基于软件网络加权 k-核分析的关键类识别方法

潘伟丰[1],宋贝贝[1],胡 博[2],李 兵[3],姜 波[1]
(1. 浙江工商大学计算机与信息工程学院,浙江杭州 310018;2. 金蝶国际软件集团金蝶研究院,广东深圳 518057;
3. 武汉大学国际软件学院,湖北武汉 430072)

摘 要: 识别关键类可以帮助人们尽快理解不熟悉的软件系统. 尽管目前已有一些关键类识别方面的工作,但是大部分方法构建的依赖图是无权的,未考虑类之间交互的种类及次数. 有鉴于此,提出了一种基于软件网络加权 k-核分析的关键类识别方法. 首先,用加权类耦合网络抽象类(接口)、类(接口)间的交互及其次数;然后,引入加权 k-核分解方法计算类的加权核数;最后,以加权核数作为类重要性的量度指标,降序排列所有类,并通过过滤得到候选关键类. 真实软件上的数据实验验证了本文方法的有效性.

关键词: 关键类; 加权 k-核分解; 软件网络; 程序理解
中图分类号: TP311.5 **文献标识码:** A **文章编号:** 0372-2112 (2018)05-1071-07
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2018.05.007

Identifying Key Classes Based on Weighted k-Core Analysis of Software Networks

PAN Wei-feng[1], SONG Bei-bei[1], HU Bo[2], LI Bing[3], JIANG Bo[1]
(1. *School of Computer Science and Information Engineering, Zhejiang Gongshang University, Hangzhou, Zhejiang* 310018, *China*;
2. *Kingdee Research, Kingdee International Software Group Co. Ltd, Shenzhen, Guangdong* 518057, *China*;
3. *International School of Software, Wuhan, Hubei* 430072, *China*)

Abstract: Identifying key classes can help software engineers understand software systems that they previously were not familiar with. Though there are some methods on the identification of key classes, a majority of them use un-weighted dependency graphs, neglecting the coupling types and frequencies. In this paper, we propose a method to identify the key classes based on weighted k-core analysis of software networks. First, it uses a weighted class coupling network to represent classes (interfaces) and their couplings and coupling frequencies. Then, we introduce the weighted k-core decomposition method to compute the weighted coreness for each class (interface). Finally, we use the weighted coreness to quantify the importance of classes (interfaces) and sort them in a descending order with respect to their weighted corenesses. And the top-ranked classes (interfaces) will serve as the key class candidates. Empirical results show the effectiveness of our approach.

Key words: key class; weighted k-core decomposition; software network; program comprehension

1 引言

软件维护是软件生存期中一项频繁进行的活动,自其第一个版本发布之后便开始[1],占了软件总预算的90%以上[2]. 要维护软件,首先必须理解软件[3,4]. 然而理解软件并不是一件容易的事情[5],特别是当软件变得异常复杂而分配给维护的人力、物力等资源又比较稀缺的时候,理解软件更是难上加难. 类是面向对象软件的基本组成单元,是理解和维护的主要对象[6]. 关键类作为实现系统核心功能的类,更是软件理解的重点[5,6]. 选择从关键类开始分析和理解软件已成为理解复杂软件的有效方式之一[5~10]. 因此,提供有效的

收稿日期:2016-11-16;修回日期:2017-04-13;责任编辑:蓝红杰
基金项目:国家自然科学基金(No. 61572371);浙江省自然科学基金(No. LY15F020004);浙江省科技厅公益技术研究项目(No. 2014C23008)

技术,识别软件中的关键类,对于分析和理解软件,进而控制和降低维护费用具有重要意义.

目前,软件关键类识别方面已有一些工作,但是仍然比较欠缺. Inoue 等人[7]使用构件图(component graph)抽象构件及构件间的使用关系,并使用 PageRank 算法的一个修改版本计算节点及边的权重,实现构件排序,最终用于构件检索. Zaidman 和 Demeyer[8]通过运行一组测试用例收集软件的执行轨迹,进而构建类依赖图并计算动态耦合度量值,最后利用 HITS(hyperlink-induced topic search)算法识别关键类. 该方法在识别关键类上准确度较高,但是其时间和空间开销比较大. 在此基础上,Zaidman 等人[5]进一步提出了一种基于静态度量的关键类识别方法,通过构建静态类依赖图并使用 HITS 算法识别关键类,在一定程度上缓解了空间开销,但时间开销仍然比较大. 周毓明等人[9]用类依赖图抽象类及类间依赖关系,分别使用 PageRank 算法、HITS 和介数中心性识别关键类. 在此基础上,周毓明等人[10]进一步使用 h 指数及其衍生指数 a 指数量度类的重要性,取得了较好的效果. 姜淑娟等人[6]用状态转换模型抽象软件,并通过求解转换模型的唯一输入输出序列构造状态转换树,最后通过计算状态转换树节点的复杂度来量度类的重要性.

我们发现,上述工作提出的方法都包含两个关键步骤:依赖图的构建和类重要性的量度. 大部分方法构建的依赖图是无权的,未考虑类之间交互的种类及次数. 尽管周毓明等人在文献[10]中考虑了类间交互的种类及次数,但他们仅考虑了类间由于继承、方法参数、成员属性和方法返回值类型产生的交互,忽略了类间由于方法调用和局部变量产生的交互,也未考虑接口及类与接口间由于实现产生的交互. 同时,他们的方法在计算交互次数时,采用了简化处理,所得次数不太精确. 但是,从他们的结果可见,采用了加权依赖图得到的结果要优于 Zaidman 等人[5,8]提出的静态方法.

有鉴于此,本文提出了一种基于软件网络加权 k-核分析的关键类识别方法 ICAN(Identifying key Classes based on weighted k-core Analysis of software Networks). ICAN 用加权软件网络模型抽象类(接口)、类(接口)间的交互及其次数;同时,引入加权 k-核分解方法挖掘软件网络的层次结构进而得到类节点的加权核数;最终,以加权核数为排序标准,降序排列所有类节点,并通过过滤得到排名靠前的候选关键类. 数据实验部分,我们将 ICAN 方法运用于开源软件 Ant-1.6.1 和 JMeter-2.0.1 的关键类识别中,并与相关方法进行比较,验证了本文方法的有效性.

2 ICAN 方法

ICAN 方法主要包含 3 个部分:解析源代码获得软件结构信息;用加权网络形式化的表示软件的结构信息;引入加权 k-核分解方法计算网络中类节点的加权核数用以量度类的重要性,并按照加权核数降序排列类节点,过滤得到候选关键类. 以下各小节将详述各主要部分.

2.1 软件结构信息提取

我们主要以开源的 Java 软件作为研究对象. 软件结构信息提取是 ICAN 方法的第一步,指从软件的源代码(.java 文件)中提取各类软件元素(类、接口、属性、方法、局部变量等)和元素间的交互(类间继承、类对接口的实现、方法调用等)及其次数. 结构信息的提取由我们自主研发的软件网络分析平台 SNAP(software network analysis platform)[11]实现(由文献[11]中的 SSQAT 更名而来). SNAP 可以提取软件的结构信息,还可以实现一系列的网络分析及可视化.

2.2 加权软件网络模型

软件结构是影响软件质量的重要因素之一[4],主要包括软件元素及元素间的交互[11],表现为一种内部互连的复杂网络拓扑的形态. 我们将构建类粒度的加权软件网络形式化的抽象软件的拓扑结构信息.

定义 1 加权类耦合网络(weighted class coupling network,WCCN):

$$\mathrm{WCCN}=(N,E,\boldsymbol{P}), \tag{1}$$

其中,N 是网络的节点集,抽象软件中的类或接口(在不特别说明时,下文的类也包含了接口). E 是网络的边集,表示类节点间的交互关系. $\boldsymbol{P}$ 是一个对称矩阵,存储类间是否存在交互及交互的次数,即:若类 i 和类 j 之间存在交互,且交互了 n 次,则 WCCN 中相应的节点间存在一条无向边 (j,i)(边 (i,j) 与 (j,i) 是同一条边),边权 $P(i,j)=P(j,i)=n$;若类 i 和类 j 间不存在交互,则 $P(i,j)=P(j,i)=0$. Java 软件中,类 i 和类 j 在以下 7 种情况下会发生交互[12].

(1)继承关系:如果类 i 通过关键字 extends 继承另外一个类 j,反之亦然.

(2)实现关系:如果类 i 通过关键字 implements 实现了一个接口 j.

(3)形参关系:如果类 i 中的某一方法使用了类 j 的引用作为形式参数,反之亦然.

(4)成员属性关系:如果类 i 具有一个类型为类 j 的成员属性,反之亦然.

(5)方法调用关系:如果类 i 中的某一方法调用了类 j 对象的一个方法,反之亦然.

(6)局部变量关系:如果类 i 中的某一方法内定义了一个以类 j 为类型的局部变量,反之亦然.

(7)方法返回值关系:如果类 i 中的某一方法的返回值是类 j 的对象,反之亦然.

两个类之间可能同时存在不止一种的交互,同一

种交互也可能存在多次. 例如:类 i 和类 j 之间存在 1 次类继承交互、1 次形参关系交互和 2 次方法调用交互,则这两个类间总的交互次数是通过各种交互次数累加得到的,即 $P(i,j)=P(j,i)=1+1+2=4$. 与现有工作[7~10]相比,我们构建的加权软件网络能够更加准确的抽象软件的拓扑结构,不但考虑了类也考虑了接口,同时也考虑了类之间的各种交互关系及其次数.

图 1 给出了一个代码片段及其相应的 WCCN. 从图 1 可见,示例代码中存在接口 Animal、类 Mammal、类 Dog、类 Zoom 和类 Adoptor,所以 WCCN 中共有 5 个节点,分别代表这 5 个类. 同时,因为 Mammal 实现了 Animal 1 次,所以代表 Mammal 和 Animal 的节点间存在一条无向边,边权为 1;Dog 继承了 Mammal 1 次,所以代表 Dog 和 Mammal 的节点间存在一条无向边,边权为 1;Adoptor 包含一个 Dog 类的局部变量 myDog 且返回 Dog 类的对象,所以代表 Adoptor 和 Dog 的节点间存在一条无向边,边权为 2;Adoptor 拥有 1 个 Zoom 类的成员对象 myZoom,其方法 setZoom 以类 Zoom 的对象为参数,同时方法 say 调用了 myZoom 对象的 say 方法,所以代表 Adoptor 和 Zoom 的节点间存在一条无向边,边权为 3.

```
public interface Animal {
    public abstract void animalMethod()
}
class Mammal implements Animal {
    public void animalMethod() {
        System.out.println("Mammal");
    }
}
class Dog extends Mammal {
    public void animalMethod() {
        System.out.println("Dog");
    }
}
class Zoom {
    public void say() {
        System.out.println("Zoom");
    }
}
class Adoptor {
    private Zoom myZoom;
    public void setZoom(Zoom newZoom) {
        myZoom = newZoom;
    }
    public Dog getDog() {
        Dog myDog = new Dog();
        return myDog;
    }
    public void say() {
        myZoom.say();
    }
}
```

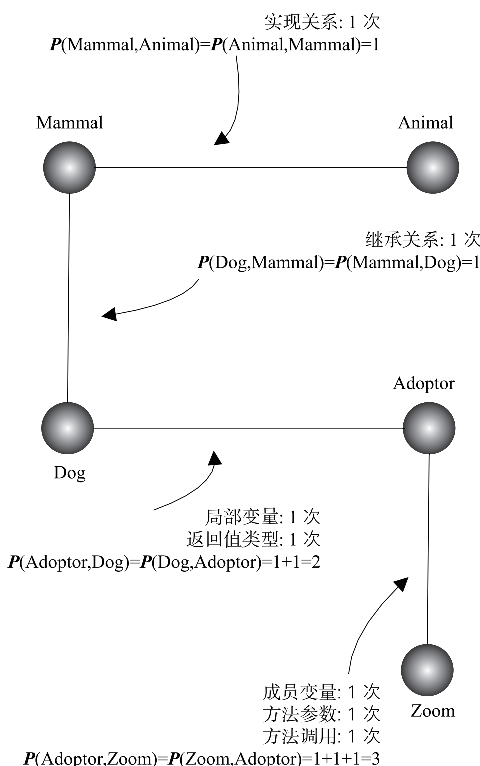

图1　代码片段及相应的WCCN

2.3　加权 k-核分解方法

本节中我们将引入加权 k-核分解方法($W_{k\text{-core}}$)[13],它是对 k-核分解方法[14]的扩展,广泛应用于挖掘加权复杂网络中的关键节点. $W_{k\text{-core}}$ 主要基于节点的加权度. 加权度是对度的扩展,在其计算过程中同时考虑了节点的度及与节点相连边的权值. 节点 i 的加权度 $k'(i)$ 定义为:

$$k'(i)=\sqrt{k(i)\sum_{j=1}^{k(i)}w_{ij}}\qquad(2)$$

其中,$k(i)$ 表示节点 i 的度,$\sum_{j=1}^{k(i)}w_{ij}$ 表示与节点 i 相连边的权值和. 在无权网络中,往往用 $w_{ij}=1$ 表示节点 i 和 j 间存在边,用 $w_{ij}=0$ 表示节点 i 和 j 间不存在边,所以在无权网络中 $k'(i)$ 可以简化为 $k(i)$;在加权网络中,$k'(i)$ 通常是一个小数,所以文献[13]将其离散化为与该小数最近的整数,即:

$$k'(i)'=\begin{cases}\lfloor k'(i)\rfloor, & k'(i)-\lfloor k'(i)\rfloor<0.5\\ \lceil k'(i)\rceil, & \text{otherwise}\end{cases}\qquad(3)$$

在节点加权度的基础上,我们参照 k-核分解方法类似的给出加权 k-核、节点加权核数、加权 k-壳等的定义. 令 $G(V,E)$ 是一个具有 $|V|$ 个节点和 $|E|$ 条边的加权网络,则加权 k-核的定义如下:

定义 2　加权 k-核(weighted k-core):

网络的加权 k-核是指反复去掉加权度值小于或等于 k 的节点及其连边后剩余的子图,并且该子图是具有这一性质的最大子图.

$W_{k\text{-core}}$ 采取类似 k-核分解的剪枝过程来得到网络的加权 k-核[15]:递归地移除加权度值小于 k 的所有节

点,直到剩余网络中所有的节点的加权度值至少为 k.

定义 3 加权核数(weighted coreness):

若节点 i 属于加权 k-核,但不属于加权($k+1$)-核,则节点 i 的加权核数值为 k. 加权核数的最大值 k_{max} 称为加权网络的核数. 节点的加权核数可以描述这个节点在这个网络中的深度. 显然地,一个连通图的所有节点属于加权 1-核.

定义 4 加权 k-壳(weighted k-shell):

加权网络的加权 k-壳由所有加权核数为 k 的节点及它们之间的连边构成.

图 2 给出了一个简单的例子用以说明 $W_{k\text{-core}}$ 的具体实施过程. 示例网络只有 5 个节点 4 条边(见图 2 最左边部分),除了节点 B 和 C 之间的边权值为 9,其余边的权值均为 1. 首先,我们计算所有节点的加权度,然后去除网络中加权度 <1 的所有节点(节点 E),得到加权 1-核. 然后,我们重新计算加权 1-core 中剩余节点的加权度,移除加权度 <2 的所有节点(节点 D),得到了加权 2-核. 接着,我们重新计算加权 2-core 中剩余节点的加权度,移除加权度 <3 的所有节点(节点 A),得到了加权 3-核. 这个过程不断重复直到网络中没有剩余节点.

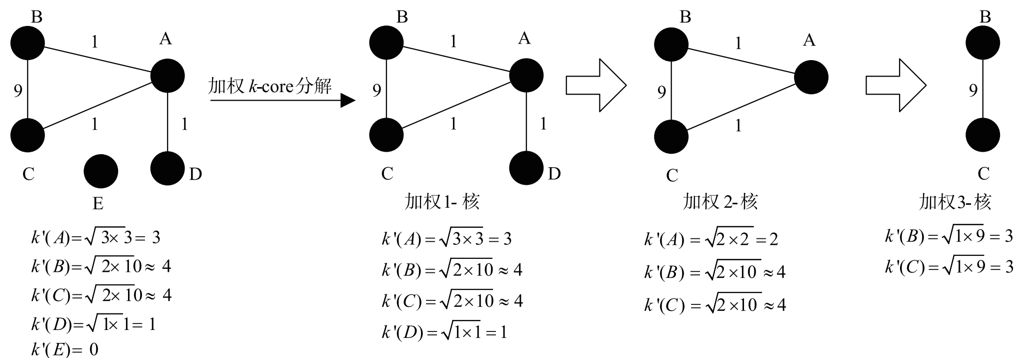

图2 $W_{k\text{-core}}$实施例

2.4 类重要性排序

我们将类节点的加权核数作为其重要性的度量指标,采用一个两阶段排序过程实现类重要性的排序.

(1)第一阶段:ICAN 采用快速排序算法[16]将每个 k-壳内的类节点按照离散化处理前的加权度(根据整个 WCCN 计算)降序排列.

(2)第二阶段:按照 k 值从大到小依次输出每个 k-壳内的经过第一阶段排序的类节点.

经过两阶段排序,ICAN 将输出一个经过排序的类列表,但是这个类列表包含了系统中所有的类,规模往往比较大. 为了得到候选的关键类,ICAN 还将使用一个过滤值 $p\in[0,1]$ 对类列表进行过滤,仅输出排名 top-$p\times|V|$($|V|$代表网络中的节点数)的节点作为候选关键类.

3 实验

本节我们将以真实的软件系统为实验对象,通过与现有工作的对比,验证本文方法的有效性.

3.1 研究问题

通过实验我们重点回答以下两个研究问题.

问题 1:ICAN 方法与现有工作相比是否更有效?

目前已有一些关键类识别方面的工作,我们希望知道 ICAN 方法是否比这些方法更有效.

问题 2:ICAN 方法的扩展性如何?

作为一个可行的关键类识别方法,往往会应用于不同规模的系统中,我们希望知道 ICAN 方法是否适用于发现大规模复杂软件中的关键类.

3.2 实验系统

本文以开源 Java 软件项目 Ant-1.6.1(http://azureus.sourceforge.net)和 JMeter-2.0.1(http://jmeter.apache.org)为实验对象展开研究. 其中,Ant 是一个跨平台的构建工具,可以实现软件项目的编译、测试、部署等;JMeter 是基于 Java 的压力测试工具. 这两个软件是目前关键类识别工作所用的标准测试集,所有的现有工作都用这两个软件验证其方法的有效性. 我们实验的软件环境为 Windows 7 64 位操作系统(旗舰版)和 JDK1.7;硬件环境为 Intel(R) Core(TM) i7-5600U@2.6 GHz CPU 和 8G 内存. 表 1 列出了这两个项目的一些统计数据. 其中,代码行排除了注释行及空行数,包数排除了外部导入的包,类数包含了内部类和接口的数量.

表 1 实验系统的统计数据

项目	版本号	代码行	包数	类数	方法/属性数
Ant	1.6.1	97,512	360	4,527	59,844
JMeter	2.0.1	78,304	290	3,477	45,936

3.3 实验过程

我们使用自主研发的软件网络分析平台 SNAP[11]解析 Java 软件源代码(.java 文件),得到软件的结构信息,进而构建 WCCN,并使用 $W_{k\text{-core}}$ 挖掘 WCCN 的层次

结构,得到类节点的加权核数,接着使用两阶段排序方法将节点按照其加权核数降序排列,并使用过滤值过滤得到候选的关键类.

3.4　结果分析

以下小节,我们将通过对实验结果的分析,回答在3.1节中提出的两个研究问题.

3.4.1　ICAN 方法与现有工作相比是否更有效?

目前关键类识别方面的工作主要是文献[5-10].本节中我们将 ICAN 方法与各文献中的最优方法进行比较,即 PageRank 算法[7]、IC_CC′+HITS[5,8]、h 指数[10]、a 指数(h 指数的衍生指数)[10],以及 UIO 方法(文献中的算法3)[6].

因为文献[7]并未使用本文的数据集进行关键类的识别,为了便于比较,我们实现了其 PageRank 算法.而其它几个方法因为无法获取其实验的详细过程性数据,也无法从公开渠道获取其实现源码,且其实现过程较复杂,本文并未实现,而是沿用其工作中给的结果.因此,为了便于比较,对于 ICAN 方法中的参数——过滤值 p,我们采用与文献中相同的设置,即:在与 UIO 比较时,分别设置 $p=0.1$ 和 $p=0.2$;而在与其它方法比较时,$p=0.15$.同时,为了公平的对各方法进行比较,我们使用了与文献中相同规模的数据集.

表2和表3罗列的是各个方法从两个系统中识别的关键类.其中,列“关键类”罗列的是这两个系统中已知的关键类[5,8];“√”表示相应方法识别出了该关键类;“×”表示相应方法未能识别出该关键类.因文献[10]并未给出 h 指数和 a 指数所能识别的关键类列表,因此我们未能将 h 指数和 a 指数的结果罗列在表2和表3中.

表2　Ant 中各方法识别的关键类

关键类	ICAN	PageRank	IC_CC′+HITS	ICAN	UIO	ICAN	UIO
	$p=0.15$			$p=0.1$		$p=0.2$	
Project	√	√	√	√	√	√	√
UnknownElement	√	√	√	√	√	√	√
Task	√	√	√	√	√	√	√
Main	×	×	√	×	×	√	×
IntrospectionHelper	√	√	√	√	√	√	√
ProjectHelper	√	×	√	√	√	√	√
RuntimeConfigurable	√	√	√	√	√	√	√
Target	√	√	√	√	√	√	√
ElementHandler	×	×	√	×	×	×	×
TaskContainer	×	√	×	×	×	×	×

表3　JMeter 中各方法识别的关键类

关键类	ICAN	PageRank	IC_CC′+HITS	ICAN	UIO	ICAN	UIO
	$p=0.15$			$p=0.1$		$p=0.2$	
AbstractAction	×	√	√	×	√	×	√
JMeterEngine	×	×	√	×	×	×	×
JMeterTreeModel	√	√	√	√	√	√	√
JMeterThread	√	×	√	√	√	√	√
JMeterGUIComponent	√	√	×	√	×	√	×
PreCompiler	×	×	√	×	×	×	×
Sampler	√	√	√	√	√	√	√
SampleResult	√	√	√	√	×	√	×
TestCompiler	√	×	√	√	√	√	√
TestElement	√	√	√	√	√	√	√
TestListener	√	√	√	×	×	√	√
TestPlan	√	×	√	√	√	√	√
TestPlanGui	×	×	√	×	×	×	×
ThreadGroup	√	×	√	√	√	√	√

为了比较上述方法在关键类识别上的性能差异,我们采用信息检索中常用的召回率(*recall*)和准确率(*precision*)作为评价标准.

$$recall=\frac{|R\cap K|}{|K|}\times 100\% \tag{4}$$

$$precision=\frac{|R\cap K|}{|R|}\times 100\% \tag{5}$$

其中,R 是某一个方法推荐的候选关键类集合,K 是系统中已知的关键类.所以 *recall* 表示推荐的关键类中已知关键类占已知关键类总数的比例,*precision* 表示推荐

的候选关键类中已知关键类占推荐总数的比例.

表4罗列了各方法所得结果的 *precision* 和 *recall*. 其中,p 是前文所述的过滤值. 从表4可见,在两个系统中,ICAN 方法的召回率和准确率都仅次于 IC_CC′+HITS,但是总体而言都优于其它5种方法. 在 Ant-1.6.1 中,ICAN 方法的召回率和准确率与 PageRank 相同,而在 JMeter-2.0.1 中,ICAN 方法的召回率和准确率均高 PageRank;在两个系统中,ICAN 方法的召回率和准确率均高于 h 指数;在 Ant-1.6.1 中,ICAN 方法的召回率与 a 指数同,但是准确率优于 a 指数;在 JMeter-2.0.1 中,ICAN 方法的召回率和准确率均优于 a 指数;当 $p=0.1$ 时,ICAN 方法的召回率和准确率在 Ant-1.6.1 中与 UIO 相同,而在 JMeter-2.0.1中均优于 UIO;当 $p=0.2$ 时,在两个系统中,ICAN 方法的召回率和准确率均优于 UIO. 在两个系统中,ICAN 方法的总耗时均不到1分钟,是一种比较高效的方法.

表4 相关工作比较

p 值	方法	Ant-1.6.1		JMeter-2.0.1	
		Recall (%)	*Precision* (%)	*Recall* (%)	*Precision* (%)
0.15	ICAN	70	7.03	71.43	29.76
	PageRank	70	7.03	50.00	20.83
	IC_CC′+HITS	90	47	93	46
	h 指数	50	5	50	6.6
	a 指数	70	7	57	7.5
0.1	ICAN	70	10.54	64.29	40.18
	UIO	70	10.54	57.14	28.88
0.2	ICAN	80	6.02	71.43	22.32
	UIO	70	5.3	64	16

通过分析我们发现,ICAN 方法在召回率和准确率方面次于 IC_CC′+HITS 的原因在于:Zaidman 等人采用的是一种动态分析方法,他们跟踪的只是测试用例执行轨迹上的类. 在 Ant-1.6.1 中,他们仅跟踪了127个类,约占我们跟踪类的19.13%;在 JMeter-2.0.1 中,他们跟踪了189个类,约占我们跟踪类的84.38%. 动态分析方法 IC_CC′+HITS 在召回率和准确率上均优于静态分析方法 ICAN,但是收集测试用例的执行轨迹是比较耗时的(耗时6330秒). 同时,测试用例设计的质量也会对方法性能产生影响,方法性能不太稳定. 相对而言,ICAN 属于静态分析方法,收集软件的静态结构信息相对容易(实验系统中整个方法运行仅48秒),方法性能也会比较稳定.

3.4.2 ICAN 方法的扩展性如何?

作为一个实用的方法,ICAN 方法会被应用于不同规模的系统,用于挖掘系统中的关键类. 本节中,我们将通过跟踪 ICAN 方法应用于实际系统时的时间开销来检验其扩展性.

如前文所述,ICAN 方法识别系统中的关键类主要包括3个步骤:(1)解析软件源码获取结构信息;(2)构建 WCCN;(3)应用 $W_{k\text{-core}}$ 方法计算 WCCN 中节点核数,并过滤得到候选关键类.

此部分我们新增了2个规模比较大的软件作为实验对象. 表5列出了这两个系统的一些统计数据.

表6列出了 ICAN 方法在各实验系统中每一个步骤的耗时. 从表6可见,第1步的耗时与系统的代码行存在正相关;第2步和第3步的耗时与网络的规模(节点数)存在正相关. 但是,我们发现:尽管 Vuze-4400 规模极大,具有7572个类,38908个方法和属性,但是挖掘其关键类所需时间不足6分钟. ICAN 方法具有较好的扩展性.

表5 新增实验系统的统计数据

项目	版本号	代码行	包数	类数	方法/属性数
Tomcat	8.5.4	294,949	207	3,347	51,651
Vuze	4400	478,316	473	7,572	38,908

表6 实验系统每一步骤的时间消耗

步骤	Ant (s)	JMeter (s)	Tomcat (s)	Vuze (s)
1	16	3	36	62
2	32	5	153	268
3	0	0	1	1

注:运行时间为0s,表示其运行时间不足0.5秒,并不是不需要时间

4 结论和下一步工作

本文从软件静态结构出发,提出了一种基于软件网络加权 k-核分析的关键类识别方法,实现从 Java 软件源代码中识别关键类,并用 Ant-1.6.1 和 JMeter-2.0.1 两个真实软件进行验证. 本文方法根据类(接口)间的7种耦合关系构建加权依赖图(加权类耦合网络),在此基础上使用加权 k-核分解方法计算类(接口)的加权核数并排序. 实验结果表明,我们的方法可以识别大部分的关键类,在检查前15%的类时,可以取得70%的召回率. 本文的方法可以帮助开发人员尽快熟悉一个陌生的软件系统.

未来我们将用更多来自不同领域、具有不同规模的软件验证本文方法的有效性；同时分析关键类与软件缺陷分布、可靠性、易变性等的关系，探索关键类识别方法在更多实际环境中的应用价值.

参考文献

[1] Boehm B. The changing nature of software evolution[J]. IEEE Software, 2010, 27(4): 26 – 29.

[2] D'Ambros M. Supporting software evolution analysis with historical dependencies and defect information[A]. IEEE International Conference on Software Maintenance[C]. Beijing, China: IEEE, 2008. 412 – 415.

[3] Yau S, Collofello J S. Design stability measures for software maintenance[J]. IEEE Transactions on Software Engineering, 1985, 11(9): 849 – 856.

[4] Yau S, Collofello J S. Some stability measures for software maintenance[J]. IEEE Transactions on Software Engineering, 1980, 6(6): 545 – 552.

[5] Zaidman A, Demeyer S. Automatic identification of key classes in a software system using webmining techniques [J]. Journal of Software Maintenance and Evolution: Research and Practice, 2008, 20(6): 387 – 417.

[6] 姜淑娟，鞠小林，王兴亚，等. 基于 UIO 序列的类重要性度量[J]. 电子学报，2015，43(10)：2062 – 2068.

Jiang S J, Ju X L, Wang X Y, et al. Measuring the importance of classes using UIO sequence[J]. Acta Electronica Sinica, 2015, 43(10): 2062 – 2068. (in Chinese)

[7] Inoue K, Yokomori R, Yamamoto T, et al. Ranking significance of software components based on use relations[J]. IEEE Transactions on Software Engineering, 2005, 31(3): 213 – 225.

[8] Zaidman A, Calders T, Demeyer S, et al. Applying webmining techniques to execution traces to support the program comprehension process[A]. European Conference on Software Maintenance and Reengineering[C]. Manchester, UK: IEEE, 2005. 134 – 142.

[9] 周毓明，徐宝文. 基于依赖结构分析的类重要性度量方法[J]. 东南大学学报(自然科学版)，2008，38(3)：380 – 384.

Zhou Y M, Xu B W. Dependence structure analysis-based approach for measuring importance of classes[J]. Journal of Southeast University (Natural Science Edition), 2008, 38(3): 380 – 384. (in Chinese)

[10] 王木生，卢红敏，周毓明，等. 利用 h 指数及其衍生度量识别关键类[J]. 计算机科学与探索，2011，5(10)：809 – 903.

[11] Pan W F, Li B, Ma Y T, et al. Multi-granularity evolution analysis of software using complex network theory[J]. Journal of Systems Science and Complexity, 2011, 24(6): 1068 – 1082.

[12] Briand L C, Daly J W, Wüst J K. A unified framework for coupling measurement in object-oriented systems[J]. IEEE Transactions on Software Engineering, 1999, 25(1): 91 – 121.

[13] Garas A, Schweitzer F, Havlin S. A k-shell decomposition method for weighted networks[J]. New Journal of Physics, 2012, 14(8): 83030 – 83043.

[14] 李辉，赵海，徐久强，等. 基于 k-核的大规模软件宏观拓扑结构层次性研究[J]. 电子学报，2010，38(11)：2635 – 2643.

Li H, Zhao H, Xu J Q, et al. Research on hierarchy of large-scale software macro topology based on k-core[J]. Acta Electronica Sinica, 2010, 38(11): 2635 – 2643. (in Chinese)

[15] Batagelj V, Zaversnik M. An O(m) algorithm for cores decomposition of networks[J]. Computer Science, 2003, 1(6): 34 – 37.

[16] Sedgewick R. Implementing quicksort programs[J]. Communications of the ACM, 1978, 21(10): 847 – 857.

[17] Shannon P, Markiel A, Ozier O, et al. Cytoscape website [EB/OL]. http://www.cytoscape.org, 2016-11-13.

[18] Costa L F, Rodrigues F A, Travieso G, et al. Characterization of complex networks: a survey of measurements[J]. Advances in Physics, 2007, 56(1): 167 – 242.

作者简介

潘伟丰　男，1982 年 12 月出生于浙江省杭州市，毕业于武汉大学获工学博士学位，现为浙江工商大学副教授，硕士生导师，主要研究方向为软件工程、服务计算、复杂网络和智能计算.

E-mail: wfpan1982@163.com

宋贝贝　女，1989 年 4 月出生于河南省太康县，现为浙江工商大学硕士研究生，主要研究方向为软件工程和服务计算.

E-mail: m13023651060@163.com