文章编号:1671-4598(2024)08-0243-07 DOI:10.16526/j.cnki.11-4762/tp.2024.08.035 中图分类号:TP273 文献标识码:A

基于图可解释网络的软件错误定位

邹凯胜[1,2]，周世健[1]，樊 鑫[1,2]
(1.南昌航空大学 软件学院，南昌 330038；2.南昌航空大学 软件测评中心，南昌 330038)

摘要：软件错误定位技术旨在通过挖掘程序与测试用例执行数据，提升定位准确性；针对SBFL技术过于依赖二进制覆盖信息的问题，提出一种基于图可解释网络的软件错误定位方法，将测试执行转化为图结构，利用图注意网络建模深度挖掘代码片段隐含的信息及其相互关系，并采用强化学习思想对图注意力网络学习后的决策过程进行解释，从而确定关键节点，缩小错误定位范围；实验的场景设立在Defects4j数据集的5个项目进行，并与SBFL及未经过解释的深度学习方法进行了对比；结果显示，基于图可解释网络的定位方法在Top-1、Top-3和Top-5指标上分别提升了7.26%、7.56%和9.96%，EXAM指数也提升了8.98%，显著优于其他方法。

关键词：软件测试；错误定位程序谱；图注意力网络；可解释模型

Software Fault Localization Based on Graph Interpretable Networks

WU Kaisheng[1,2], ZHOU Shijian[1], FAN Xin[1,2]
(1. School of Software, Nanchang Hangkong University, Nanchang 330038, China;
2. Software Testing and Evaluation Center, Nanchang Hangkong University, Nanchang 330038, China)

Abstract: Software fault localization aims to improve the accuracy of localization by mining program and execution data of test cases. To address the issue that spectrum-based fault localization (SBFL) technology relies too much on binary coverage information, a software fault localization method based on graph-interpretable networks is proposed. This method transforms test execution into a graph structure, utilizes the graph attention network modeling to deeply mine the implicit information and interrelationships among code segments. The reinforcement learning principles are used to explain the decision-making process after the graph attention network learning, thereby determine the key nodes and reduce the fault localization range. The experiments are conducted on five projects from the Defects4j dataset and compared with the SBFL and uninterpreted deep learning methods. The results show that based on graph-interpretable networks, the localization method improves the Top-1, Top-3, and Top-5 indicators by 7.26%, 7.56%, and 9.96%, respectively, and the EXAM index also increases by 8.98%, significantly outperforming other methods.

Keywords: software testing; fault localization; program spectrum; graph attention network; interpretable model

0 引言

ChatGPT的问世掀起了国内外科技工作爱好者对大语言模型的研究热潮，为推动人类科技发展作出了贡献。随着科技产业的蓬勃发展，人类活动对软件的依赖日益加深，因此，保证软件的正常运行显得尤其重要。软件调试是软件质量保证的重要手段，其中包括错误检测，错误定位和错误修复[1]。软件错误定位是调试过程中至关重要的一环，它可以帮助开发人员更快、更准确地定位程序中的错误，从而提高软件的质量和可靠性。

在调试过程中，错误定位是最耗时的活动之一[2]，也是修复程序错误的先决条件，这激发了许多关于自动错误定位技术的研究，例如基于程序频谱的错误定位技术[3-4]，基于突变的程序错误定位技术[5-6]，基于切片的程序定位技术[7]，以及基于机器学习方法的错误定位技术[8-9]等。在上述的研究方法中，基于程序频谱的错误定位技术（SBFL，spectrum-based fault localization），由于其轻量性、可扩展性和适用性被称作为使用最广泛的程序错误定位技术[10]。

SBFL的定位效果受可疑度值计算公式的影响，多种有效的计算公式如Jaccard[11]、Tarantula[12]、Ochiai[13]、OP2[14]和Dstar[15]已被设计出以提升SBFL的有效性。然而，还没有任何一种公式能够完美适用于所有的测试程序[16]。实际上，基于覆盖信息的故障定位技术的基本思想是，如果一条语句被更多的失败测试用例和更少的通过测试用例执行，则该语句应被赋予更高的可疑度值。一些研究者尝试将程序谱的种类扩展为程序变量谱、方法调用谱、时间频谱和基于命中的频谱[17]，但主流仍是以覆盖信息作为研究的度量手段，不足以反映程序中的错误传播机制，这会限制错误定位的有效性[18]。在我们之前的研究中也已经证实了[19]，对程序动态运行时的特征进行挖掘可以有效

收稿日期:2024-03-05； **修回日期**:2024-03-19。
基金项目:国家自然科学基金地区基金项目(42261070)。
作者简介:邹凯胜(1998-),男,硕士。
引用格式:邹凯胜,周世健,樊 鑫.基于图可解释网络的软件错误定位[J].计算机测量与控制,2024,32(8):243-249.

地提高程序错误定位的效果。但是，在处理复杂度高的网络结构时，实验效果会低于预期。

为了解决这个问题，研究焦点逐渐转向深度学习模型在图结构分析中的应用。深度学习模型可以自动学习节点和边的特征表示，从而更好地捕捉到节点之间的复杂关系。特别是图注意力网络（GAT，graph attention networks），它能够同时考虑图的拓扑结构和节点属性信息，通过引入注意力机制，自动学习不同节点之间的权重关系，从而更准确地捕捉图数据中的关键信息。此外，深度学习模型通常具有较强的泛化能力，可以处理更复杂的图结构和数据分布。近几年，MLP-FL[20]、CNN-FL[21]、GNet4FL[22]和 DeepRL 4FL[23]都是具有代表性和有效性的。然而，深度学习模型同样面临一些问题。由于模型的复杂性和缺乏透明度，研究人员往往难以理解其工作原理和决策过程，这在一定程度上影响了模型的可信度和可靠性。鉴于此，受到强化学习思想的启发，开始关注可解释网络模型的发展。可解释模型通过提供清晰、易于理解的决策过程，揭示模型工作原理和决策依据[24]，从而增强模型的可信度和可靠性，还能够基于学习成果寻找对决策影响最大的子图。通过这种方法，可以更准确地定位到对模型分析 结果产生关键影响的部分。

本文旨在聚焦深度学习方法的最新成果，提出一种基于图解释模型的软件错误定位技术，利用可解释模型对图注意力模型的学习成果进行深入分析。期望通过分析结果，能够识别出最可能影响程序错误的子图，进而缩小软件错误定位的搜索范围，提升软件错误定位的准确度。

1 实验场景假设

本节设定了实验的场景，明确将实验数据聚焦于图结构之上，并规定实验方法应采用基于强化学习思想的图解释网络，以确保实验的针对性和有效性。

1.1 图结构

基于 Java 跟踪器，深入解析源代码，构建详尽的符号表，并通过控制流与数据流分析，全面记录程序运行时的执行路径。在控制流分析中，精准识别条件分支、循环结构以及方法调用，确保程序的每一个执行节点都被准确捕捉。同时，数据流分析构建出清晰的数据依赖图，细致记录变量的传递路径，为后续的测试工作提供坚实基础。完成测试后，所得数据依赖关系以二元组 DR 的形式保存在跟踪文件中：

$$DR = (\text{源代码行}, \text{目标代码行}) \tag{1}$$

其中：DR 由源代码行与目标代码行组成，直观展示两者之间的依赖关系。这样的记录方式不仅方便查询，而且为后续的测试覆盖率计算提供了精确的数据支持。

此外，跟踪文件被进一步划分为程序执行集，即每个测试用例 t 所对应的数据依赖关系集。通过这种方式，可以清晰地了解每个测试用例的执行情况，从而准确提取执行路径并计算测试覆盖率。$Trace_t$ 的定义明确指出了其由多个 DR 组成，即：

$$\{Trace_t = DR_1, DR_2, \cdots, DR_n, n \in N\} \tag{2}$$

其中：n 为自然数，代表 DR 的数量。

最终，通过构建程序执行网络，将程序语义特征和交互关系进行有机融合，得到一个清晰直观的图结构模型。该模型有助于深入理解程序的执行逻辑，并为后续的测试优化和性能提升提供有力支持。

1.2 解释模型

在深度学习的领域中，强化学习以其独特的试错机制，成为学习最优行为策略的关键方法。特别是面对序列决策问题，每一步的决策都直接影响着后续的状态和奖励，强化学习更是展现出了其独特的优势。

马尔可夫决策过程（MDP，Markov decision process）作为强化学习的基石，提供了一种数学化的描述方式，用于刻画那些具有马尔可夫性质的随机动态系统。在 MDP 中，状态转移虽然随机，但仅受当前状态和所执行动作的影响，与过去的状态和动作无关。这一特性使得问题得以被拆解成一系列独立的决策步骤，每个步骤都基于当前的状态和奖励信号来做出最佳决策。

因此，可以将强化学习应用于图注意力网络（GAT，graph attention networks）学习结果的解释，从而有效缩减解释空间[25]。图解释模型在软件错误定位中的解释过程，实质上是一个寻找最优子集的过程，这一过程能够精确地缩小寻找关键节点的范围。具体而言，网络中的每个节点可视为一个潜在的状态，选择某节点作为关键节点则视为一个动作，而选择后得到的网络结构或功能改善程度则作为奖励信号。通过不断的试错与策略调整，强化学习模型逐渐学会如何挑选出最优子集，即那些对网络结构和功能产生最大影响的关键节点。

在构建图解释模型时，强化学习与深度神经网络的结合显得尤为关键。深度神经网络首先对网络数据进行分析，提取出关键特征和规律。随后，引入强化学习的特性，用以对这些学习成果进行深入的评估和分析。通过不断的试错与策略调整，强化学习模型逐步识别出对网络结构和功能具有显著影响的关键节点，从而确定最重要的子集。这一过程显著缩小了错误节点的搜索范围，使得关键节点的定位更为高效和精准，为后续深入分析和研究提供了坚实基础。

2 实验框架

为了说明图解释模型在错误定位研究的可实施性，图 1 展示了基于图可解释模型的软件错误定位方法的架构。

具体的执行情况如下所述：

步骤 1：构建程序执行图。

程序的动态执行关系和测试用例覆盖率信息可以从执行跟踪中提取。通过构造程序执行图，融合了程序谱和交互关系的隐含信息，使基于复杂网络的错误定位可疑性分析成为可能。由于传统基于频谱的错误定位技术要求测试过程是动态的，所以程序执行图被构造为一个节点加权的有向动态网络 $PEN = (V, L, W)$，其中 $V = \{v_1, v_2, \cdots, v_n\}$ 是程序元素节点集。$L = \{v_i \rightarrow v_j \mid i, j \in |V$

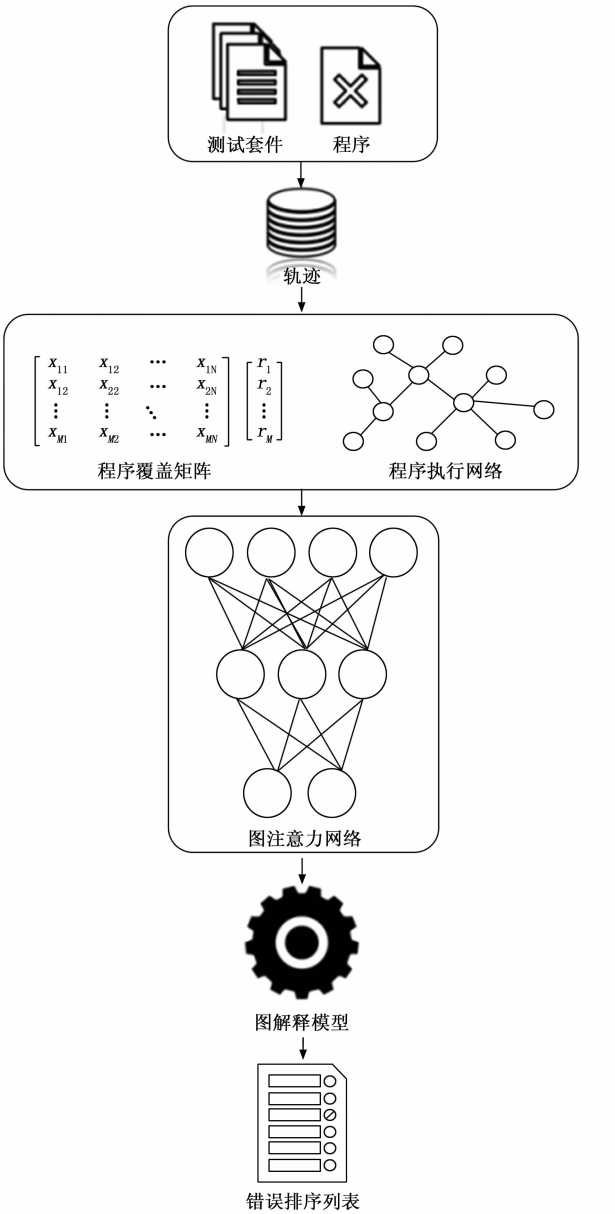

图 1　实验架构

$|\}$ 是程序执行轨迹的集合，其中 v_j 是执行关系的结束点，v_i 是执行关系的起始点。$v_i \rightarrow v_j$ 意味着 v_i 对 v_j 有影响。$\boldsymbol{W}=\{w_1, w_2, \cdots, w_n \mid n=|V|\}$ 是节点权重的集合，节点的权重就是程序元素的可疑度。

步骤 2：图表示学习模型训练。

利用图注意力网络对构建的图进行表示学习。GAT 通过注意力机制为每个节点学习一个表示向量，该向量能够捕获节点及其邻居的信息。在 GAT 中，对于每个节点，都会计算其与其他邻居节点之间的注意力分数，并据此确定邻居节点对当前节点的重要程度。然后，根据这些重要程度对邻居节点的特征进行聚合，得到当前节点的表示。

使用 GAT 模型对图进行训练，目标是让模型能够学习到能够区分“有错误”和“无错误”的代码元素的表示。在训练过程中，可以使用 SBFL 生成的可疑度得分作为监督信息，例如，将可疑度高的节点标记为“有错误”，可疑度低的节点标记为“无错误”。

步骤 3：解释模型对 GAT 的分析。

利用图解释模型对 GAT 模型的预测结果进行解释。它可以分析 GAT 模型内部的注意力权重，找出对预测结果影响最大的节点和边。这些影响最大的节点即为可能包含错误的代码元素。

步骤 4：程序的错误定位。

通过对解释模型分析得到的元素进行审查和分析，可以定位到具体的错误位置。研究人员可以通过手动审查代码、添加调试信息或使用其他调试工具来确认错误位置。

3　图可解释模型的构建

构建的图可解释模型主要依赖于图神经网络对图结构自主学习能力的分析能力，以及强化学习对图神经网络学习成果的决策过程解释。在这一章，主要介绍这两种技术的构建过程为后续的实验提供指导思想。

3.1　图注意力网络

图注意力网络本质上是引入注意力机制的图神经网络，所以核心在图注意力层。GAT 可以将代码片段纳入训练过程，捕获变量之间的依赖关系并生成相应的嵌入向量。

如图 2 所示，GAT 的训练架构由 6 个部分组成，它们是输入层、连接层、掩蔽的多头注意层、归一化层、n 个 Transformer 层和线性层。输入包括了覆盖信息和传播信息的条件向量与测试用例的执行结果。第二层是连接层，连接层的作用是为下一层提供表示。在连接层之后，有 n 个 Transformer 以产生上下文表示。每个 Transformer 层都包含一个架构相同的 Transformer，它在输入上应用多头自注意操作，然后是前馈层。掩蔽的多头注意力旨在过滤掉不相关的信号。注意力屏蔽函数通过将注意力得分加上一个无穷大的负值，使注意力权重在使用 softmax 函数后变为

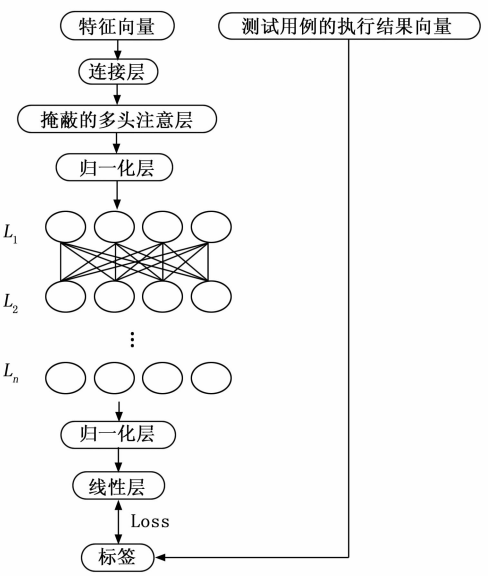

图 2　GAT 架构

零，从而避开查询所关注的关键字。

GAT 的输入是图数据中的节点特征向量集 $\boldsymbol{h}$，如公式所示：

$$\boldsymbol{h} = \{\vec{h}_1, \vec{h}_2, \cdots, \vec{h}_n\} \vec{h}_1 \in \boldsymbol{R}^F \tag{3}$$

其中：N 为节点个数，F 为节点特征的数量，$\boldsymbol{R}$ 为某一个节点的特征。矩阵 $\boldsymbol{h}$ 的大小为 $N \times F$，代表了图结构中所有节点的特征，向量 $\boldsymbol{R}$ 的大小为 $F \times 1$。输出为新的节点特征向量集 h'，与输入同理。为了得到相对应的输入和输出的转换，需要根据输入特征进行至少一次的线性变换。设置一个针对所有节点的权重矩阵 $\boldsymbol{W} \in \boldsymbol{R}^{F' \times F}$，$\boldsymbol{W}$ 可以反应输入节点特征 F 和输出节点特征 F' 的关系。对每个节点执行自注意力机制，注意力系数定义如公式所示：

$$e_{ij} = a(\boldsymbol{W}h_i, \boldsymbol{W}h_j) \tag{4}$$

其中：i 和 j 分别为节点 i 和节点 j，表示了节点 j 对节点 i 的重要性。值得注意的是 a 表示自定义函数，它可以将注意力分配到节点 i 的邻居节点 N_i 上，即 $j \in N_i$。为了使注意力系数更好地计算和比较，引入 sofmax 函数对所有与节点 i 相邻的节点 j 进行正则化，如公式所示：

$$a_{ij} = \mathrm{softmax}(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{(k \in N_i)} \exp(e_{ik})} \tag{5}$$

在注意力机制中，a 是一个单层的前馈神经网络，$\vec{\boldsymbol{a}} \in R^{2F'}$ 是神经网络中连接层之间的权重矩阵，在该前馈神经网络的输出层还加入了用于非线性化过程 LeakyReLU 函数。结合公式（4）和公式（5），得到完整的注意力机制：

$$a_{ij} = \frac{\exp(\mathrm{LeakyReLU}(\vec{a}^T[\boldsymbol{W}\vec{h}_i \parallel \boldsymbol{W}h_j \to]))}{\sum_{k \in N_i} \exp(\mathrm{LeakyReLU}(\vec{a}^T[\boldsymbol{W}\vec{h}_i \parallel \boldsymbol{W}h_j \to]))} \tag{6}$$

其中：$\parallel$ 是矩阵连接，$\boldsymbol{T}$ 是矩阵转置。通过上述运算得到了正则化后的不同节点之间的注意力系数，为了稳定注意力机制的学习过程，采用多头自注意力机制。具体而言就是使用多个独立自注意力机制执行公式，然后将其特征连接，可以用来预测每个节点的特征，如公式所示：

$$\vec{h}_i = \mathrm{sigmoid}\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{j \in N_i} a_{ij}^k \boldsymbol{W}^k \vec{h}_j\right) \tag{7}$$

其中：K 表示注意力机制的个数，k 表示 K 中的第 k 个，$\boldsymbol{W}^k$ 表示第 k 个注意力机制下输出特征的线性变化权重矩阵。

3.2 适用于图注意力网络的可解释模型

此方法通过将一个已经训练好的 GAT 和其预测结果作为输入，然后输出表示程序执行网络出错可能的特征信息，即子图以及该子图上更少的特征，表示其输出最大程度地影响了该 GAT 的预测结果。得到的子图可以最大化与 GAT 预测结果的互信息。

如图 3 所示，图解释模型的目的是寻找最小传播链。适用于图注意力网络的可解释模型是一种通过边掩码学习来搜索子图的技术，通过屏蔽程序执行网络中的边缘并学习最小传播链中的边缘屏蔽向量来获得最小传播链。通过切割原始传播链中的边缘，图解释模型试图找出本地化模型是否会获得相同的结果。

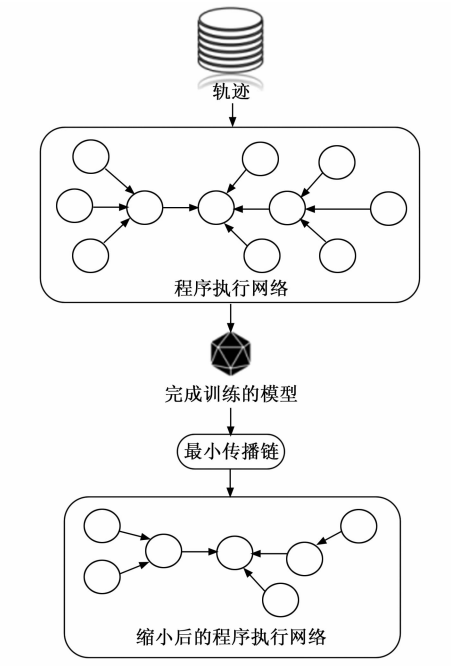

图 3 图解释模型架构

为了可以解释对预测结果最重要的特征，可解释模型会学习一个特征选择器 F，并认为 X_S^F 是子图 G_S 中节点特征的子集，$X_S^F = X_j^F \mid v_j \in G_S$，这样利用互信息 MI，可以将图注意力网络的可解释模型表示为：

$$\max_{G_s, F} MI(Y, (G_S, F)) = H(Y) - H(Y \mid G = G_S, X = X_S^F) \tag{8}$$

当经过 GAT 训练后的图解钩确定后 Φ，$H(Y)$ 就是确定的，这样公式就变成了最小化条件熵，如公式（9）所示：

$$H(Y \mid G = G_S, X = X_S^F) = -E_{Y \mid G_S, X_S^F}[\log P_\Phi(Y \mid G = G_S, X = X_S^F)] \tag{9}$$

由于图 G_S 选择的可能性非常大，通过学习子图的邻接矩阵并且再次增加限制，确保子图中不会出现原图没有的连边。这样转化成了优化子图的期望，如公式（10）所示：

$$\min_{G'_s} E_{G_s \sim G'_s} H(Y \mid G = G_S, X = X_S^F) \tag{10}$$

引入詹森不等式，凸函数条件下，函数的期望小于输入的期望，由于是求最小，那么只要得到输入期望的最小值即可，因此期望符号可以拿到子图 G_S，于是可以得到公式：

$$\min_{G'_s} H(Y \mid G = E[G_S], X = X_S^F) \tag{11}$$

最后，可以学习掩码集，并且切割结果子传播链用于解释预测失败结果。

实验的整个过程将基于图解释模型得到的子传播链进行分析。首先依据传统的 SBFL 技术，可以得到对每个节点的可疑度排序列表。接下来，通过模型的分析得到节点之间的传播关系，通过节点自身具备的可疑度值，结合传播关系影响程度，得到新的程序元素可疑程序的排序列表。

4 实验结果与分析

本文中涉及的图注意力网络和图可解释网络均使用

TensorFlow 框架，编程语言为 Java 和 Python，操作系统为 Ubuntu 18.04。实验的物理环境为处理器：AMD Ryzen 7 5800H with Radeon Graphics 3.20 GHz；显卡：GTX3060；硬盘：1 TB SSD；内存：16 GB。

本文方法中神经网络的超参数设置为：初始学习率（Learning Rate）为 0.001，辍学率（Dropout Rate）为 0.99，训练批次大小（Training Batch Size）为 2，评估批次大小（Evaluation Batch Size）为 32，采用自适应矩估计（Adam）优化算法加快模型的收敛。另外，本次实验尝试了一些其他参数，结果没有太大变化。在这项研究中，我们专注于所提出的方法的有效性，因此，并没有花太多的时间在参数调整上。

4.1　实验数据集

实验中，在 Defects 4J[26] 上评估了本文的方法，Defects4J（Defects for Java）是一个广泛用于软件错误研究的成熟实验数据集，主要针对 Java 程序。它提供了一组真实的开源 Java 项目，这些项目包含了人工插入的真实缺陷，以及修复了这些缺陷的版本。Defects4J 的目标是帮助研究人员和开发人员测试和改正错误检测和修复技术。Defects4J 在学术界和工业界都有广泛应用，它被认为是一个标准的缺陷研究基准。许多研究论文和项目都使用了 Defects4J 进行实验和评估。实验使用了真实的错误数据集 Defects 4J（版本 1.1.0），其中包含 6 个开源 Java 项目：JFreeChart、Google Closure Compiler、Apache Commons Lang、Apache Commons Math、Joda-Time 和 Mockito。所有基准程序的错误版本可分别从 Defects4j 资源库下载，对于每个错误，Defects4J 提供了一系列工具和脚本，用于下载、编译和运行项目，以及访问缺陷和修复版本。请注意，我们使用了 Defects4J 中选择的 348 个真实的错误，其中排除了跟踪系统问题的项目 Mockito 和其他 16 个错误，因为程序的错误没有定位在方法或构造函数内部，也有一些错误项目无法通过跟踪器得到执行轨迹。表 1 显示了 Defects4J 数据集的详细信息。

表 1　Defects4J 数据集

标识	名称	错误个数	代码行数/k	测试用例数
Chart	JFreechart	25	96	2 205
Closure	Closure compiler	133	90	7 927
Lang	Apache commons-lang	65	22	2 245
Math	Apache commons-math	98	81	3 546
Time	Joda-Time	27	28	4 130

4.2　评价指标

本文使用两种广泛使用的性能衡量指标 *EXAM* 和 *Top-N*，来评估所研究的错误定位技术的有效性。请注意，所有的指标都不考虑测试代码。

EXAM 是一个常用的错误定位效果评估指标，用于衡量错误定位技术的性能[27]。它是在发现错误元素之前必须检查元素的百分比。其计算公式如下：

$$EXAM = \frac{\sum_{i=1}^{n} rank(i)}{N} \tag{10}$$

其中：N 代表代码库中的代码行数，n 代表程序中错误的总数，$rank(i)$ 表示检测到第 i 个错误时需要检查的代码行数。*EXAM* 充分地考虑了研究人员在定位到程序错误所需要的代价。在最佳情况下，按照可疑度排序列表，研究人员只需要检查与错误总数相同数量的程序元素就可以定位到所有的错误。而在最坏情况下，研究人员需要查看所有列表中的可以元素。因此，*EXAM* 的值越小，表示错误定位的效果越好，因为它表示错误位置在排序中更接近前面。

Top-N 是另一种错误定位效果评估指标，它主要关注错误定位的前 N 个推荐代码位置是否包含了错误位置。在 *Top-N* 指标中，我们考虑将前 N 个被错误定位技术推荐的代码位置作为候选，然后通过比较其中是否包含真正的错误位置来评估性能。即在前 N 个推荐代码位置中，正确的错误位置的个数。*Top-N* 越高，表示错误定位技术在前 N 个推荐位置中包含了更多正确的错误位置，即性能越好。值得注意的是，开发人员通常仅检查排名靠前的元素[28]，因此，N 一般被设置为 1，3，5 这样的数字，*Top-N* 在错误定位实际运用中极为重要。

4.3　结果分析

在实验探究阶段，本文选取了 5 个传统 SBFL 可疑度计算公式（Jaccard、Tarantula、Ochiai、OP2 和 Dstar）和 4 个深度学习方法（MLP-FL、CNN-FL、GNet4FL 和 DeepRL 4FL）作为基线与 GRFL 进行了对比实验。实验的结果如表 2 所示。

表 2　10 种方法在 Defects4J 数据集上的结果

程序	技术	*Top*-1	*Top*-3	*Top*-5	*EXAM*
Chart	Jaccard	6	16	19	0.038
	Tarantula	7	17	20	0.037
	Ochiai	6	13	21	0.039
	OP2	5	14	16	0.061
	Dstar	5	16	19	0.042
	MLP-FL	8	13	17	0.039
	CNN-FL	10	16	19	0.036
	DeepFL	12	20	20	0.034
	GNet4FL	19	22	23	0.031
	GEFL	20	23	23	0.031
Closure	Jaccard	14	38	39	0.046
	Tarantula	12	26	36	0.031
	Ochiai	14	29	39	0.037
	OP2	17	33	42	0.053
	Dstar	14	38	39	0.044
	MLP-FL	19	30	48	0.036
	CNN-FL	32	43	50	0.033
	DeepFL	38	47	51	0.031
	GNet4FL	37	48	53	0.027
	GEFL	42	55	67	0.023

续表

程序	技术	*Top*-1	*Top*-3	*Top*-5	*EXAM*
Lang	Jaccard	22	45	56	0.043
	Tarantula	21	45	56	0.042
	Ochiai	22	44	56	0.043
	OP2	23	45	56	0.046
	Dstar	23	45	55	0.044
	MLP-FL	31	39	55	0.043
	CNN-FL	39	47	58	0.044
	DeepFL	46	54	59	0.036
	GNet4FL	42	55	57	0.039
	GEFL	45	57	59	0.035
Math	Jaccard	20	57	69	0.109
	Tarantula	20	57	60	0.102
	Ochiai	20	58	70	0.102
	OP2	19	52	61	0.110
	Dstar	20	56	69	0.102
	MLP-FL	51	61	72	0.076
	CNN-FL	59	68	76	0.071
	DeepFL	63	85	91	0.046
	GNet4FL	67	82	88	0.047
	GEFL	71	86	92	0.041
Time	Jaccard	5	9	18	0.033
	Tarantula	5	11	16	0.032
	Ochiai	6	11	18	0.031
	OP2	8	12	14	0.041
	Dstar	6	11	12	0.039
	MLP-FL	9	14	16	0.032
	CNN-FL	10	16	18	0.031
	DeepFL	13	17	17	0.031
	GNet4FL	14	18	20	0.023
	GEFL	14	21	24	0.022
总计	Jaccard	67	165	201	0.067
	Tarantula	65	156	188	0.061
	Ochiai	68	155	204	0.063
	OP2	72	156	189	0.078
	Dstar	68	166	194	0.068
	MLP-FL	118	157	208	0.057
	CNN-FL	150	190	221	0.054
	DeepFL	172	223	238	0.045
	GNet4FL	179	225	241	0.042
	GEFL	192	242	265	0.038

根据上述实验数据所示，基于深度学习的错误定位技术相比于传统的基于统计学方法的 SBFL 具有明显的优势，而本文提出的 GEFL 相比于其他的 4 种深度学习模型方法拥有更高的准确度。在衡量指标 *Top*-1，*Top*-3 和 *Top*-5 上均较其他方法分别提升了 7.26%，7.56%和 9.96%，*EXAM* 指数提升了 8.98%。也就是说，本文提出的实验方法能够有效提高自动化程序错误定位技术的准确性，能够辅助调试人员在更短的时间内找到程序中出错的位置。

5 结束语

本文针对计算机软件程序错误定位这一热点问题，以程序测试的执行过程作为角度进行研究，提出了一种基于图解释网络的程序错误定位方法。在程序覆盖信息的基础上，将测试用例对被测程序的影响关系转化为程序执行网络，利用这种特殊的网络结构有效地对代码进行表征，然后使用图注意力网络进行影响程度的传播，最后通过图解释模型对图注意力网络的训练结果进行分析，得到了最为重要的节点子集合，进一步缩小了程序定位的研究范围，结合解释后的重要程度和错误可疑度，可以得到新的程序可疑列表，辅助研究人员完成后续的错误定位工作。

在未来的研究中，将进一步探索可以将程序测试执行过程中的信息损失更少的特征表示方法，丰富深度学习模型的学习特征以及优化神经网络结构使其拥有更好的鲁棒性。

参考文献：

[1] HOWDEN W E. Theoretical and empirical studies of program testing [J]. IEEE Transactions on Software Engineering, 1978, 4 (4): 293 - 298.

[2] KIM J, AN G, FELDT R, et al. Ahead of time mutation based fault localisation using statistical inference [C] //2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE). IEEE, 2021: 253 - 263.

[3] WONG W E, LEE H J, GAO R, et al. Spectrum-based techniques for software fault localization [J]. Handbook of Software Fault Localization: Foundations and Advances, 2023: 201 - 270.

[4] PARSA S. Spectrum-based fault localization [M] //Software Testing Automation: Testability Evaluation, Refactoring, Test Data Generation and Fault Localization. Cham: Springer International Publishing, 2023: 317 - 332.

[5] LI Z, SHI B, WANG H, et al. Hmbfl: Higher-order mutation-based fault localization [C] //2021 8th International Conference on Dependable Systems and Their Applications (DSA). IEEE, 2021: 66 - 77.

[6] LI Z, WANG H, LIU Y. Hmer: a hybrid mutation execution reduction approach for mutation-based fault localization [J]. Journal of Systems and Software, 2020, 168: 110661.

[7] CAO H, WANG F, DENG M, et al. The improved dynamic slicing for spectrum-based fault localization [J]. Peer J Computer Science, 2022, 8: e1071.

[8] ISHIMOTO Y, KONDO M, UBAYASHI N, et al. PAFL: probabilistic automaton-based fault localization for recurrent neural networks [J]. Information and Software Technology, 2023, 155: 107 - 117.

[9] WIDYASARI R, PRANA G A A, HARYONO S A, et al. XAI4FL: enhancing spectrum-based fault localization with explainable artificial intelligence [C] //Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension, 2022: 499 - 510.

[10] PEARSON S, CAMPOS J, JUST R, et al. Evaluating and improving fault localization [C] //2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE). IEEE, 2017: 609 - 620.

[11] ABREU R, ZOETEWEIJ P, GEMUND A J C V. On the accuracy of spectrum-based fault localization [C] //Testing: Academic & Industrial Conference Practice & Research Techniques-mutation. IEEE Computer Society, 2007: 89 - 98.

[12] JONES J A, HARROLD M J, STASKO J. Visualization of test information to assist fault localization [C] //Proceedings of the 24th International Conference on Software Engineering. 2002: 467 - 477.

[13] ABREU R, ZOETEWEIJ P, VAN GEMUND A J C. An evaluation of similarity coefficients for software fault localization [C] //2006 12th Pacific Rim International Symposium on Dependable Computing (PRDC'06). IEEE, 2006: 39 - 46.

[14] NAISH L, LEE H J, RAMAMOHANARAO K. A model for spectra-based software diagnosis [J]. ACM Transactions on Software Engineering and Methodology (TOSEM), 2011, 20 (3): 1 - 32.

[15] WONG W E, DEBROY V, GAO R, et al. The DStar method for effective software fault localization [J]. IEEE Transactions on Reliability, 2013, 63 (1): 290 - 308.

[16] YOO S, XIE X, KUO F C, et al. No pot of gold at the end of program spectrum rainbow: Greatest risk evaluation formula does not exist [J]. RN, 2014, 14 (14): 14.

[17] VANCSICS B, HORVáTH F, SZATMáRI A, et al. Fault localization using function call frequencies [J]. Journal of Systems and Software, 2022, 193: 111429.

[18] LEI Y, MAO X, ZHANG M, et al. Toward understanding information models of fault localization: Elaborate is not always better [C] //2017 IEEE 41st Annual Computer Software and Applications [19] Conference (COMPSAC). IEEE, 2017, 1: 57 - 66.

[19] FAN X, WU K, ZHANG S, et al. Fault localization using trustrank algorithm [J]. Applied Sciences, 2023, 13 (22): 12344.

[20] SRINIVASAN S M, TRUONG-HUU T, GURUSAMY M. Machine learning-based link fault identification and localization in complex networks [J]. IEEE Internet of Things Journal, 2019, 6 (4): 6556 - 6566.

[21] ZHANG Z, LEI Y, MAO X, et al. CNN-FL: An effective approach for localizing faults using convolutional neural networks [C] //2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER). IEEE, 2019: 445 - 455.

[22] QIAN J, JU X, CHEN X. GNet4FL: effective fault localization via graph convolutional neural network [J]. Automated Software Engineering, 2023, 30 (2): 16.

[23] LI Y, WANG S, NGUYEN T N. Fault localization with code coverage representation learning [C] // 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), IEEE, 2021: 661 - 673.

[24] MOLNAR C. Interpretable machine learning [M]. Lulu. com, 2020.

[25] YING Z, BOURGEOIS D, YOU J, et al. Gnnexplainer: Generating explanations for graph neural networks [J]. Advances in Neural Information Processing Systems, 2019: 32.

[26] JUST R, JALALI D, ERNST M D. Defects4J: A database of existing faults to enable controlled testing studies for Java programs [C] //Proceedings of the 2014 International Symposium on Software Testing and Analysis, 2014: 437 - 440.

[27] PEARSON S, CAMPOS J, JUST R, et al. Evaluating & improving fault localization techniques [J]. University of Washington Department of Computer Science and Engineering, Seattle, WA, USA, Tech. Rep. UW-CSE-16-08-03, 2016: 27.

[28] KOCHHAR P S, XIA X, LO D, et al. Practitioners' expectations on automated fault localization [C] //Proceedings of the 25th International Symposium on Software Testing and Analysis, 2016: 165 - 176.

(上接第 242 页)

[22] MA C, YANG X, ZHANG C, et al. Long-term correlation tracking [C] // Computer Vision and Pattern Recognition. IEEE, 2015: 5388 - 5396.

[23] DANELLJAN M, HAGER G, KHAN F S, et al. Learning spatially regularized correlation filters for visual tracking [C] // IEEE International Conference on Computer Vision. IEEE, 2016: 4310 - 4318.

[24] BERTINETTO L, VALMADRE J, GOLODETZ S, et al. Staple: complementary learners for real-time tracking [C] // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016: 1401 - 1409.

[25] KIANI G H, FAGG A, LUCEY S, et al. Learning background-aware correlation filters for visual tracking [C] //Proceedings of the IEEE International Conference on Computer Vision, 2017: 1135 - 1143.

[26] YANG W, ZHAO M, HUANG Y, et al. Adaptive online learning based robust visual tracking [J]. IEEE Access, 2018, 6: 14790 - 14798.

[27] ROSS D A, LIM J, LIN R S, et al. Incremental learning for robust visual tracking [J]. International Journal of Computer Vision, 2008, 77 (1 - 3): 125 - 141.

[28] KWAK S, NAM W, HAN B, et al. Learning occlusion with likelihoods for visual tracking [C] // International Conference on Computer Vision. IEEE Computer Society, 2011: 1551 - 1558.

[29] MESHGI K, MAEDA S I, OBA S, et al. An occlusion-aware particle filter tracker to handle complex and persistent occlusions [J]. Computer Vision & Image Understanding, 2016, 150 (C): 81 - 94.