 sigmoid 函数、tanh 函数和 ReLU 函数。近年来，ReLU 函数的改进方法 LeakyReLU 函数得到了广泛应用，该函数对负值的输入依然具有较小的梯度，可以缓解梯度消失等问题。因此，本文选用 LeakyReLU 作为激活函数，最后节点转为标量y_v，表示如下：

$$y_v=LeakyReLU(\boldsymbol{WZ}_v+\boldsymbol{b}) \tag{12}$$

4.3 损失函数

为了训练编码器中的聚合和组合参数，以及解码器中的连接系数和偏置，需要尽可能使模型输出的节点标量值排序与采用节点收缩法得到的节点对整个网络鲁棒性的影响力排序一致。本文采用 Pairwise 排序学习，对于节点对(v_i,v_j)，假如它们的真实影响力分别为I_i和I_j，而模型学习到的值分别为y_i和y_j，则本文需要$y_{ij}=y_i-y_j$的相对排序顺序与$I_{ij}=I_i-I_j$保持一致，可通过二元交叉熵代价函数$C_{i,j}$来推断y_{ij}，具体公式为：

$$C_{i,j}=-g(I_{ij})*\log\sigma(y_{ij})-(1-g(I_{ij}))*\log(1-\sigma(y_{ij})) \tag{13}$$

$$Loss=\sum_{i,j\in V}C_{i,j} \tag{14}$$

其中，$g(x)=1/(1+e^{-x})$，以确保排序损失函数 $Loss$ 最小。

5 实验与结果分析

5.1 实验环境

本文软件网络建模的部分在 Window 10 系统的 Java 环境下完成(JDK 1.9),其余部分的实验都是在租用的云服务器上进行的,使用 Linux 系统(Red Hat 4.8.5-28)32GB 内存和 4 个 11GB 显存的 GTX1080ti 型号 GPU。图神经网络模型用 Adam 优化器在 TensorFlow 框架[1)] 实现,模型部分参数设置如表 1 所列。

表 1 部分模型的参数设置

Table 1 Part of model parameter setting

参数名称	参数值
学习率 γ	0.0002
嵌入向量维度 d	128
脱离概率 q	3
批训练大小 batch-size	8
平均节点抽样次数	5
解码器中聚合层数 L	5

5.2 实验数据

本文选取两组开源项目数据集,如表 2 所列。第一组数据来自软件项目 Maven 和 Vuze,这两个数据集中没有专家标注的关键类信息。第二组数据来自互联网公开的 Ant-1.6.1 和 JMeter-2.0.1 数据集,该数据中含有专家已标注好的关键类信息,且已在文献[26,30,45]中被采用,其中 Ant 中拥有 10 个专家标注的关键类,JMeter 中拥有 14 个专家标注的关键类。

表 2 本文实验的数据集信息

Table 2 Experimental data set information in this paper

数据	名称	标注	版本号	节点数	连边数
第一组	Maven	无	4.5.0.2	4736	17094
	Vuze	无	3.2.2	7662	39732
第二组	Ant	有	1.6.1	664	2640
	JMeter	有	2.0.1	277	1058

5.3 实验设置

5.3.1 评价指标

在复杂网络中,常用来评价网络节点影响力大小的标准有两种,一种是基于网络的鲁棒性和脆弱性,另一种是基于网络的传播动力学模型。评价思路是将排序得到的 Top-k%个节点作为研究对象,通过考查这些节点对网络结构和功能及对其他节点的影响大小来进行评价。

本文选用基于网络鲁棒性的指标来评价各方法识别关键类的效果。将网络中所有节点按照重要性计算方法的计算结果进行降序排序,然后将前 k%节点从网络中移除,并计算剩余网络中连通子图的数量、最大连通子图[46]中的节点数量比例,以及网络中节点对的效率。即对于规模为 N 的网络 $\mathcal{N}$,用 $\mathcal{N}_k$ 表示从网络中删除前 k%节点后得到的剩余网络,$\mathcal{N}_k^{\max}$ 代表 $\mathcal{N}_k$ 网络中的最大连通子图,$|\mathcal{N}_k^{\max}|$ 为其包含的节点数,$\sum\mathcal{N}_k^i$ 为剩余网络 $\mathcal{N}_k$ 中的连通子图数,$\mathcal{N}_k^i$ 为第 i 个连通子图。因此网络中移除前 k%节点后,剩余网络中连通子图的数量 $\sum\mathcal{N}_k^i$ 越多,最大连通子图的节点数量 $|\mathcal{N}_k^{\max}|$ 就越小,表示删除的节点集对网络的鲁棒性影响越大,即这些节点对于网络的重要性越大。

网络中节点对的效率是节点之间最短路径的乘法逆。网络的平均全局连通效率是所有节点对的平均效率,表示为:

$$\varepsilon=\frac{1}{N(N-1)}\sum_{i\neq j}\frac{1}{d_{ij}} \tag{15}$$

其中,ε 表示网络的全局连通效率,$\varepsilon\in[0,1]$,当网络中的节点全部不连通时,ε 为 0。当网络中有且只有一对连通的两个节点时,ε 为 1。本文使用删除网络一定节点后剩余节点的连通效率和网络最初的连通效率的比值作为评估指标。该比值越高,说明剩余的节点间连通能力越弱,代表网络被破坏的程度越严重,即删除的节点集对网络鲁棒性的影响越大。

对于第二组已提供关键类标注信息的数据集,本文采用普遍使用的召回率(recall)和准确率(precision)为评价指标,其中召回率和准确率定义为:

$$recall=\frac{TP}{TP+FN} \tag{16}$$

$$precision=\frac{TP}{TP+FP} \tag{17}$$

其中,TP 表示某种识别方法所识别出的关键类也同样是专家标记的关键类的数量,即识别方法成功识别出关键类的数量;FN 为专家标记的关键类集合中未被识别方法识别出的关键类数量;$TP+FN$ 等同于专家标记出的关键类的数量;FP 表示某种识别方法所识别出的关键类不在专家标记的关键类集合中的数量;$TP+FP$ 等同于识别方法识别出的关键类数量。

5.3.2 基准方法

为了验证本文提出的方法,在第一组数据集实验中,选取 5 种经典的复杂网络中节点重要性识别方法作为基准,具体如下。

(1)介数中心性(Betweenness Centrality,BC)[27]:根据网络中所有节点对的最短路径通过某一个节点的比例来表示该个节点在网络中的重要性,该指标刻画了节点在网络中的"中介"作用。

(2)PageRank[18]:是 Web 页面排序中的一个著名算法,由 google 公司提出。其基本思想是一个页面的重要性会受链接到它的其他页面的重要性影响,表现为页面 B 从页面 A 链接而来,如果 A 越重要,B 的重要性就会偏大。

(3)节点收缩法 IMC[5]:本文在模型训练过程中采用该方法作为网络节点重要性的 ground truth 值,在此我们仍希望对比在真实网络中,本文模型与该方法计算的节点重要性排序。

(4)接近中心性(Closeness):一个节点 v 的接近中心性是指节点 v 到其他节点的最短路径的平均长度。也就是说,对于一个节点而言,它距离其他节点越近,那么它的中心度就越高。

(5)K-Core 分解[16]:其主要思想是将外围的节点逐层移去,使处于内核的节点拥有更高的影响力,即通过逐步移去网

1) https://tensorflow.google.cn/

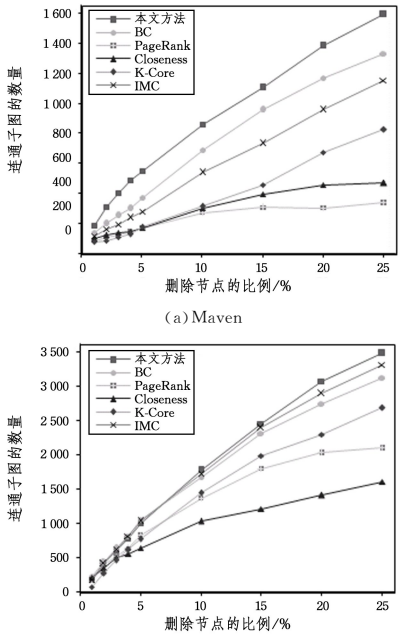
络中所有度值小于或等于 k 的节点，直到网络中没有节点为止。

第二组实验选用两种已有的软件系统关键类识别方法和文献[47]中所使用的模型进行对比分析。其中文献[45]使用 PageRank 方法进行软件网络重要节点识别，文献[30]则引入了加权 K-Core 分解法进行软件网络重要节点识别。文献[30,45,47]中的方法都基于复杂软件网络来识别软件系统关键类，具有一定的代表性。

5.4 实验结果

5.4.1 无标注下的网络鲁棒性对比分析

如图 4 所示，对于第一组实验的软件 Maven，在删除一定比例的排序靠前的节点后，本文方法的剩余网络中连通子图数量明显比其他 5 种方法多，即删除的节点对网络结构的影响更大。对于软件 Vuze，删除前 5%的节点时，本文方法与 BC 和 IMC 两种方法得到的连通子图数比较接近，比其他 3 种方法得到的连通子图数量要多。但随着删除节点比例的增大，本文方法形成的连通子图数量开始变得更多。



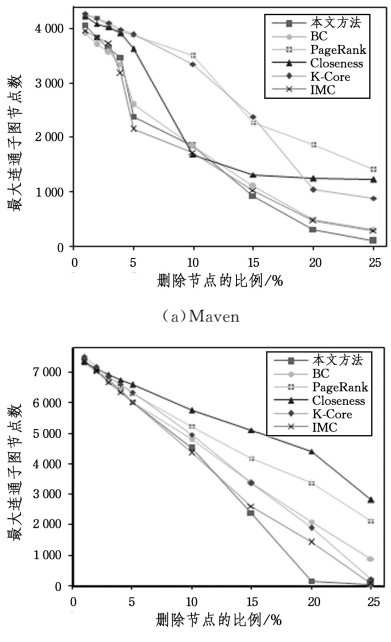
(a) Maven

(b) Vuze

图 4 剩余网络中连通子图数量

Fig. 4 Number of connected subgraphs in the remaining network

剩余网络中最大连通子图的节点数，如图 5 所示，在软件 Maven 上删除前 5%的节点时，本文方法与 BC 和 IMC 两种方法的最大连通子图的节点数大致接近，且均比其他 3 种方法的节点数要少；而在 Vuze 上，各方法的最大连通子图的节点数衰减幅度较为相当，结果分布较为密集。整体上，随着删除节点比例的增大，最大连通子图的节点数衰减幅度比较明显。在软件 Maven 中，本文方法的最大连通子图的节点数衰减的幅度最大，尤其是删除比例超过 15%时。在软件 Vuze 中，在删除前 15%的节点时，本文方法与节点收缩法的最大连通子图的节点数大致接近，其中在删除比例为 15%～20%时，本文方法的节点数呈明显的大幅度衰减趋势，最终在删除比例为 25%时，拥有超过 7000 个节点的 Vuze 软件网络的最大连通子图仅剩 39 个节点。



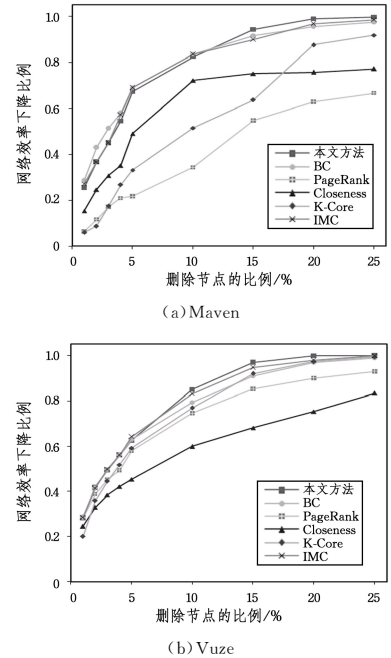
(a) Maven

(b) Vuze

图 5 剩余网络中最大连通子图节点数

Fig. 5 Maximum number of connected subgraph nodes in the remaining network

对于网络效率下降比例，图 6 与图 5 的结果整体上比较类似，在软件 Maven 中，在删除前 15%的节点之前，本文方法与 BC 和 IMC 两种方法的结果比较相似，随后本文方法的网络效率下降比例开始变得愈加突出。在软件 Vuze 中，本文方法与 IMC 的效果很相似，在删除比例为 15%～20%时，两者的网络效率下降比例差异有所增加。



(a) Maven

(b) Vuze

图 6 网络效率下降比例

Fig. 6 Ratio of network efficiency degradation

总的来说，无论从连通子图数、最大连通子图节点数，还是网络效率下降比例的角度对比，本文提出的基于图神经网络的关键类识别方法的效果都更占优势，且在 Maven

软件中优势更加明显。

5.4.2 有标注下的识别准确性对比分析

本文进一步采用带有专家标注信息的数据集进行实验验证，结果如表3所列。在前10%，15%的关键类识别中，对于Ant软件，本文方法比已有的3种方法的召回率和准确率都有所提高，在前10%的关键类识别中其召回率和准确率分别为80%和12.12%，在前15%的关键类识别中其召回率和准确率分别为100%和10%，且在前15%的关键类识别中，本文方法识别出了专家标注的全部关键类。相比文献[45]、文献[30]、文献[47]而言，本文方法的召回率分别提高了42%，42%和25%。对于JMeter软件，在前10%的关键类中，本文方法的召回率与准确率和文献[30]，文献[47]一样，分别为64.29%和32.14%，但相比文献[45]有所提高。在对比前15%的关键类识别时，本文方法的召回率和准确率都是最高，分别为78.57%和27.18%。相比文献[45]，本文方法的召回率和准确率分别提高了57.1%和63.7%，相比文献[30]和文献[47]，召回率和准确率都分别提高了10%和14.2%。

表3 各种方法的召回率、准确率比较

Table 3 Comparison of recall rate and accuracy of various methods

（单位：%）

k	方法	Ant		JMeter	
		recall	*precision*	*recall*	*precision*
10%	本文方法	80	12.12	64.29	32.14
	文献[45]	60	9.09	35.71	18.05
	文献[30]	70	10.54	64.29	32.14
	文献[47]	70	10.54	64.29	32.14
15%	本文方法	100	10	78.57	27.18
	文献[45]	70	7	50	16.6
	文献[30]	70	7	71.43	23.80
	文献[47]	80	8	71.43	23.80

表4和表5分别给出了Ant和JMeter两个软件中前10%，15%关键类在3种方法中的识别情况。

根据上述实验结果，整体上本文方法对两个软件的关键类识别上，召回率和准确率效果与已有文献中的方法具有可比性，且在前15%的情况下，效果明显更好。

总体而言，从两组实验数据集的结果中可以看出，相比已有的方法和工作，本文基于图神经网络的关键类识别方法在识别效果上有较好的改善。这说明通过图神经网络学习，可以获取网络中节点更丰富的结构信息，从而可以更好地指导软件工程实践工作。

表4 Ant中各类方法识别出的关键类情况

Table 4 Key classes identified by various methods in Ant

关键类	Top-10%				Top-15%			
	本文方法	文献[45]	文献[30]	文献[47]	本文方法	文献[45]	文献[30]	文献[47]
Project	√	√	√	√	√	√	√	√
UnknownElement	√	√	√	√	√	√	√	√
Task	√	√	√	√	√	√	√	√
Main	×	×	×	×	√	×	×	×
IntrospectionHelper	√	√	√	√	√	√	√	√
ProjectHelper	√	×	√	√	√	×	√	√
RuntimeConfiguration	√	√	√	√	√	√	√	√
Target	√	√	√	√	√	√	√	√
ElementHandler	×	×	×	×	√	×	×	×
TaskContainer	√	×	×	×	√	√	×	√

表5 JMeter中各类方法识别出的关键类情况

Table 5 Key classes identified by various methods in JMeter

关键类	Top-10%				Top-15%			
	本文方法	文献[45]	文献[30]	文献[47]	本文方法	文献[45]	文献[30]	文献[47]
AbstractAction	√	√	×	√	√	√	×	√
JMeterEngine	×	×	×	×	×	×	×	×
JMeterTreeModel	√	√	√	√	√	√	√	√
JMeterThread	√	×	√	×	√	×	√	×
JMeterGUIComponent	√	√	√	√	√	√	√	√
PreCompiler	×	×	×	×	√	×	×	√
Sampler	√	√	√	√	√	√	√	√
SampleResult	√	×	√	√	√	√	√	√
TestCompiler	√	×	√	√	√	×	√	√
TestElement	√	√	√	√	√	√	√	√
TestListener	×	×	×	×	×	√	√	×
TestPlan	√	×	√	√	√	×	√	√
TestPlanGui	×	×	×	×	×	×	×	×
ThreadGroup	×	×	√	√	√	×	√	√

5.5 实验结果的影响因素分析

在生成初始网络嵌入向量的过程中，控制不同节点间的随机游走概率参数 p 和 q 决定了初始网络嵌入所表达的特性。为了更好地学习网络的全局结构信息，本文使用偏向广度优先搜索的随机游走方式，设置 $p=1$，$q>1$，参数 q 取值越大，越偏向在初始节点周围游走。图7给出了 q 取不同值时本文方法在4个实验软件网络上的准确度。可以看出，在 q 取值为3时，本文方法均可以取得最好效果，这说明在 $p=1$，$q=3$ 的参数组合下，初始网络嵌入可以较好地捕捉软件网络中的结构特性。

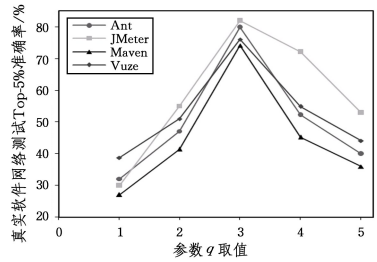


图7 参数 q 在不同取值下的准确率

Fig. 7 Accuracy of parameter q under different values

结束语 针对软件网络中的关键类识别问题，本文尝试将图神经网络应用于自动学习软件网络的节点特征，并将学习结果应用于软件系统的关键类识别研究，代替已往建立在复杂网络理论上，基于人工构建度量指标的节点重要性排序方法。此外，使用无监督网络嵌入方法对软件网络节点进行初步表征学习，解决了软件网络中缺乏节点特征的问题。

研究结果表明：(1)通过与常用的介数中心性、K-Core、接近中心性、节点收缩法以及PageRank等方法相比，从网络鲁棒性角度评价，删除本文方法识别的关键节点集对网络造成的破坏更明显，验证了本文方法识别软件网络关键节点的有效性，因此该方法可以作为提高软件系统中关键类识别的一个较好选择。

(2)本文基于图神经网络的关键类识别方法在已有公开标注数据集上，相比已有方法，本文方法识别前15%的关键节点的召回率和准确率效果更好，其中召回率和准确率的提

高幅度均在10%以上。

总体而言，相比已有方法，本文基于图神经网络的关键类识别方法的准确率较高，并且具有较好的稳定性，在规模较大的软件上表现更好。尽管如此，目前本文只是在最简单的软件网络模型上运用一种图神经网络进行学习，在后续工作中仍有很多改进的空间，改进的方面包括：

(1)丰富用于实验验证的软件系统，目前主要是选取基于Java开发的4款软件，后续工作中可以扩展到C家族语言开发的软件系统或其他Web语言开发的软件系统等。

(2)进一步改进软件网络模型，例如，在建模过程中，考虑类节点之间连边的依赖方向；对类节点之间的各种关系进行区别，并采取不同的权重计算方式，比较分析各种关系在学习过程中的影响差异；在更多粒度上构建软件网络模型，如模块层、包层和方法层。

(3)引入更多的图神经网络模型，如图卷积网络、图注意力网络和图生成网络等，并用实验进行对比分析，检验这一改进方法是否能够进一步提高关键类的识别准确率。

参 考 文 献

[1] PAN W, LI B, MA Y, et al. Measuring structural quality of object-oriented softwares via bug propagation analysis on weighted software networks[J]. Journal of Computer Science and Technology, 2010, 25(6): 1202-1213.

[2] MAGGIE H, KATERINA G P. Common trends in software fault and failure data[J]. IEEE Transactions on Software Engineering, 2009, 35(4): 484-496.

[3] DAI J Y, WANG B, SHENG J F, et al. Identifying influential nodes in complex networks based on local neighbor contribution [J]. IEEE Access, 2019, DOI: 10. 1109/ACCESS. 2019. 2939804.

[4] RUAN Y R, LAO S Y, WANG J D, et al. Node importance measurement based on neighborhood similarity in complex network[J]. Journal of Physics, 2017, 66(3): 1-8.

[5] TAN Y J, WU J, DENG H Z. Evaluation Method for Node Importance base on Node Contractionin Complex Networks[J]. Systems Engineering Theory and Practice, 2006, 26(11): 79-83.

[6] BIAN T, HU J T, DENG Y. Identifying influential nodes in complex networks based on AHP[J]. Physica A Statistical Mechanics and its Applications, 2017, 479: 422-436.

[7] WATTS D J, STROGATZ S H. Collective dynamics of 'small-world' networks[J]. Nature, 1998, 393: 440-442.

[8] BARABÁSI A L, ALBERT R. Emergence of scaling in random networks[J]. Science, 1999, 286: 509-512.

[9] CONCAS G, LOCCI M F, PINNA S, et al. Fractal dimension in software networks[J]. Europhysics Letters, 2006, 76(6): 1221-1227.

[10] WANG X F, LI X, CHEN G R. Complex Network Theory and Its Application[M]. Beijing: Tsinghua University Press, 2006.

[11] HE K Q, MA Y T, LIU J, et al. Software Network[M]. Beijing: Science Press, 2008.

[12] MYERS C R. Software systems as complex networks: Structure, function, and evolvability of software collaboration graphs [J]. Phys. Rev. E, 2003, 68: 46116.

[13] PAN W, LI B, MA Y, et al. Class structure refactoring of object-oriented softwares using community detection in dependency [J]. Networks Frontiers of Computer Science in China, 2009, 3(3): 396-404.

[14] VALVERDE S, CANCHO R F, SOLÉ R V. Scale free networks from optimal design[J]. Europhysics Letters, 2002, 60(4): 512-517.

[15] OPSAHL T, AGNEESSENS F, SKVORETZ J. Node centrality in weighted networks: generalizing degree and shortest paths [J]. Social Networks, 2010, 32(3): 245-251.

[16] KITSAK M, GALLOS L K, HAVLIN S, et al. Identification of influential spreaders in complex networks[J]. Nature Physics, 2010, 6(11): 888-893.

[17] LI P X, RENG Y Q, XI Y M. An Importance Measure of Actors (Set) within a Network[J]. Systems Engineering, 2004, 22: 13-20.

[18] BRIN S, PAGE L. The anatomy of a large-scale hypertextual web search engine[J]. Computer Networks, 2012, 56(18): 3825-3833.

[19] KLEINBERG J M. Authoritative sources in a hyperlinked environment[J]. Journal of the ACM, 1999, 46(5): 604-632.

[20] WANG M, PAN W. A comparative study of network centrality metrics in identifying key classes in software[J]. Journal of Computational Information Systems, 2012, 8(24): 10205-10212.

[21] PERIN F, RENGGLI L, RESSIA J. Ranking software artifacts [C]// Proceedings of 4th Workshop on FAMIX and Moose in Software Reengineering(FAMOO-Sr'10). 2010: 1-4.

[22] WANG M S, LU H M. Identifying Key Classes Using h-Index and its Variants[J]. Computer Science and Exploration, 2011, 5(10): 891-903.

[23] HU S W, LI B, HE P, et al. Approach Based h-index to Measuring the Important Classes in Software Network[J]. Small Microcomputer System, 2017(2): 249-253.

[24] WANG Y, YU H, ZHU Z L. A Class Integration Test Order Method Based on the Node Importance of Software[J]. Computer Research and Development, 2016, 53(3): 17-30.

[25] LI D W, LI B, HE P, et al. Ranking the importance of classes via software structural analysis[J]. Future Communication, Computing, Control and Management, 2012, 141: 441-449.

[26] JIANG S J, JU X L, WANG X Y, et al. Measuring the Importance of Classes Using UIO Sequence[J]. Electronic Journals, 2015, 43(10): 2062-2068.

[27] WANG J, AI J, YANG Y, et al. Identifying key classes of object-oriented software based on software complex network [C]// International Conference on System Reliability & Safety. IEEE, 2017: 444-449.

[28] SRINIVASAN S M, SANGWAN R S, NEILL C J. On the measures for ranking software components[J]. Innovations in Systems and Software Engineering, 2017, 13: 161-175.

[29] PAN W F, MING H, CARL K, et al. ElementRank: Ranking Java software classes and packages using multilayer complex network-based approach[C]// IEEE Transactions on Software Engineering. 2019.

[30] PAN W F, SONG B B, LI K S, et al. Identifying key classes in

object-oriented software using generalized k-core decomposition [J]. Future Generation Computer Systems, 2018, 81: 188-202.

[31] GORI M, MONFARDINI G, SCARSELLI F. A new model for learning in graph domains[C]// Proc. of IJCNN. IEEE, 2005: 729-734.

[32] SCARSELLI F, GORI M, TSOI A C, et al. The graph neural network model[J]. IEEE Transactions on Neural Networks, 2009, 20(1): 61-80.

[33] GALLICCHIO C, MICHELI A. Graph echo state networks [C]// IJCNN. IEEE, 2010: 1-8.

[34] SIMONOVSKY M, KOMODAKIS N. Graphvae: Towards generation of small graphs using variational autoencoders[C]// International Conference on Artificial Neural Networks. 2018: 412-422.

[35] BOJCHEVSKI A, SHCHUR O, ZUGNER D, et al. Netgan: Generating graphs via random walks[C]// Proceedings of the 35th Int. Conference on Machine Learning. 2018: 610-619.

[36] LIU T Y. Learning to Rank for Information Retrieval[M]. Berlin: Springer, 2011.

[37] SHI Z, KEUNG J, BENNIN K E, et al. Comparing learning to rank techniques in hybrid bug localization [J]. Applied Soft Computing, 2018, 62: 636-648.

[38] BURGES C, SHAKED T, RENSHAW E, et al. Learning to rank using gradient descent[C]// Proceedings of the 22Nd International Conference on Machine Learning. Bonn, Germany, 2005: 89-96.

[39] FREUND Y, IYER R, SCHAPIRE R E, et al. An efficient boosting algorithm for combining preferences[J]. Journal of Machine Learning Research, 2004, 4(6): 170-178.

[40] WU Q, BURGES C C, SVORE K, et al. Adapting boosting for information retrieval measures[J]. Inf. Retrieval. 2010, 13(3): 254-270.

[41] KIPF T N, WELLING M. Semi-supervised classification with graph convolutional networks[C]// Proc. of International Conference on Learning Representations. Toulon, France, 2017, arXiv: 1609.02907v4.

[42] HAMILTON W L, YING R, LESKOVEC J. Inductive representation learning on large graphs[C]// 31st Conference on Neural Information Processing Systems(NIPS 2017). Long Beach, CA, USA, 2017: 1-11.

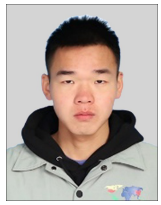
[43] LI Y, TARLOW D, BROCKSCHMIDT M, et al. Gated graph sequence neural networks[J]. arXiv: 1511.05493, 2015.

[44] XU K, LI C T, TIAN Y L, et al. Representation learning on graphs with jumping knowledge networks[C]// Proceedings of the 35th International Conference on Machine Learning. Stockholm, Sweden, 2018: 5449-5458.

[45] ZAIDMAN A, DEMEYER S. Automatic identification of key classes in a software system using web mining techniques[J]. Journal of Software Maintenance and Evolution: Research and Practice, 2008, 20(6): 387-417.

[46] IYER S, KILLINGBACK T, SUNDARAM B, et al. Attack robustness and centrality of complex networks[J]. PLoS ONE, 2013, 8(4): e59613.

[47] FAN C, ZENG L, DING Y, et al. Learning to identify high betweenness centrality nodes from scratch: a novel graph neural network approach[J]. arXiv: 1905.10418v1.



ZHANG Jian-xiong, born in 1998, postgraduate. His main research interests include network embedding, neural network and software network.

HE Peng, born in 1988, Ph.D, associate professor, postgraduate supervisor, is a member of China Computer Federation. His main research interests include software engineering and complex networks.