计算机研究与发展
Journal of Computer Research and Development
DOI: 10.7544/issn1000-1239.202330337
61(9): 2275−2289, 2024

基于图嵌入的二进制代码模块划分方法

孙华启　康　绯　舒　辉　黄宇垚　卜文娟
（中国人民解放军战略支援部队信息工程大学　郑州　450001）
（mr_sunhuaqi@163.com）

Binary Code Modularization Method Based on Graph Embedding

Sun Huaqi, Kang Fei, Shu Hui, Huang Yuyao, and Bu Wenjuan
（*PLA Strategic Support Force Information Engineering University*, *Zhengzhou* 450001）

Abstract　Reverse analysis as a key technology plays a vital role in cyber security. It helps analysts gain insight into the behavior of software and vulnerabilities detection, in order to effectively prevent attacks. The growing software scale and complexity urge some research to break down software into modules for rapid analysis via structural and functional information using community discovery algorithms. However, these studies just regard software as a social network consisting of simple nodes and edges missing valuable attribute information. We notice that the contribution of different features to the modular structure of the program is different and varies from samples. Inspired by the innovative application of graph embedding technologies in program analysis, we propose a binary code modularization method called GEBCM. The method transforms an executable program into an attributed graph, and employs graph embedding clustering methods with attention mechanism and ranking mechanism to embed representations and cluster function nodes. The result clusters group binaries into independent parts with more complete functions, revealing the semantic information of complex program structures. Experimental results show that GEBCM outperforms other modularization tools by revealing the original modular layout with an average of 10.2% higher *F*1 score. Additionally, in the new task of malware decomposition, GEBCM also exhibits better accuracy.

Key words　clustering; graph embedding; program analysis; modularization; reverse engineering

摘　要　软件逆向分析作为网络空间安全领域研究的核心支撑技术，在软件漏洞分析、恶意代码行为分析等方面有着广泛应用. 二进制代码的模块划分是该领域研究中的关键问题，通过将复杂或者大体量软件合理划分为若干模块，对于帮助分析者快速、准确理解软件结构与功能，提高分析效率起着重要作用. 对此，常见方法是将代码中的函数及其调用关系看作复杂网络，通过社区发现算法来进行函数聚类，实现模块划分，该类方法通常只考虑节点之间的连接关系，忽略了节点的属性、节点之间的相似度等信息，且对噪声和异常值比较敏感. 为了解决这些问题，提出了一种基于图嵌入的二进制代码模块划分（graph embedding based binary code modularization, GEBCM）方法. 该方法首先将软件系统抽象为属性图，然后通过带有注意力机制和排名机制的图嵌入聚类方法对函数节点进行嵌入表示并聚类. 通过聚类将二进制文件分组为更具有完整功能的独立部分，揭示了复杂程序结构中分离的模块语义信息. 在 2 个数据集上进行的实验评估，验证了所提出的 GEBCM 方法的有效性. 评估结果表明，相比其他二进制模块化工作，GEBCM 的 *F*1 值平均提高 10.2 个百分点. 此外，在针对恶意样本的评估实验中，GEBCM 能有效地划分出恶意代码的模块，表现出优秀的可扩展性.

关键词　聚类；图嵌入；程序分析；模块划分；逆向工程

中图法分类号　TP311

收稿日期：2023-04-24；**修回日期**：2023-10-16

通信作者：康绯（kfminnie@163.com）

随着网络环境日趋复杂和软件功能需求日益增长, 软件数量呈现爆炸式的增长. 据统计, 截止 2023 年 3 月, Google Play 商店中的应用程序数量已经达到 2 670 000 个[1]. 随之产生的软件安全问题已经成为影响信息系统安全的重要因素. 尽管开发者在软件安全测试中已经投入大量精力, 但是部分软件在发行后仍被发现有重要漏洞. 截至 2023 年 3 月, CVE(Common Vulnerabilities & Exposures)组织已经公布了超过 196 658 个漏洞记录[2]. 另一方面, 网络环境的多样性和复杂性为恶意软件的攻击和传播提供了便利, 对网络系统和主机的安全构成了巨大的威胁. AV-TEST(Anti Virus Test)机构每天都会注册超过 450 000 个新的恶意程序和潜在的有害应用程序[3].

为了满足日益增多的复杂软件分析需求, 动态污点分析[4]、动态二进制插桩[5] 等自动化的程序分析技术被发明并成功运用. 然而, 这些方法存在的效率低下、冗余输出等问题限制了软件的快速自动化分析. 随着人工智能产业的发展, 越来越多的机器学习方法被尝试用于解决软件安全问题, 例如软件缺陷预测[6]、程序修复[7]、恶意代码检测[8] 等. 尽管现有的这些方法能满足用户的基本程序分析需求, 但分析人员更希望得到一个详尽的程序架构信息来辅助软件分析.

软件开发通常遵循高内聚和低耦合的设计原则[9], 高内聚是指软件系统中模块内联系紧密, 低耦合指模块间联系尽量少. 这种模块化的设计方法可以降低代码的复杂性、控制开发和维护成本、促进代码的升级复用, 并且为软件模块划分提供了可能. 软件模块划分作为一种辅助软件分析技术已经广泛运用于源代码分析[10]. 然而, 大部分商业软件都是以二进制可执行文件的形式发布的, 如果能在缺少源码和调试信息的情况下将可执行文件分解为模块, 将会极大地提高软件分析的效率. 本文中的模块指的是能够完成一定功能的函数的集合, 模块内的函数联系紧密, 模块间一般具有相对的独立性, 在源码级别上模块通常表现为单个源程序文件.

本文提出了一个基于图嵌入的二进制代码模块划分(graph embedding based binary code modularization, GEBCM)方法. 具体而言, 该方法首先将函数及其链接关系组成的复杂网络抽象为属性图[11]; 然后利用自注意力机制动态调整不同链接关系的强度; 最后, 运用目标导向的聚类方法对节点进行聚类以实现软件模块的划分.

本文的主要贡献包括 3 个方面:

1)提出了一种基于图嵌入的二进制代码模块划分方法, 将程序分解为基于功能的模块;

2)基于动静结合的软件分析技术, 提出了一个面向多维特征的二进制代码函数属性图构建方法;

3)提出了一个基于神经网络的方法来生成并优化二进制函数的嵌入.

本文研究中的核心代码已经开源, 地址为 https://github.com/vipersun/GEBCM.

1 背 景

对程序进行快速逆向分析的需求推动了程序模块化研究的蓬勃发展. 目前, 已经提出了许多技术和方法[12-16], 其中二进制代码的模块划分方法备受关注. BCD(binary code decomposition)[12] 和 ModX(modularization)[13] 方法是比较典型的二进制代码模块化方法. BCD 是第 1 个将可执行文件分解为模块的静态方法, 而 ModX 则是利用模块划分和语义匹配实现第三方库检测的方法. 本节首先描述了这 2 个方法的基本技术路线, 并指出它们存在的一些局限性; 然后总结了二进制代码模块化研究所面临的挑战以及其具有的优势.

BCD 方法是一种利用静态分析技术将可执行文件分解为模块的方法. 具体来说, BCD 方法以函数为节点, 通过代码位置、数据引用和函数调用 3 种关系构建分解图. 然后, 运用纽曼的广义社区检测算法[17] 将可执行文件划分为独立的组件. 尽管 BCD 在实验中表现出较高的精确率和召回率, 但仍存在 2 个不足: 一是 BCD 方法中用来构建分解图的 3 种边只体现了程序的结构信息, 忽略了模块中函数的语义信息. 二是 BCD 中 3 种边的权重系数需要通过交叉验证的方法确定, 并且无法适用于不同类型的软件.

ModX 方法首先提出了一种基于特定程序调整的模块质量度量方法, 该方法能够对程序中的代码结构、函数调用等特征进行精确刻画, 为后续的模块划分提供参考. 然后, 采用具有启发式偏差的 Fast Unfolding Louvain 算法[18] 将二进制文件分解为模块. 该算法的贪心策略通常会得到具有较高模块度的模块划分结果. 然而, 迄今为止, 对于如何准确衡量程序模块化质量, 仍然缺乏一个统一的标准[19]. 这意味着, 模块质量度量方法可能会受到不同度量标准的影响, 从而导致结果存在偏差. 因此, 需要更加细致、全面地评估以该度量算法为核心的模块划分方法的有效性, 以保证其在实践中具有可行性和可靠性.

现有工作的局限性促使研究人员寻找一种更加准确的二进制代码模块划分方法. 为了实现这个目标, 本文总结了二进制代码模块划分领域面临的 3 个挑战.

1)虽然程序在编译的过程丢失了很多结构信息和符号信息, 但在可执行文件中仍然存在大量有用的信息, 可以用来识别模块. 我们之前提出的 DeMal(decomposition of malware)[20] 方法定义了编译器边界、序贯地址、直接调用、间接调用、数据引用 5 种模块特征, 取得了一定的研究成果. 但是这些低级特征并不能完全反映出程序的高级语义信息, 例如模块的功能、质量等. 因此, 选择能真实反映模块结构化信息、提高模型精度和效率的特征是模块划分首先面临的第 1 个挑战.

2) 现有的大部分软件模块划分方法的局限性之一是使用预先确定的权重系数将多维特征拼接在一起. 事实上, 不同类型软件的多维特征之间的差异是动态变化的. 例如, 高并发、多线程的软件会尽量少使用全局数据来避免产生变量被多次赋值等不安全问题. 因而这类软件的数据引用的特征会相对较弱. 因此, 寻找一种能够自动适应不同类型软件的特征融合方法也是模块划分面临的挑战之一.

3) 将软件视为复杂网络进行软件分析已经被广泛运用并取得了较好的效果[21-22]. BCD 和 ModX 均采用传统的基于同构网络的社区发现算法来寻找最优社区. 然而, 由函数及其链接关系形成的网络属于异构信息网络, 网络中不同的链接具有不同的语义, 这些链接在聚类过程中可能具有不同的重要程度. 因此, 选择合适的能处理复杂异构信息网络的模块划分算法是面临的最大挑战.

二进制代码模块划分是软件逆向工程中的一项重要技术, 它将一个二进制代码文件划分为多个模块, 每个模块对应程序中的一个功能或代码段. 通过对这些模块进行分析, 可以更深入地了解程序的结构和功能, 从而更好地进行二进制代码分析. 相较于直接分析, 将二进制软件分解为模块后进行分析的方法具有 3 个优势:

1)能够清晰地划分程序的组成和结构, 使分析人员能够更好地掌握软件的内部机制. 通过分离不同的模块, 分析人员可以更深入地了解每个模块的功能和作用, 从而更好地理解整个软件的结构和工作原理.

2)有助于快速逆向定位, 缩小逆向分析的工作范围. 由于二进制软件通常非常庞大且复杂, 因此在逆向分析的过程中很容易迷失方向. 但是, 通过分解为模块, 分析人员可以更加精确地定位需要分析的部分, 从而可以更加高效地进行逆向分析.

3)可以帮助分析人员识别二进制软件中的潜在漏洞和安全问题. 通过分析每个模块的代码, 分析人员可以发现可能存在的漏洞或安全隐患, 从而及时采取措施进行修复或加固, 提高软件的安全性. 此外, 分解为模块后还可以进行模块间接口的分析, 发现潜在的数据安全和传输问题.

2 相关工作

本节介绍了二进制程序分析、软件模块聚类和基于深度学习的图嵌入领域的相关工作.

2.1 二进制程序分析

二进制程序分析是指对计算机程序的二进制代码进行分析和研究, 以了解程序的结构、功能、漏洞和安全性等方面的信息[23]. 分析方式一般包括静态分析、动态分析和动静结合的混合分析方法.

常用的静态分析软件包括 IDA Pro[24], Radare2[25], Ghidra[26] 等. 其基本原理是将二进制代码反汇编为包含丰富语义的指令序列, 并进一步分析程序样本的令牌、字串、树形、图形等属性信息. 动态分析是指在受控环境中实际执行待检测程序, 并依据待检测程序表现出的动态行为特征进行分析. Henderson 等人[27] 基于 QEMU 提出了一个多目标、全系统的动态二进制分析框架 DECAF, 极大地提高了分析效率和可用性. 为了兼顾动态、静态 2 类技术的优点, 一些混合分析工具被开发出来.BitBlaze 平台[28] 由混合分析组件构成, 以代码语义分析为基础, 采用动静结合形式化分析技术实现可执行程序安全性分析. 跨平台二进制分析平台 Angr[29] 结合动态与静态分析技术实现自动化的二进制文件分析, 提供了控制流、数据流分析以及符号执行等功能. 这些分析方法和工具大多数都是对完整的二进制程序进行直接分析, 忽略了软件功能结构中包含的重要语义信息, 因此在面对体量庞大的二进制代码时速度和准确性都会严重下降.

2.2 软件模块聚类

软件模块聚类是一种基于模块间关系的软件划分方法, 旨在得到高内聚、低耦合的功能组件, 以此来提高软件结构的清晰度、易理解性、调试性, 同时降低软件分析的复杂性.

Xia 等人[14] 提出了一种针对软件模块的聚类方

法, 该方法采用层次聚类组合的方式, 利用主成分分析将多个聚类算法的结果进行组合, 以尽可能保留每个算法的基本信息, 从而达到最佳的聚类效果. Papachristou[15] 在模块级别上使用 Doc2Vec 算法和调用关系构建一个网络图, 并利用 Louvain 算法挖掘其中的社区结构. Pan 等人[16] 针对传统软件模块聚类中由于使用无向无权图而导致的软件拓扑结构信息丢失的情况, 提出了基于加权有向图来表示软件拓扑结构的一种广义 K 核分解方法, 并根据模块的广义核心对类进行排序来确定软件中的关键类. 文献 [14−16] 的方法大多数都是在源码级别上进行聚类, 但是实际情况中, 获取软件的源码是一项非常困难的任务.

2.3 基于深度学习的图嵌入

基于深度学习的图嵌入是指利用深度学习技术将图数据映射为低维稠密向量的过程, 并在此过程中保留图的拓扑结构、顶点间关系以及其他相关信息. 根据输出粒度的不同, 图嵌入可以分为节点嵌入、边嵌入、混合嵌入和整图嵌入[30].

SDNE(structural deep network embedding)[31] 是第 1 个将深度学习应用于网络表示学习的方法. 该方法利用多层非线性函数来捕捉高度非线性的网络结构, 并利用一阶和二阶接近度分别表示局部和全局网络结构. 通过在半监督模型中联合优化, 得到最终的网络嵌入表示.DNGR(deep neural networks for learning graph representations)[32] 是一种采用随机冲浪模型直接捕获图结构信息并引入堆叠去噪自动编码器来得到嵌入表示的方法. 文献 [31−32] 的方法都是以节点的全局信息作为输入, 不适用于大规模稀疏图的表示学习. 为了解决这个问题, 图卷积神经网络(graph convolutional network, GCN)[33] 通过在图上定义卷积算子, 利用边的信息对节点信息进行聚合, 从而生成包含邻居节点信息、结构信息和当前节点信息的新的节点表示. 然而, GCN 也存在一些局限性, 例如无法处理有向图, 也难以实现给不同邻居节点分配不同学习权重的功能. 图注意力网络(graph attention network, GAT)[34] 通过引入注意力机制将节点间的相关性融入模型, 将邻居节点与当前节点的相关性进行注意力计算并作为邻居节点的权重加入节点的汇聚表示. GAT 只考虑了节点的连接, 忽略了边的强度、类型等特征, 因此不适用于处理复杂的异构信息网络.

3 基于图嵌入的二进制代码模块划分方法

鉴于现有工作的局限性, 本文提出了 GEBCM 方法, 目的是为了更加准确快速地将二进制可执行文件分解为模块. 本节首先描述了 GEBCM 的整体架构, 然后阐述了该方法的具体实现以及如何克服第 1 节中提到的 3 个挑战.

3.1 GEBCM 的整体架构

图 1 展示了 GEBCM 的整体架构. 给定一个二进制可执行文件, 首先对其进行混合分析, 以获取其中包含的函数及其属性信息, 其中属性信息包括函数地址、调用关系、数据引用和 API 调用等信息. 接下来, 以函数作为图的节点, 将提取的函数属性信息作为节点属性信息, 根据函数间的调用关系构建有向属性图(见 3.2 节). 由于不同的属性信息往往具有不

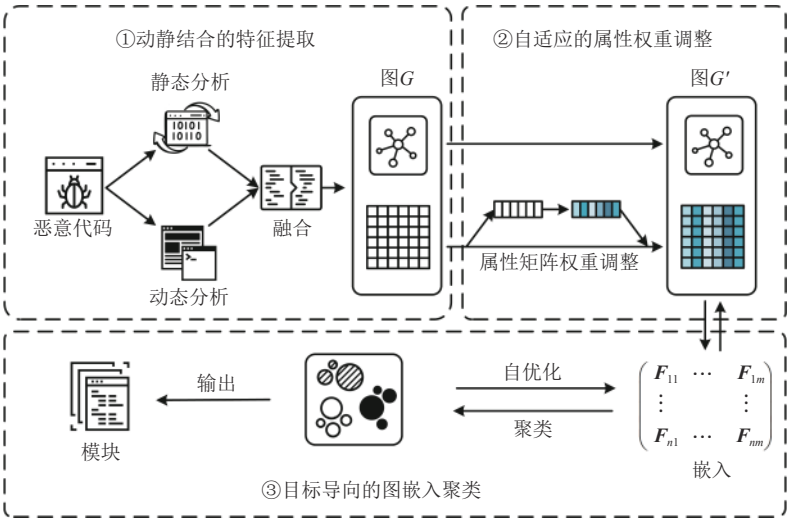


Fig. 1 System architecture

图 1 系统架构

同的重要性，因此采用自注意力机制来自动学习属性权重（见 3.3 节）. 在优化后的属性图中，利用图自编码器对节点进行表示学习，得到每个节点的嵌入表示. 同时，通过基于重构损失的自优化模块对节点嵌入表示进行优化. 最后，运用优化后的 DAEGC（deep attentional embedded graph clustering）算法[35]对节点进行聚类，实现模块的划分（见 3.3 节）.

定义 1. 一个可执行程序可以表示为属性图 $G=(V,E,\boldsymbol{F})$，其中点集 $V=\{v_1,v_2,\cdots,v_n\},|V|=n$ 表示函数集合，边集 $E\subseteq V\times V$ 表示函数间调用关系集合，$\boldsymbol{F}\in\mathbb{R}^{n\times m}$ 表示节点属性矩阵，m 表示节点属性维度.

3.2 动静结合的特征提取

函数作为模块的基本组成单元，其特征信息的提取对于实现精准的模块划分至关重要. 为了充分从二进制文件中提取有用信息并克服挑战 1，GEBCM 方法首先采用动静结合的混合分析方法对软件进行充分解析以消除代码混淆、加壳、加密等影响；然后提取函数的地址、调用、数据、API 调用等 4 种与程序结构紧密相关的信息，并对提取的特征信息进行归一化处理；最后以函数为节点、函数间调用关系为边构建程序属性图.

在源码编译生成二进制可执行文件的过程中，很多有利于理解代码意图的信息被丢失. 因此，在特征提取的过程中，首先需要通过反汇编将二进制机器码转换为等价的汇编代码.GEBCM 方法采用 IDA Pro 作为基本的静态分析工具，用于提取二进制可执行文件中的函数和数据. 为了尽可能恢复程序的真实语义，该方法还通过二进制插桩平台 Pin[36]记录样本的执行轨迹，以识别间接跳转并抵消抗静态分析手段带来的不良影响. 为了尽可能从可执行程序中提取反映模块化结构的信息，本文从软件设计的角度出发对二进制文件中存在的能够表达原始模块化结构的信息进行分析，并构建相应的属性矩阵.

1）函数地址. 在将源码转换为可执行文件的过程中，通常需要进行链接操作，以将单个源程序文件、静态库文件和动态库文件等生成的中间文件链接到一起，形成可执行文件. 虽然不同的链接器对多个中间文件的组合顺序有不同的规定，但中间文件内的函数位置基本保持原来的连续性[37]. 因此，在函数特征的选择时，需要优先考虑体现模块结构信息的函数地址特征. 为了构建函数地址属性矩阵 $\boldsymbol{F}_{\text{addr}}$，首先利用 IDA Pro 识别出每个函数的起始地址，并根据地址大小为每个函数分配一个位置索引；然后构建一个大小等于索引数量的方阵，并将方阵每个位置的值设为以该位置的行号与列号之差为输入的标准正态分布的概率密度函数的值.

2）函数调用. 在软件开发过程中，为了提高程序的模块化特性，通常将执行同一任务或相近任务的函数放到同一模块中. 因此，相互调用次数较多且函数地址相近的一组函数通常属于同一个模块. 因此，函数调用可以作为一个重要特征加入属性矩阵. 为了充分表征函数之间的调用关系和函数之间的距离，本文以函数为节点构建了一个邻接矩阵 $\boldsymbol{F}_{\text{call}}$. 在该邻接矩阵中，每个节点代表一个函数，每个元素表示 2 个节点之间的关系. 具体而言，如果函数 i 调用了函数 j，则将邻接矩阵中第 i 行第 j 列的元素设为 $1/d$，其中 d 表示函数 i 和函数 j 之间的索引距离.

3）数据引用. 根据高通信内聚和低公共耦合的设计理念，设计良好的软件模块之间应尽可能避免使用公用数据. 因此，访问相同全局数据区的函数很可能属于同一模块. 为了构建数据引用属性矩阵 $\boldsymbol{F}_{\text{data}}$，首先利用混合预处理技术识别二进制文件中的全局变量、静态变量和常量等数据，并为它们分配一个索引. 在每个函数的数据引用属性向量中，如果该函数使用了某个数据，则将其对应的索引位置置为 1.

4）API 调用. 在软件设计过程中，通常会根据功能划分模块，模块内的函数为完成这个功能会使用语义相近的 API 序列. 因此，API 调用可以作为函数的语义特征加入属性矩阵. 本文将 Windows 操作系统上常用的 294 个动态链接库的 API 分为 8 个大类 41 个小类，如表 1 所示. 为了构建 API 调用特征的属性矩阵 $\boldsymbol{F}_{\text{API}}$，我们将每个函数使用的 API 划分到相应的分类中，然后将函数对应的属性位置置为 1，使得 API 调用的属性矩阵的列维度为 41.

在获得 4 种特征信息的属性矩阵后，将它们横向拼接形成恶意代码函数的属性矩阵 $\boldsymbol{F}=(\boldsymbol{F}_{\text{addr}},\boldsymbol{F}_{\text{call}},\boldsymbol{F}_{\text{data}},\boldsymbol{F}_{\text{API}})\in\mathbb{R}^{n\times m}$，其中 n 为函数个数，m 表示属性维度. 然后，以恶意代码中所有函数 V 为节点、函数间的调用关系 E 为有向边、$\boldsymbol{F}$ 为函数节点属性信息来构建属性图 $G=(V,E,\boldsymbol{F})$.

3.3 自适应的特征权重调整

第 1 节挑战 2 中描述了一个事实，即不同的特征对输出结果的重要性存在差异，而这种差异在不同类型的软件上也不尽相同. 为了充分利用这些特征之间的互补性最大化模型性能，在每个特征中加入了一个自适应动态调整的权重系数，从而形成了最终的二进制代码属性矩阵.

本文采用了计算机视觉领域的神经网络模型 SE

Table 1 Classification of API Function

表 1 API 函数分类

类型	具体分类
基本系统服务	数据访问和存储、动态链接库、索引服务、进程间通信、进程和线程、远程桌面服务、同步、服务、Windows 系统信息、Windows 消息
系统管理	应用程序安装和服务、压缩、设备、文档和打印、增量压缩、内核事务管理器、内存管理、NT Native、电源管理系、统管理、Windows 驱动程序工具包、Windows 环境开发
系统安全	安全与认证
网络	互联网、网络、网络安全服务、无线网络
用户接口	Windows 数据类型、虚拟化、Windows 应用程序 UI 开发
运行时库	Microsoft.NET 库、Netscape Portable 运行时库、Visual C++运行时库、其他资源库
图形和多媒体	音频和视频、图形和游戏
其他	组件对象模型、诊断、Office 开发、Web 开发、未记录

block[38] 来实现自适应特征权重调整.SE block 的网络结构如图 2 所示. 该模型首先对特征矩阵进行全局平均池化，将通道级的特征压缩到 1 个维度，即挤压(squeeze)操作. 接着，通过一个 2 层的全连接神经网络对压缩后的特征进行激励(excitation)操作，自动学习各个通道间的关系，得到不同通道的权重. 最后，将权重乘以原来的特征矩阵，得到最终特征. SE block 本质上是在通道维度上实现了注意力机制，使模型更加关注信息量最大的通道特征，并抑制不重要的通道特征以提高模型的性能.SE block 的权重向量计算方法为

$$\boldsymbol{s}=Sigmoid(\boldsymbol{W}_2ReLU(\boldsymbol{W}_1\boldsymbol{z})), \tag{1}$$

其中 $\boldsymbol{z}\in\mathbb{R}^c$ 是 $\boldsymbol{F}$ 经过全局平均池化后得到的向量，$\boldsymbol{F}$ 是软件的函数属性矩阵.$\boldsymbol{W}_1\in\mathbb{R}^{\frac{m}{r}\times m}$，$\boldsymbol{W}_2\in\mathbb{R}^{m\times\frac{m}{r}}$ 分别是 2 个全连接层的权值矩阵，r 是代表缩减率的超参数，m 表示节点属性维度. 在获得权重向量 $\boldsymbol{s}$ 后，将每个维度的权重与相应的原始矩阵向量相乘，最终生成带有权重的属性矩阵.

GEBCM 方法将函数属性矩阵作为 SE block 的输入，并将每一个列向量视为一个通道，通过挤压和激励计算每个通道的权重值，最终通过权重相乘(scale)得到加权的属性矩阵.

3.4 目标导向的图嵌入聚类

二进制代码模块划分是一种图划分问题，旨在挖掘图 G 中的模块，该问题已被证明为 NP 难问题[39]. 图嵌入方法可以将图投影到低维向量空间中，以更低的复杂度处理图数据，同时保留图的内在结构属性. 为了克服挑战 3，GEBCM 使用图嵌入方法对二进制代码属性图中的节点进行表示学习，并通过在低维空间上对节点进行聚类来实现模块的划分. 本节

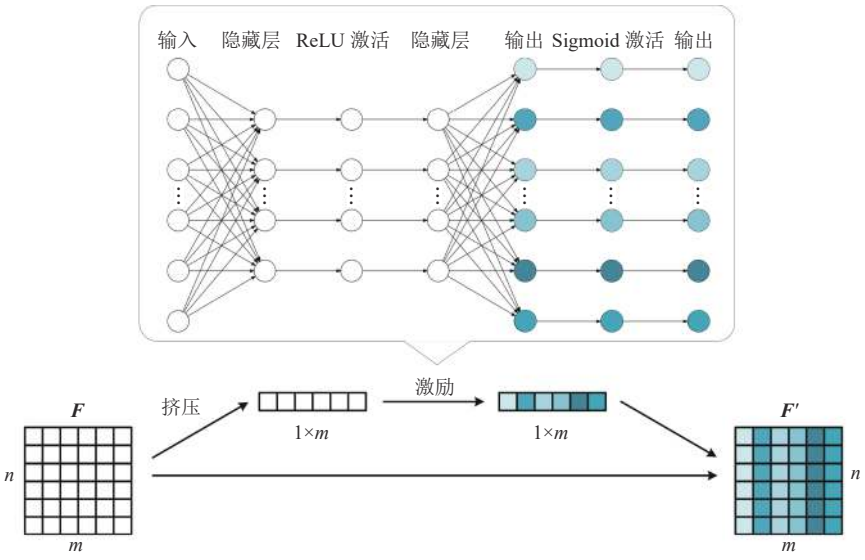


Fig. 2 SE block structure

图 2 SE block 结构

使用了一个两阶段的目标导向的深度图嵌入聚类算法，其基本流程如图 3 所示. 在第 1 阶段（图 3 中①②③④）中，首先针对二进制代码属性图设计了一种图自编码器，对图 3 中的结构和节点信息进行图表示学习，并通过自训练模块对节点嵌入表示进行优化. 在第 2 阶段（图 3 中⑤⑥⑦②），运用目标导向的聚类算法，在第 1 阶段生成的节点嵌入表示的基础上对函数进行聚类，从而实现模块的划分.

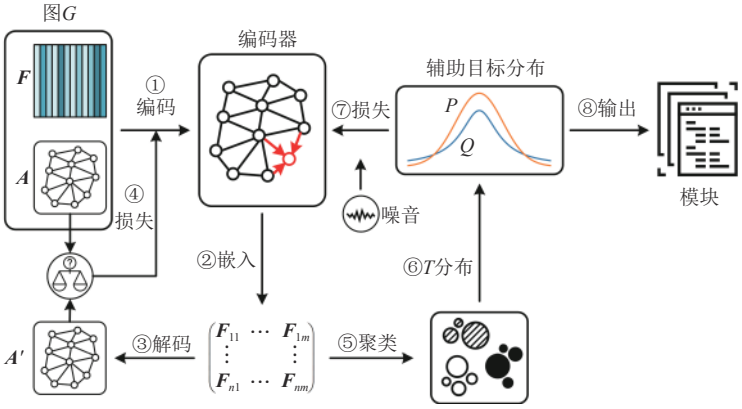


Fig. 3 Goal-oriented graph embedding clustering algorithm

图 3 目标导向的图嵌入聚类算法

3.4.1 加入排名的图自编码器

为了充分捕获网络结构和节点属性信息，本文基于 GAT 设计了一个面向二进制代码属性图的图自编码器. 其核心思想是通过关注邻居节点来学习每个节点的隐藏表示，以便在将节点信息映射为低维向量时尽可能多地保留图的拓扑结构.

为了衡量不同邻居节点在节点表示中的重要性，GAT 计算每个邻居节点与当前节点的注意力系数并将其作为权重系数，加入节点表示中：

$$\boldsymbol{x}_i' = ReLU\left(\sum_{j\in N_i}\alpha_{ij}\boldsymbol{W}\boldsymbol{x}_j\right), \tag{2}$$

$$\alpha_{ij} = \frac{\exp(LeakyReLU(\boldsymbol{a}^{\mathrm{T}}[\boldsymbol{W}\boldsymbol{x}_i \,\|\, \boldsymbol{W}\boldsymbol{x}_j]))}{\sum_{k\in N_i}\exp(LeakyReLU(\boldsymbol{a}^{\mathrm{T}}[\boldsymbol{W}\boldsymbol{x}_i \,\|\, \boldsymbol{W}\boldsymbol{x}_j]))}, \tag{3}$$

其中 $\boldsymbol{x}_i'$ 表示节点 v_i 的输出表示，$\boldsymbol{x}_j$ 是节点 v_j 的原始表示，N_i 表示节点 v_i 的所有邻居节点. $\boldsymbol{W}\in\mathbb{R}^{m'\times m}$ 为共享权重矩阵，m 和 m' 分别表示节点的初始特征维度和更新后的特征维度. α_{ij} 表示节点 v_j 对节点 v_i 的注意力系数. $\boldsymbol{a}\in\mathbb{R}^{2m'}$ 是一个权重向量，其作用是将拼接后的高维特征映射到一个实数上.

定义 2. 函数重要度.是指函数在程序中的重要程度，通常表示为函数调用其他函数次数与被其他函数调用次数的比值. 本文仅关注用户自定义的函数.

在 GAT 中，所有的节点被视为具有相同类型的实体. 然而，在二进制代码属性图中，函数节点由于语义的不同具有不同的重要度. 例如，POSGrabber[40] 是一款安装在 POS 机中用于窃取信用卡信息的恶意软件，其函数调用情况如图 4 所示，POSGrabber 中的 Common 模块主要定义了数据加解密、内存管理等通用工具类函数，因此该模块中的函数被调用的次数远高于其他模块，函数的重要度也会低于其他函数. 由于这类函数与其他模块函数的联系较为紧密，因此会对聚类的准确性产生较大的影响. 为了解决这个问题，GEBCM 将函数重要度作为函数的排名加入到节点的嵌入表示中. 更新后的节点表示为

$$\boldsymbol{x}_i' = ReLU\left(\sum_{j\in N_i} I_j\alpha_{ij}\boldsymbol{W}\boldsymbol{x}_j\right), \tag{4}$$

其中 I_j 为函数 v_j 的重要度.

为了提高图自编码器的有效性，本文使用反向传播算法对节点的嵌入表示进行优化，以最小化图

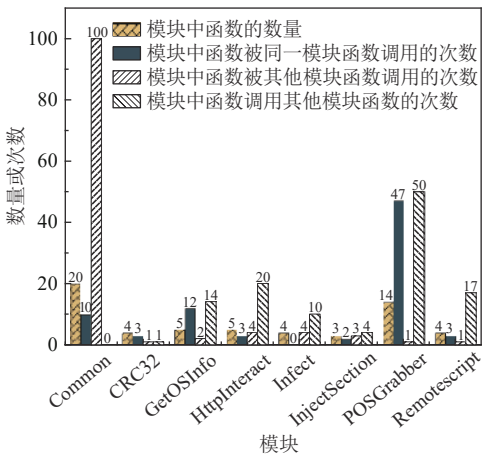


Fig. 4 Statistics of POSGrabber function calls

图 4 POSGrabber 函数调用的统计信息

自编码器的重构损失. 损失函数为

$$L_{\mathrm{ReSt}}=\sum_{i=1}^{n} loss\left(A_{ij},\hat{A}_{ij}\right), \tag{5}$$

其中A_{ij}和$\hat{A}_{ij}$分别是图的原始结构矩阵和重构矩阵中第i行第j列的值. 重构矩阵的元素是通过一个简单的内积解码器计算得到的:

$$\hat{A}_{ij}=Sigmoid\left(\boldsymbol{x}_i'^{\mathrm{T}}\boldsymbol{x}_j'\right). \tag{6}$$

3.4.2 目标导向的聚类算法

聚类是一种无监督学习任务, 由于缺乏真实标签的指导, 所以通常需要定义一个目标函数或损失函数来指导聚类过程. 本文采用了一种基于辅助目标分布的方法[41], 通过迭代优化聚类目标来进行聚类. 该方法首先在基于图自编码器生成的节点表示的基础上, 通过初始聚类算法对节点进行聚类, 得到一个初始软聚类标签和聚类中心(初始Q分布). 然后, 通过定义的优化目标分布对聚类进行指导以更新节点嵌入表示. 为确保目标分布的有效性, 每更新5次节点表示后对目标分布进行重新计算. 最终, 通过迭代缩小损失, 直到达到最大迭代次数.

为了获得较为准确的初始聚类中心, 该算法在节点表示的基础上运行了多次k-means算法, 每次使用不同的随机聚类中心. 然后, 选择其中表现最优的结果作为初始聚类中心. 在得到初始聚类中心后, 使用T分布[42]来度量节点属于聚类中心的概率, 即节点的软聚类标签:

$$q_{iu}=\frac{\left(1+\|\boldsymbol{x}_i-\boldsymbol{\mu}_u\|^2/\alpha\right)^{-\frac{\alpha+1}{2}}}{\sum_k\left(1+\|\boldsymbol{x}_i-\boldsymbol{\mu}_k\|^2/\alpha\right)^{-\frac{\alpha+1}{2}}}, \tag{7}$$

其中$\boldsymbol{x}_i$为节点v_i的嵌入表示, $\boldsymbol{\mu}_u$为簇u的聚类中心, α为T分布的自由度, q_{iu}表示节点v_i属于簇u的概率. 由于高概率的软聚类标签具有较高置信度, 算法将实际分布提高到二次方并进行归一化作为目标分布:

$$p_{iu}=\frac{q_{iu}^2\Big/\sum_i q_{iu}}{\sum_k\left(q_{ik}^2\Big/\sum_i q_{ik}\right)}, \tag{8}$$

其中p_{iu}表示节点v_i属于簇u的目标概率.

为了提高算法的鲁棒性和泛化能力, 同时避免算法陷入局部最优解, 本文在计算损失的过程中引入了高斯噪声作为随机扰动, 使得算法对于略微不同的输入也能够产生相对稳定的输出. 聚类损失的计算方法为

$$L_{\mathrm{Clust}}=KL(P\|Q)=\sum_i\sum_u p_{iu}\lg\frac{p_{iu}}{(q_{iu}+\varepsilon)}, \tag{9}$$

其中L_{Clust}表示P和Q的散度损失, P表示目标分布, ε表示服从高斯分布的随机扰动项. 为了共同优化嵌入过程和聚类过程, 定义了一个总损失函数L:

$$L=L_{\mathrm{ReSt}}+\gamma L_{\mathrm{Clust}}, \tag{10}$$

γ作为超参数, 可以通过预先实验获得. 最后通过优化后的Q分布获得每个节点的聚类标签.

算法整体流程如算法1所述.

算法 1. 基于图嵌入的二进制代码模块划分算法.

输入: 二进制可执行文件;

输出: 模块.

/*第1阶段*/

① 对二进制可执行文件执行静态分析以构建属性图G;

② 执行动态分析以补充G中的信息;

/*第2阶段*/

③ 将图G中的属性矩阵输入SE block, 将属性矩阵在列方向上压缩到1个维度;

④ 通过2层全连接网络和多个激活函数计算每个通道之间的相关性, 进而生成权重向量;

⑤ 将权重向量每一维与对应的原始属性矩阵向量相乘, 最终得到加权的属性矩阵;

/*第3阶段*/

⑥ 对于属性图中的每个节点, 首先将其属性向量作为其原始表示;

⑦ 计算每个节点的邻居节点对当前节点的重要性系数, 并将其与邻居节点表示相乘加入当前节点的表示, 最终得到属性图G中所有节点的初始嵌入;

⑧ Repeat

⑨ 计算重构损失;

⑩ 反向传播损失以更新节点嵌入表示;

⑪ Until($epoch \geqslant max_embedding_epoch$)

⑫ Repeat

⑬ 对节点进行聚类获得(初始)聚类中心;

⑭ Repeat

⑮ 通过T分布计算Q分布(实际分布);

⑯ if $epoch\%update_interval == 0$

⑰ 生成目标分布;

⑱ end if

⑲ 计算总损失;

⑳ 反向传播损失以更新节点嵌入表示;

㉑ Until($epoch \geqslant max_distribution_epoch$)

㉒ Until($epoch \geqslant max_iteration_epoch$)

㉓ 优化后的 Q 分布作为输出.

4 实验与结果

本节对本文方法进行评估，本文的实验机器是 Intel® Core ™ i7-8700H CPU @ 3.60 GHz 和 32 GB 内存的 Windows 10 Pro 系统.

4.1 超参数设置

经过多次预先评估实验，对算法 1 中的超参数设置为：图自编码器的迭代次数 *max_embedding_epoch*=30；目标分布的迭代周期 *update_interval*=5；每重新计算 1 次聚类中心，迭代 30 次节点表示，即 *max_distribution_epoch*=30；聚类中心的迭代次数为 10，即 *max_iteration_epoch*=10.在计算总损失过程中，鉴于第 1 阶段已经获得了相当准确的节点嵌入表示，因此将参数 γ 设定为 10，使模型更加聚焦于聚类过程的损失. 算法 1 中 k-means 算法的 k 值设定为函数的个数与 10 的比值.

4.2 数据集和指标

为了验证本文方法的准确性和扩展性，本文从开源软件仓库或官方网站中筛选符合 2 个条件的项目源码：1）具有一定模块化特性，即至少具备 3 个以上的源代码文件；2）拥有完整的工程结构，该结构包含了编译所需的所有文件. 接下来，通过编译脚本对收集的项目进行自动编译，并在链接过程中输出包含调试符号信息的映射文件，以获得函数和模块的对应关系. 最终构建 3 个数据集，如表 2 所示.

Table 2 Parameters of Three Datasets

表 2 3 个数据集参数

数据集	样本数量	平均函数个数	平均模块个数	样本平均大小/KB
S1	85	865	18	985
S2	1 232	947	15	910
M1	136	264	14	331

数据集 S1 由从软件官方网址获取的开源软件组成. 这类软件通常由组织者或志愿者共同开发和维护，符合软件设计的各种原则，具有高度的可靠性、稳定性和安全性.

数据集 S2 由从 GitHub 开源仓库获取的开源软件组成. 这类软件通常由个人或者小团队设计和开发，功能简单，但缺乏良好的模块化设计.

数据集 M1 由基于 SourceFinder 方法[43]提取的 136 个常见恶意软件的源码编译的可执行文件组成，主要类型包括蠕虫、僵尸网络、间谍软件、勒索软件以及混合型恶意软件.

根据是否存在真实值，聚类效果的评价可分为有监督评价和无监督评价 2 类[44]. 为了准确评估算法的聚类效果，本文采用依靠真实值的有监督评价指标作为模型的评价指标，并使用无监督评价指标对软件模块化质量进行评估.

1）有监督评价指标

在本文中，函数的真实标签和模型生成的标签都是人工赋予的用来区分簇的无语义标签，因此在评估之前需要使用 Hungarian 算法[45]实现真实标签和预测标签的最优匹配.

① *Prc*（precision）. *Prc* 是用来表示正样本结果中预测的准确程度.

$$Prc=\frac{1}{n}\sum_{i=1}^{n}\frac{TP_i}{TP_i+FP_i}, \tag{11}$$

其中 TP_i 表示正确分配到模块 i 的样本数量，FP_i 表示被预测为正确的负样本数量，n 表示簇的个数.

② *F*1. *F*1 是统计学中用来衡量分类模型精确度的一种指标.

$$\begin{aligned} Prc&=\frac{\sum TP}{\sum TP+\sum FP},\\ Recall&=\frac{\sum TP}{\sum TP+\sum FN},\\ F1&=\frac{2\times Prc\times Recall}{Prc+Recall}. \end{aligned} \tag{12}$$

③ *NMI*（normalized mutual information）. *NMI* 是用于度量 2 个聚类结果相似程度的指标，可以将预测结果与真实值进行比较以对聚类结果进行评估.

$$NMI=\frac{\sum d_{h,l}\lg\left(d_{h,l}/d_hc_l\right)}{\sqrt{\left(\sum_h \mathrm{d}_h\lg(d_h/d)\right)\left(\sum_l c_l\lg(c_l/d)\right)}}, \tag{13}$$

其中 d_h 和 c_l 分别表示类 h 和簇 l 中的元素个数，$d_{h,l}$ 表示既属于类 h 又属于簇 l 的元素个数.

2）无监督评价指标

① *CP*（compactness）. *CP* 是通过计算簇中样本点与簇中心的距离来度量簇的紧密程度的指标. 高内聚的软件模块内的函数应该具有相似的嵌入表示，因此 *CP* 可以作为度量模块内聚度的指标，高内聚的模块 *CP* 值倾向于 0.

$$CP=\frac{1}{K}\sum_{k=1}^{K}\frac{1}{|\Omega_k|}\sum_{\boldsymbol{x}_i\in\Omega_k}\|\boldsymbol{x}_i-\mu_k\|, \tag{14}$$

其中 Ω_k 表示簇 k 的样本点集合，$\boldsymbol{x}_i$ 表示 k 中的某个样本，μ_k 表示簇中心，K 表示簇的个数.

② *SP*（separation）. *SP* 通过计算各个簇的中心点

之间的距离来表示簇的离散程度. 不同模块在嵌入表示后应该具有较大的差异性，因此 *SP* 可用来对软件模块间的耦合度进行度量，*SP* 值越大表示模块耦合度越低.

$$SP=\frac{2}{K^2-K}\sum_{i=1}^{K}\sum_{j=i+1}^{K}\left\|\mu_i-\mu_j\right\|. \tag{15}$$

③ *DVI*(dunn validity index). *DVI* 是度量簇内点的紧密程度和簇间离散程度的综合指标，它是通过计算任意 2 个簇的样本点的最小距离与任意簇中 2 个样本点的最大距离的比值得到的.*DVI* 的值越大表示簇间间距越大、簇内距离越小，软件设计越符合高内聚、低耦合的设计思想.

$$DVI=\frac{\min\limits_{\substack{0<m\neq n\leqslant K\\ \forall \boldsymbol{x}_i\in\Omega_m,\forall \boldsymbol{x}_j\in\Omega_n}}\left\{\left\|\boldsymbol{x}_i-\boldsymbol{x}_j\right\|\right\}}{\max\limits_{0<\beta\leqslant K,\forall \boldsymbol{x}_i,\boldsymbol{x}_j\in\Omega_\beta}\left\{\left\|\boldsymbol{x}_i-\boldsymbol{x}_j\right\|\right\}}. \tag{16}$$

4.3 实验 1.特征选择评估

为了评估不同的特征对模块划分结果的影响. 本节设计了一组实验，实验分为对照组和实验组，对照组采用 GEBCM 的方法将 4 个特征进行组合，实验组则分别使用其中一个特征. 从数据集 S1 和 S2 中，各随机选取 3 个样本作为实验对象，详见表 3. 最后，通过比较对照组和实验组的 *F*1 值，来评估不同特征对程序模块划分结果的影响.

Table 3 Experimental Sample Information

表 3 实验样本信息

样本	链接
LogMonitor	https://github.com/microsoft/windows-container-tools
Vcperf	https://github.com/microsoft/vcperf
Yara32	https://github.com/VirusTotal/yara
Hp-socket	https://github.com/ldcsaa/HP-Socket
Omgocrprj	https://github.com/imil/OMGOCRCAL
TetrisSolver	https://github.com/zsebastian/AI-Homework

图 5 展示了不同的输入特征对模型性能的影响. 实验结果表明，相较于单个特征，使用 GEBCM 的组合特征方法具有更高的 *F*1 值. 进一步分析发现，单个特征在不同的样本上表现出的 *F*1 值差异较大. 造成这种现象的主要原因可能是单个特征存在的信息不足，无法充分描述函数间复杂的关系，从而影响模型的准确性.

4.4 实验 2.权重参数调整评估

特征权重参数调整是软件模块划分任务中的重要环节. 其主要目的在于识别关键特征，以优化模型

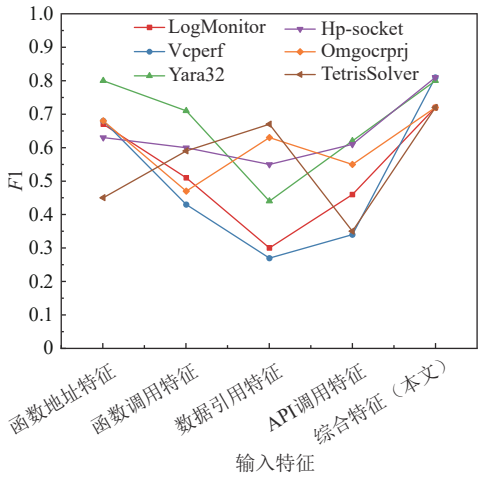


Fig. 5 Experimental results of different features

图 5 不同特征的实验结果

预测的准确性和泛化能力. 由于软件模块划分属于无监督任务，难以定义目标函数，因此输入特征数据的权重通常被设置为超参数. 本文旨在评估 SE block 权重参数调整方法与贝叶斯优化[46]、遗传算法[47]和梯度优化[48]3 种常用的超参数优化方法之间的差异. 为此，本文选择了这 3 种典型的超参数优化方法作为实验组，与 SE block 权重参数调整方法进行对比.

图 6 展示了不同权重参数优化方法的实验结果. 综合来看，基于 SE block 的权重参数调整方法在 6 个样本中表现出的 *F*1 值最佳，其他超参数优化方法在不同的样本上波动较大，这是因为不同样本之间的特征差异较大造成的. 需要注意的是，虽然在个别样本上不做任何优化的特征组合能取得最高的 *F*1 值，但这并不意味着在大多数情况下不需要进行复杂的参数优化. 事实上，参数优化可以帮助模型快速识别

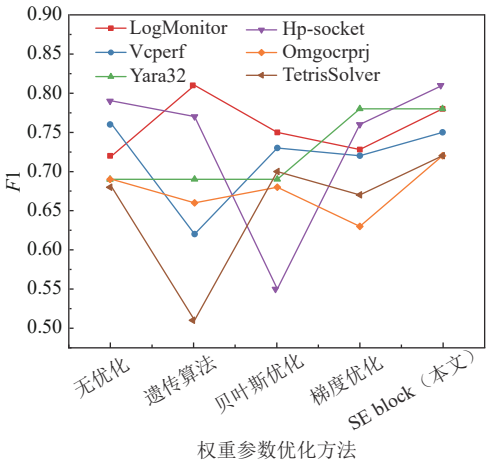


Fig. 6 Experimental results of different weighting parameter adjustment methods

图 6 不同权重参数调整方法的实验结果

重要特征，捕获复杂的模块化信息，从而提高模型的性能.

4.5 实验 3.聚类方法评估

为了全面评估 GEBCM 方法的性能，本节在数据集 S1 和 S2 上与其他主流的二进制模块方法进行比较. 由于算法 BCD 和 ModX 未提供可直接使用的源代码，本文根据相关文献的描述，自行实现了一个简单的原型版本. 在实验过程中，首先对输入的二进制可执行文件进行混合预处理，以获得每个方法所需的数据，然后使用 5 种方法对软件进行模块划分，并使用相同的评估指标对实验结果进行评估.

表 4 记录了 5 种方法在 2 个数据集上的实验结果. 从直观上看，无论是哪个数据集，GEBCM 的 *Prc*, *NMI*, *F*1 指标都优于其他方法. 同时，使用集成社区发现的 DeMal 方法要优于使用单一算法的 BCD 和 ModX 方法. 通过对比 DAEGC 和 GEBCM 结果可以发现，在加入节点排名和随机扰动后，模型效果有所提升. 这表明，节点排名可以帮助模型更好地理解不同节点之间的重要性和关系，而随机扰动则可以增加数据的多样性，提高模型的泛化能力. 需要注意的是，这 2 种基于图嵌入方法的分解效果均好于基于社区发现的方法. 但由于图嵌入方法中存在复杂的网络架构和大量的计算，因此也产生了更大的时间开销.

Table 4 Experimental Results on S1 and S2

表 4 在 S1 和 S2 上的实验结果

数据集	方法	*Prc*	*NMI*	*F*1	平均时间/s
S1	BCD	0.623	0.505	0.641	0.05
	ModX	0.648	0.547	0.687	0.03
	DeMal	0.651	0.595	0.704	8.79
	DAEGC	0.738	0.676	0.778	12.15
	GEBCM	0.783	0.737	0.802	15.67
S2	BCD	0.539	0.498	0.589	0.04
	ModX	0.592	0.512	0.635	0.03
	DeMal	0.551	0.495	0.643	7.63
	DAEGC	0.668	0.676	0.657	11.28
	GEBCM	0.703	0.687	0.701	14.53

表 5 展示了 GEBCM 方法在数据集 S1 中，以 *F*1 为参考对样本进行划分所得到的统计值. 通过对比结果，发现具备高 *SP*, *DVI* 和低 *CP* 的软件在各项评估指标上均表现优异，这说明符合软件工程设计原则的软件模块划分结果具有较高的置信度.

为了尽可能提取与程序结构相关的特征，实验 1~3 在虚拟环境中使用了 Pin 工具来记录程序的执行轨迹，以恢复部分间接调用关系. 为了深入研究间接调用对程序模块划分的影响，本实验从数据集 S1 中随机选择了一部分样本，并将这些样本在执行动态分析前后的模块划分结果进行了对比.

Table 5 Module Independence Assessment on S1

表 5 S1 的模块独立性评估

*F*1 分位数	*Prc*	*NMI*	*F*1	*CP*	*SP*	*DVI*
前 25%	0.809	0.787	0.864	0.152	1.384	0.694
前 50%	0.755	0.725	0.805	0.161	1.372	0.524
平均	0.783	0.737	0.802	0.176	1.365	0.429

实验结果如图 7 所示，横坐标表示恢复的间接调用数量在所有函数调用数量中的占比(间接调用占比)，纵坐标表示加入动态分析后的 *F*1 值与 2 种情况下 *F*1 值和的比值(动态占比). 可以观察到，当间接调用占比较小时，样本在加入间接调用后的 *F*1 值表现不稳定. 然而，随着间接调用占比的增加，加入间接调用后的 *F*1 值始终优于仅基于静态分析的情况. 这表明间接调用的恢复显著增强了模块划分的结果，进一步强调了动态分析在逆向工程中的关键作用.

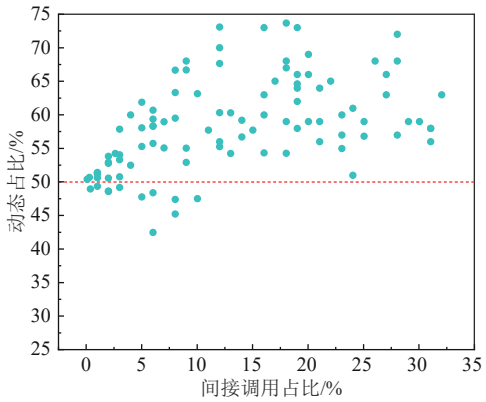


Fig. 7 Impact of indirect call recovery on module partitioning effectiveness

图 7 间接调用恢复对模块划分效果的影响

4.6 实验 4. 扩展性评估

为了评估 GEBCM 在处理恶意代码方面的扩展性能力，本文在恶意软件构成的数据集 M1 上进行了实验，并与目前主流的恶意软件模块分解工作 DeMal 进行了对比. 为了充分提取函数关系，实验设定恶意代码不采取代码混淆、加壳、加密等自保护手段，以确保实验结果更加准确可靠.

表 6 展示了 2 种方法在 M1 数据集上的实验结果. 值得注意的是，GEBCM 在该数据集上的 *F*1 值达到了 0.692，相较于 DeMal，*NMI* 值提高了约 27 个百

Table 6 Experimental Results on M1 Dataset

表 6 在 M1 数据集上的实验结果

方法	*Prc*	*NMI*	*F*1	平均时间/s
DeMal	0.571	0.532	0.633	6.58
GEBCM	0.612	0.674	0.692	14.14

分点. 这一结果表明, GEBCM 在处理恶意代码方面具有良好的可扩展性和高效性, 能够有效地识别出恶意代码中的关键函数和模块信息.

4.7 案例研究

本文 4.3~4.6 节对 GEBCM 方法的性能和扩展性进行了定量评估. 本节以 POSGrabber 为例, 深入研究模块划分的效果.

POSGrabber 是一款窃取用户数据的恶意软件, 它从 POS 的内存中抓取未加密的卡数据并写入文件, 然后发送到异地服务器. 其模块划分结果如表 7 所示, 可以看出, 属于同一模块的函数在模块划分后仍倾向于聚集在一起, 但部分模块中仍混杂了其他模块的函数. 下面以模块 2 为例深入研究造成这种现象的原因. 该模块中除了含有大量的 POSGrabber 主逻辑模块的函数外, 还包含了函数 *BeginInjection* 和 *Memset*. 通过审查源码发现, 函数 *Memset* 共被调用了 58 次, 其中 20 次源自 POSGrabber 模块的函数, 远超其他模块的调用次数, 因此系统将 *Memset* 划分到该模块. 而函数 *BeginInjection* 共被调用 3 次, 其中 2 次源自 POSGrabber 模块、1 次源自 InjectSection 模块, 相差并不大, 但其函数内部除了使用系统自带的 API 函数外, 还使用了 2 次函数 *Memset*, 故系统认定 *BeginInjection* 与 POSGrabber 模块的函数联系更加紧密, 并将 *BeginInjection* 划分到该模块. 这表明本文方法可以捕捉到函数之间的关联性, 进而恢复程序的模块化结构.

与 DAEGC 方法相比, GEBCM 方法在节点的表示过程中加入了函数排名, 旨在更准确地划分公共功能类函数, 即 Common 模块中的函数. 通过对比 DAEGC 方法在该样本上的划分结果(*F*1=0.62)发现, DAEGC 方法将 Common 模块的 21 个函数中的 11 个划分到了其他 4 个模块, 而 GEBCM 中仅有 4 个函数被划分到其他 3 个模块中. 同时, 由于对 Common 模块函数的准确划分, 减少了对其他函数的影响, 使得其他模块中的函数也更倾向于聚集在一起, 从而提高了整体的准确性. 这意味着 GEBCM 能够更好地恢

Table 7 Results of Modularization Partitioning for POSGrabber

表 7 POSGrabber 模块划分的结果

模块序号	原始模块	函数
1	Common	*Atoi*, *Base64_decode*, *Base64_encode*, *Check_digit*, *CopyTill*, *Do_rand*, *Xor*, *From_hex*, *GetDebugPrivs*, *Url_decode*, *GetParentProcessId*, *Srand*, *IsPCx64*, *Ny_toLower*, *Rand*, *RandStrA*, *RandStrW*
	GetOSInfo	*CryptEncodeCombine*
	HttpInteract	*DownloadFile*, *GetDownloadFileSize*, *GetCookie*
2	POSGrabber	*AddCache*, *AddItem*, *DigitsLen*, *IsDigit*, *IsEndDataValid*, *IsNameChar*, *IsValidCC*, *EntryPoint*, *ScanMemory*, *SkipProcess*, *Track1NameLength*, *TrackSearch*, *TrackSearchNoSentinels*
	InjectSection	*BeginInjection*
	Common	*Memset*
3	GetOSInfo	*GatherInfo*, *GetOSVersion*, *GetIdleTime*, *GetProcList*
	HttpInteract	*HttpMain*
	Remotescript	*Uninstall*
4	Remotescript	*ExecCommands*, *Downloader*, *Update*
5	CRC32	*PartialCRC*, *Reflect*
	Infect	*Infect*
	Common	*GenUnique*, *MonitorShutdown*
6	Infect	*CheckIfInfected*, *ProtectRegistry*
7	InjectSection	*CopySectionToProcess*, *MonitorChild*
	POSGrabber	*ScanMemoryLoop*, *DetectShutdown*
	Infect	*DisableOpenFileWarning*
	Common	*Memcpy*
8	HttpInteract	*HttpInteract*

复程序的模块结构和功能特征.

5 讨 论

本节首先讨论了软件模块划分的应用情况，然后阐述了该方法的局限性以及未来改进的方向.

5.1 应 用

1)加速逆向分析过程. 随着软件功能和复杂度的不断增长，可执行文件的规模也在不断扩大，这严重制约了逆向分析的速度和准确性. 基于二进制代码模块划分技术，分析人员可以将软件划分为相对独立的功能模块并恢复程序的模块化控制结构，有助于快速定位特定功能的代码位置，进而加快逆向分析的过程.

2)模块相似性分析. 软件同源性分析是指通过分析代码的特征信息，包括序列信息和图结构等，通过计算特征的相似性来判断同源性. 然而，随着软件函数规模的扩大，计算的复杂度呈指数级增长，效率也无法得到保证. 软件模块划分为同源性分析提供了一种新的途径，分析人员可以将函数规模较大的软件按照功能进行划分，然后对每个模块进行特征提取并计算相似性. 通过这种方法，可以排除库函数和其他通用函数等软件“噪声”的干扰，提高同源性判定的准确率，并极大地缩短分析时间.

3)二进制代码复用. 二进制代码复用是指从目标软件中提取功能较为完整的模块进行重新封装以供复用. 在恶意软件分析领域，模块划分可以帮助分析人员提取恶意软件中的加密、打包、通信等模块，通过在安全可控的环境中实现模块复用以提取执行代码、分析通信协议、捕获流量特征，从而提高恶意软件自动化分析的速度. 在软件漏洞修复领域，分析人员可以通过模块划分来对闭源软件的功能模块进行比对分析，以获取未公开漏洞的补丁，并通过代码重用的方式进行漏洞修补，从而提高软件的健壮性.

5.2 局限性与展望

1)相比于其他二进制模块化方法，使用 GEBCM 处理函数规模较大的样本时需要更多的时间开销. 这种开销主要源于在生成节点嵌入的过程中存在大量的复杂计算. 一种可能的解决方法是采用一些优化技术来降低计算复杂度，例如基于采样的方法来减少计算量，或者使用基于近似的方法来降低精度要求，从而加快计算速度. 此外，在节点嵌入时，可以使用适当的参数和算法来调整节点表示的准确性和可靠性，以实现模块划分的准确性和计算时间成本之间的平衡.

2)为了提取函数的语义信息，GEBCM 提取了每个函数调用的 API 序列，并将调用相似 API 的 2 个函数定义为具有相似的功能语义. 实际上，这种方法存在较大的局限性. 首先这种方法依赖于 API 文档的完整性和准确性. 如果 API 文档中缺少某些 API，就会对函数的语义特征造成影响. 其次，混淆技术可以通过更改 API 调用来混淆代码，从而使函数的语义特征不可靠. 因此，未来需要寻找能够稳定表征函数功能语义信息的特征来解决这些问题.

3)虽然软件研究人员提出了各种软件模块化设计原则，但是目前还没有定量衡量软件模块划分好坏的方法. 本文提出的衡量软件内聚度和耦合度的评估指标也只是提供一组相对指标，很难证明这些指标反映了真正的模块质量. 未来，我们将在大量的商业开源软件上进行更加深入的研究，对文中的评估指标进行实证研究并提出一组评价参考区间，帮助软件分析人员快速了解软件模块设计情况.

6 结 论

本文提出了一种基于图嵌入的二进制代码模块划分方法 GEBCM.该方法首先将二进制代码抽象为属性图，然后通过带有注意力机制和排名机制的图嵌入聚类方法对函数进行聚类，以实现软件模块的划分. 在本文的多项评估实验中，GEBCM 表现出良好的精确度，能够较为准确地实现二进制函数的聚类. 此外，二进制代码模块划分作为一种通用方法，在逆向工程、同源性分析、二进制代码重用等领域也具有重要应用.

作者贡献声明：孙华启提出了研究思路、完成大部分实验并撰写论文；康绯提供指导并修改论文；舒辉提出了实验设计方法；黄宇垚完成了部分实验和论文校对；卜文娟优化了论文思路.

参 考 文 献

[1] Statista. Number of available applications in the Google play store from December 2009 to March 2023 [EB/OL]. [2023-04-17]. https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store

[2] Mitre. Common vulnerabilities and exposures [EB/OL]. [2023-03-04].https://cve.mitre.org

[3] AV-TEST. Malware [EB/OL]. [2023-04-17]. https://www.av-test.

org/en/statistics/malware

[4] Schwartz E J, Avgerinos T, Brumley D. All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask) [C] //Proc of the 2010 IEEE Symp on Security and Privacy. Piscataway, NJ: IEEE, 2010: 317−331

[5] D'Elia D C, Coppa E, Nicchi S, et al. SoK: Using dynamic binary instrumentation for security (and how you may get caught red handed) [C]//Proc of the 2019 ACM Asia Conf on Computer and Communications Security. New York: ACM, 2019: 15−27

[6] Chen Jinyin, Hu Keke, Yu Yue, et al. Software visualization and deep transfer learning for effective software defect prediction [C] //Proc of the 42nd ACM/IEEE Int Conf on Software Engineering. Los Alamitos, CA: IEEE Computer Society, 2020: 578−589

[7] Yasunaga M, Liang P. Graph-based, self-supervised program repair from diagnostic feedback [C] //Proc of the 37th Int Conf on Machine Learning. New York: ACM, 2020: 10799−10808

[8] Gibert D, Mateu C, Planes J. The rise of machine learning for detection and classification of malware: Research developments, trends and challenges[J]. Journal of Network and Computer Applications, 2020, 153: 102526

[9] Bavota G, Oliveto R, Gethers M, et al. Methodbook: Recommending move method refactorings via relational topic models[J]. IEEE Transactions on Software Engineering, 2014, 40(7): 671−694

[10] Sarhan Q I, Ahmed B S, Bures M, et al. Software module clustering: An in-depth literature analysis[J]. IEEE Transactions on Software Engineering, 2022, 48(6): 1905−1928

[11] Zhou Yang, Cheng Hong, Yu J X. Graph clustering based on structural/attribute similarities[J]. Proceedings of the VLDB Endowment, 2009, 2(1): 718−729

[12] Karande V, Chandra S, Lin Z, et al. BCD: Decomposing binary code into components using graph-based clustering [C] //Proc of the 2018 on Asia Conf on Computer and Communications Security. New York: ACM, 2018: 393−398

[13] Yang Can, Xu Zhengzi, Chen Hongxu, et al. ModX: Binary level partially imported third-party library detection via program modularization and semantic matching [C] //Proc of the 44th Int Conf on Software Engineering. New York: ACM, 2022: 1393−1405

[14] Xia Hong, Zhang Yongkang, Chen Yanping, et al. Software module clustering using the hierarchical clustering combination method [C] //Proc of the 7th Int Conf on Cloud Computing and Big Data Analytics. Piscataway, NJ: IEEE, 2022: 155−160

[15] Papachristou M. Software clusterings with vector semantics and the call graph [C] //Proc of the 27th ACM Joint Meeting on European Software Engineering Conf and Symp on the Foundations of Software Engineering (ESEC/FSE). New York: ACM, 2019: 1184−1186

[16] Pan Weifeng, Song Beibei, Li Kangshun, et al. Identifying key classes in object-oriented software using generalized k-core decomposition[J]. Future Generations Computer Systems, 2018, 81: 188−202

[17] Newman M E J. Fast algorithm for detecting community structure in network [J]. arXiv preprint, arXiv: cond-mat/0309508, 2003

[18] Newman M E J, Girvan M. Finding and evaluating community structure in networks [J]. arXiv preprint, arXiv: cond-mat/ 030 8217, 2003

[19] Candela I, Bavota G, Russo B, et al. Using cohesion and coupling for software remodularization: Is it enough[J]. ACM Transactions on Software Engineering & Methodology, 2016, 25(24): 1−28

[20] Huang Yuyao, Shu Hui, Kang Fei. DeMal: Module decomposition of malware based on community discovery[J]. Computers & Security, 2022, 117: 102680

[21] Lian Wen, Kirk D, Dromey R G. Software systems as complex networks [C] //Proc of the 6th IEEE Int Conf on Cognitive Informatics. Piscataway, NJ: IEEE, 2007: 106−115

[22] Li Hao, Wang Tian, Xu Xinxin, et al. Modeling software systems as complex networks: Analysis and their applications [J]. Mathematical Problems in Engineering, 2020, 2020: 1−7

[23] Nielson H R. Principles of Program Analysis [M]. Berlin: Sprin ger, 1999

[24] Hey-rays. IDA Pro-A powerful disassembler and a versatile debugger [EB/OL]. [2023-04-17].https://www.hex-rays.com/ida-pro

[25] Radare2. Libre reversing framework for Unix geeks [EB/OL]. [2023-04-17].https://github.com/radareorg/radare2

[26] NSA. A software reverse engineering suite of tools developed by, NSA's research directorate in support of the cybersecurity mission [EB/OL]. [2023-04-17].https://github.com/NationalSecurityAgency/ghidra/releases

[27] Henderson A, Prakash A, Yan L, et al. Make it work, make it right, make it fast: Building a platform-neutral whole-system dynamic binary analysis platform [C] //Proc of the 2014 Int Symp on Software Testing and Analysis. Piscataway, NJ: IEEE, 2014: 248−258

[28] Song D, Brumley D, Yin Heng, et al. BitBlaze: A new approach to computer security via binary analysis [C] //Proc of the 4th Int Conf on Information Systems Security. Berlin: Springer, 2008: 1−25

[29] Yan S, Wang Ruoyu, Salls C, et al. SOK: (State of) The art of war: Offensive techniques in binary analysis [C] //Proc of the 37th IEEE Symp on Security and Privacy. Piscataway, NJ: IEEE, 2016: 138−157

[30] Cai Hongyun, Zheng V W, Chang C C. A comprehensive survey of graph embedding: Problems, techniques and applications[J]. IEEE Transactions on Knowledge & Data Engineering, 2018, 30(9): 1616−1637

[31] Wang Daixin, Cui Peng, Zhu Wenwu. Structural deep network embedding [C] //Proc of the 22nd ACM SIGKDD Int Conf on Knowledge Discovery and Data Mining. New York: ACM, 2016: 1225−1234

[32] Cao Shaosheng, Lu Wei, Xu Qiongkai. Deep neural networks for learning graph representations [C] //Proc of the 30th AAAI Conf on Artificial Intelligence. Palo Alto, CA: AAAI, 2016: 1145−1152

[33] Kipf T N, Welling M. Semi-supervised classification with graph convolutional networks [J]. arXiv preprint, arXiv: 1609.02907, 2016

[34] Velikovi P, Cucurull G, Casanova A, et al. Graph attention networks [J]. arXiv preprint, arXiv: 1710.10903, 2017

[35] Wang Chun, Pan Shirui, Hu Ruiqi, et al. Attributed graph clustering: A deep attentional embedding approach [C] //Proc of the 28th Int Joint Conf on Artificial Intelligence. San Francisco, CA: Morgan Kaufmann, 2019: 3670−3676

[36] Intel. Pin-A dynamic binary instrumentation tool [EB/OL]. [2023-04-07].https://www.intel.com/content/www/us/en/developer/articles/tool/

pin-a-dynamic-binary-instrumentation-tool.html
[37] Levine J R. Linkers and Loaders [M]. San Francisco, CA: Morgan Kaufmann, 1999
[38] Hu Jie, Li Shen, Samuel A, et al. Squeeze-and-excitation networks[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2020, 42(8): 2011−2023
[39] Garey M R, Johnson D S. Computers and Intractability: A Guide to the Theory of NP-Completeness [M]. New York: W. H. Freeman and Company, 1979
[40] Github. Dexter-POSGrabber [EB/OL]. [2023-04-17].https://github.com/whobin/Dexter-POSGrabber
[41] Xie Junyuan, Girshick R, Farhadi A. Unsupervised deep embedding for clustering analysis [C] //Proc of the 33rd Int Conf on Machine Learning. New York: ACM, 2016: 478−487
[42] Xiang Shunhua, Nie Feiping, Zhang Changshui. Learning a Mahalanobis distance metric for data clustering and classification[J]. Pattern Recognition, 2008, 41(12): 3600−3612
[43] Rokon M, Islam R, Darki A, et al. SourceFinder: Finding malware source-code from publicly available repositories [C] //Proc of the 23rd Int Symp on Research in Attacks, Intrusions and Defenses (RAID 2020). Berkeley, CA: USENIX Association, 2020: 149−163
[44] Fahad A, Alshatri N, Tari Z, et al. A survey of clustering algorithms for big data: Taxonomy and empirical analysis[J]. IEEE Transactions on Emerging Topics in Computing, 2014, 2(3): 267−279
[45] Kuhn H W. The Hungarian method for the assignment problem[J]. Naval Research Logistics, 2010, 52(1/2): 7−21
[46] Snoek J, Larochelle H, Adams R P. Practical Bayesian optimization of machine learning algorithms [C] //Proc of the 25th Int Conf on Neural Information Processing Systems. New York: Curran Associates, 2012: 2951−2959
[47] Xiao Xueli, Yan Ming, Basodi S, et al. Efficient hyperparameter optimization in deep learning using a variable length genetic algorithm [J]. arXiv preprint, arXiv: 2006.12703, 2020
[48] Maclaurin D, Duvenaud D, Adams R P. Gradient-based hyperparameter optimization through reversible learning [J]. arXiv preprint, arXiv: 1502.03492, 2015



Sun Huaqi, born in 1994. Master. His main research interests include reverse engineering and program analysis.
孙华启, 1994 年生. 硕士. 主要研究方向为逆向工程、程序分析.



Kang Fei, born in 1972. PhD, professor. Her main research interests include cyberspace security, reverse engineering, and software protection.
康　绯, 1972 年生. 博士, 教授. 主要研究方向为网络空间安全、逆向工程、软件保护.



Shu Hui, born in 1974. PhD, professor, PhD supervisor. His main research interests include cyberspace security, program analysis, and reverse engineering.
舒　辉, 1974 年生. 博士, 教授, 博士生导师. 主要研究方向为网络空间安全、程序分析、逆向工程.



Huang Yuyao, born in 1996. PhD candidate. His main research interests include network protocol reverse-engineering and program analysis.
黄宇垚, 1996 年生. 博士研究生. 主要研究方向为网络协议逆向工程和程序分析.



Bu Wenjuan, born in 1984. PhD candidate. Her main research interests include cyberspace security and reverse engineering.
卜文娟, 1984 年生. 博士研究生. 主要研究方向为网络空间安全、逆向工程.