基于风险分析的回归测试用例优先级排序

于 海 杨 月 王 莹 张 伟 朱志良

(东北大学软件学院 沈阳 110169)

摘 要 该文利用软件组件间信息流的传递过程,提出了基于风险分析的回归测试用例优先级排序算法(Risk Analysis-based Test Case Prioritization,RA-TCP).该算法针对现有的优先级排序技术未能有效利用测试用例所覆盖信息的问题,在类粒度下将软件抽象为基于信息流的类级有向网络模型,然后将每个测试用例所覆盖的类间信息传递关系用一组杠铃模型表示,结合概率风险评估方法和故障树理论计算杠铃模型的风险值,最后以测试用例所覆盖的杠铃风险总和作为其排序依据.实验结果表明,风险越高的测试用例覆盖错误的可能性越大,RA-TCP 算法提高了具有严重风险的错误发现速率,与 7 种排序算法对比,RA-TCP 算法具有较高的错误检出率及较好的稳定性.

关键词 回归测试;测试用例优先级;复杂网络;风险分析;信息流;软件测试;软件工程

中图法分类号 TP311 **DOI 号** 10.11897/SP.J.1016.2019.02127

Risk Analysis-Based Test Case Prioritization for Regression Testing

YU Hai YANG Yue WANG Ying ZHANG Wei ZHU Zhi-Liang

(*Software College, Northeastern University, Shenyang* 110169)

Abstract The target of regression testing is to ensure that the modified program still satisfies its requirements. Due to the evolution of a software system, software engineers should re-run the existing test cases to detect whether new faults have been introduced into previously tested code. To improve the effectiveness of regression testing, researchers have studied various techniques including regression test selection, test suite minimization and test case prioritization. Test case prioritization techniques allow testers to schedule their test cases so that those test cases with the highest priority, according to some criterion, are executed earlier in the regression testing process than lower priority test cases. Increasing a test suite's rate of fault detection is considered as one potential goal of test case prioritization, which is a measurement of how quickly a test suite detects faults during the testing process. An improved rate of fault detection can provide earlier feedback on the system under regression test, and let developers begin debugging earlier than might otherwise be possible. In this study, we present a Risk Analysis-based Test Case Prioritization (RA-TCP) algorithm based on the transmission of information flows among software components. The existing works didn't effectively use the information covered by test cases to assign their priorities. To address this problem, the proposed algorithm firstly abstracts software as an information flow-based directed network model under classes. Then, class relationships covered by each test case are represented by a set of barbell motifs. On this basis, the execution probabilities,

收稿日期:2016-10-25;在线出版日期:2017-12-11.本课题得到国家自然科学基金(61374178,61402092,61603082)资助.**于 海**,副教授,硕士生导师,主要研究方向为复杂网络理论、混沌加密技术、软件测试、软件重构及软件体系结构.E-mail:yuhai@mail.neu.edu.cn.**杨 月**,硕士研究生,主要研究方向为软件测试及复杂网络理论.**王 莹**(通信作者),博士,主要研究方向为软件体系结构、复杂网络理论及软件测试.E-mail:wangying8052@163.com.**张 伟**,副教授,硕士生导师,主要研究方向为信号处理、多媒体信息编码技术及软件体系结构.**朱志良**,教授,博士生导师,中国计算机学会(CCF)会员,主要研究领域为混沌加密理论、软件测试、软件重构技术及复杂网络理论.

vulnerabilities and failure consequences of nodes and edges in the software network are calculated using risk analysis theory. Furthermore, to quantitatively analyze the risk coverage of test cases, we decompose the functional paths covered by test cases into a series of barbell motifs. With the aid of fault tree model, we provide the formal semantics of causal chain of the system failure. Finally, we assign a priority to each test case by calculating the sum of risk indexes of all the barbell motifs covered by it. Assigning higher priority to the critical test cases helps to detect the severer faults in the early testing steps, thereby the system reliability and test efficiency are improved. Four open source software are adopted as our experimental subjects to verify the validity of the proposed approach. The experimental results show that for these systems, the risk coverage distribution of test cases conforms to the Pareto principle, i. e. , 80 percent of critical codes in the system are covered by 20 percent of test cases. Comparing with the other state-of-the-art prioritization algorithms, RA-TCP technique has a higher error detection rate and performs stable across different systems. The prioritization techniques based on historical code change information are more appropriate to regression testing, while RA-TCP technique which based on the complexity metric can be applied to solve the non-regression testing problem. In addition, RA-TCP technique can maximize the system risk reduction rate during the testing process. This advantage will be more obvious if the test case suite covers more high-risk barbell motifs of the system.

Keywords regression testing; test case prioritization; complex networks; risk analysis; information flow; software testing; software engineering

1 引 言

回归测试是保证软件质量的重要方式,其目的是确保修改后的程序仍能满足用户的需求,即验证程序的正确性[1].回归测试在整个软件测试周期中具有重要的作用,不仅频繁被执行,所花费的测试成本也极其昂贵,占软件维护成本的近 50%[2].因此,为了在此阶段使软件的变更与计算达到尽可能高效,降低软件的运行周期和成本,测试人员希望可以对测试用例按某种方式进行排序,使优先级更高的测试用例可以更早的运行,即测试用例优先级排序技术[3-4].

测试用例优先级排序可以显著提高测试套件的故障检测率,通过不同的测试用例优先级排序技术以达到成本与效益需求间的平衡[5].这种排序技术能将特定测试目标最大化,通过重新排列测试用例的执行顺序最大程度地提高故障检测能力,是降低回归测试成本的一种有效方法[6].

近年来,研究人员提出了很多经典的测试用例优先级算法来解决测试用例的排序问题.其中,基于代码的排序技术在处理更大更复杂的软件时难以管理从代码中获取的信息,具有一定局限性.基于软件结构复杂性信息的排序技术是根据拓扑结构复杂性将软件网络中的类节点赋予测试重要性,并对每个测试用例所覆盖的类测试重要性求和,以此作为优先级排序依据,但其没有考虑测试用例所覆盖的类与类之间存在的信息传递关系.因此,提出一种回归测试过程更高效、覆盖信息更全面的测试用例排序算法具有重要的研究价值.

我们发现有研究人员将风险分析方法应用于软件工程领域,以度量由于软件模块失效而给系统造成损失的可能性和量级.Yacoub 等人[7]基于构件依赖图提出了一种体系结构层级的分析方法可以在软件开发初期对系统可靠性风险进行评估.Hettiarachchi 等人[8]利用需求的修改状态和复杂性等信息评估软件的系统风险,从而获取高风险的系统组件信息.Cui 等人[9]根据软件网络的涟漪效应分析面向对象软件的结构风险,并提出评价节点涟漪程度的度量指标,用以识别软件的脆弱节点及潜在风险节点,为软件设计和维护提供指导.研究表明,风险评估方法是分析系统可靠性的有效途径之一,即综合考虑软件系统中所有组件的情况,进而计算出所有事件发生的可能性及其后果.因此,有必要

将风险分析模型应用到测试用例优先级排序技术当中.在资源受约束的情况下,优先执行代码风险覆盖率较高的用例,以提高测试效率.

本文综合考虑了软件组件的动态执行概率、故障率以及失效后果等多种因素,提出了一种基于风险分析的测试用例优先级排序技术(Risk Analysis-based Test Case Prioritization,RA-TCP).将系统运行过程中,类间的动态功能调用等价于信息流的传递,从而将软件抽象为基于信息流的类级有向网络.结合复杂网络理论[10],将测试用例所覆盖的功能路径分解为一系列以类为节点,类间信息传递关系为边的杠铃.利用故障树定量分析杠铃模型可能失效的所有状态事件,度量出测试用例覆盖风险的概率,进而对测试用例进行排序.通过实例验证和与其他算法的对比分析,证明了 RA-TCP 算法的有效性、稳定性以及其排序效果与风险各项指标的相关性.

2　相关研究

2.1　测试用例优先级技术

回归测试优先级技术根据某些准则重新排列测试用例的执行顺序,即为满足特定目标而指定测试用例的优先级[11].Rothermel 等人[12]将测试用例优先级问题抽象为寻找测试用例全排列中的最优集合问题:给定测试用例集 T,T 的全排列集合 PT,映射 PT 到实数的函数 f,求一个 $T'\in PT$,使得 $(\forall T'')(T''\in PT)(T''\neq T')[f(T')\geq f(T'')]$.其中,$f$ 表示测试结果的目标函数.

测试用例优先级技术使用覆盖信息作为目标,关注于尽可能早地最大化某种程序元素的覆盖率,例如语句、分支、数据流、历史变更信息等[13].Elbaum 等人[3,14]研究的测试用例优先级排序方法,将语句覆盖优先级、功能覆盖优先级和错误索引空间优先级考虑到测试用例排序中.Jeffrey 等人[15]基于测试用例输出相关切片的覆盖情况提出了一种新的测试用例优先级排序技术.Mei 等人[16]通过静态分析的方式对 JUnit 测试用例和被测程序进行分析,构建出函数调用图,再对测试用例的代码覆盖能力进行测试用例优先级排序.Yoo 等人[17]通过对专家知识和代码覆盖率的分析,提出基于聚类的测试用例优先级排序技术.Pan 等人[18]将复杂网络理论引入回归测试中,通过将软件系统抽象为加权类依赖网络的方式计算类错误产生的可能性和严重性,从而获取类的测试重要性,再结合测试用例的历史信息对测试用例赋予优先级.

上述针对测试用例的优先级排序算法都是基于代码或软件结构复杂性信息的测试重要性进行研究,而在覆盖信息重要性测度当中,并未结合面向对象软件系统的拓扑结构特征考虑其可靠性风险的权重.由此,本文以测试用例对系统风险的覆盖率作为排序依据,力求以最高的效率来改善系统可靠性.

2.2　风险分析在软件测试中的应用

通常,在软件发布之前进行充分的测试是系统可靠性和健壮性的重要保证.测试过程中,引入一些从测试中间结果或者程序结构中发现的潜在风险知识作为辅助措施,将有助于进行故障定位,提高测试效率[19].

鉴于此,Felderer 等人[20]设计出一种通用的算法框架可将风险分析集成于测试过程,用以优化测试资源分配.文中,他们系统地讨论了基于风险的软件测试的潜在收益,前提条件以及面临的挑战.Schneidewind 等人[21]利用假设的缺陷及故障分布情况,根据软件生产者与消费者之间的关系,开发出针对于不同测试场景的风险驱动的测试模板.最终通过在真实软件上的实验分析,论证了其理论的有效性.Amland[22]通过使用两个相互关联的因素来计算风险值,将风险因子定义为故障率与故障代价的乘积.并将风险因子作为关键模块的度量标准,进而提出了基于风险分析的测试方法.Chen 等人[23]对如何处理潜在问题区域的课题进行研究,提出了一种基于风险分析的面向对象软件回归测试选择技术.Zimmermann 等人[24]针对安全苛求软件系统提出了一种基于风险的测试用例自动化生成方案.该模型结合统计测试与马尔可夫链模型来描述系统的使用剖面,最终只生成覆盖风险程度较高的关键测试用例.

尽管基于风险分析的软件测试研究已经取得了一定进展,然而很少有文献从软件的网络观点探讨系统组件的脆弱性、风险以及故障后果的定义,并利用其解决回归测试优先级问题.这就构成了本研究的初衷,将金融领域的风险标准定义延伸到面向对象软件系统的可靠性分析当中,以此解决实际的工程问题.

2.3　面向对象的软件网络模型

复杂系统和复杂网络的研究强调从整体上把握系统,通过将软件抽象为节点和边的网络形式,为

理解软件系统提供了有价值的视角和不同的分析维度[25].

2002 年 Valverde 等人[26]首先将复杂网络理论引入到软件系统拓扑结构分析中.通过逆向工程的方法将面向对象软件的类图作为研究对象,用无向图来表示软件系统,即模型中的节点代表类,边表示类之间的交互关系,如继承和关联关系等.随后,Moura 等人[27],Wheeldon 等人[28],Valverde 等人[29]和 Myers[30]使用有向图来描述软件系统的结构,将模块或函数抽象为节点,它们之间的调用关系抽象为有向边,并定义这种描述软件系统结构的网络为软件协作图或软件依赖网络,同时认为软件系统实质上代表了一种人工复杂网络.

软件系统的复杂网络研究可以用形式化的方法来描述软件系统的性质,具有良好的数据基础,更容易洞察系统整体的特征和规律,是刻画软件复杂性的有力工具[31].Wang 等人[32]将该理论应用于面向对象软件集成测试序列的生成算法中,结合类级有向网络和 Class-HITS 算法找到复杂程度高、传播范围较大的节点,并将其视为测试重点.兼顾测试效率和代价,保证在集成过程中重要节点具有较高的优先级,同时确保构造的测试桩总复杂度较小.Pan 等人[33]通过分析错误在软件网络上的传播行为,建立错误传播网络,根据获取的类测试重要性来计算测试用例的优先级,进而对测试用例排序.然而上述软件网络模型均忽略了系统模块间信息流量的传递过程,亦忽视了每一段功能路径失效时节点和链路的组合故障模式.如何结合网络的拓扑结构分析面向对象软件的风险因素,并将其应用到回归测试领域,依然是一个值得探讨的问题.

3 基于信息流的类级有向网络模型

本文通过构建基于信息流的类级有向网络模型(Class-level Directed Network Model based on Information Flow,CDNMIF)来分析与描述面向对象软件系统的拓扑结构.

设$\mathcal{S}$为任意面向对象软件系统,C_i表示系统$\mathcal{S}$中的任意一个类,m_{ip},a_{iq}分别是类C_i中的任意方法和属性,$i\in\{1,2,\cdots,NC\}$,则我们有$\mathcal{S}=\{C_1,C_2,\cdots,C_{NC}\}$,$C_i=\{m_{ip}\mid p\in\{1,2,\cdots,NM_i\}\}\cup\{a_{iq}\mid q\in\{1,2,\cdots,NA_i\}\}=\{m_{i1},m_{i2},\cdots,m_{iNM_i}\}\cup\{a_{i1},a_{i2},\cdots,a_{iNA_i}\}$,其中,$NC$表示系统中类的总数,$NM_i$和$NA_i$分别表示类$C_i$中的方法和属性总数.

定义 1. 类间有向信息传递关系.对于系统的任意方法$m_{it}\in C_i$,$m_{jk}\in C_j$,$i\neq j$,其信息流传递关系$m_{it}\rightarrow m_{jk}$应满足下述 3 个条件之一:

(1)m_{it}调用了m_{jk},且向m_{jk}传递了参数;

(2)m_{jk}调用了m_{it},且m_{it}传递给m_{jk}一个返回值;

(3)m_{jt}使用了全局变量a_p,同时调用了m_{ik},且m_{ik}在函数体内更新了全局变量a_p.

定义 2. 信息流量.信息流的传递是系统中任一方法通过调用或被调用的方式直接或间接地将参数信息传递给另一个方法的过程.若将类节点C_p指向C_q的边表示为$\langle C_p,C_q\rangle$,$p\neq q$,则C_p传递给C_q的信息流量为

$$L(C_p,C_q)=\sum_{m_{pi}\in C_p}\sum_{m_{qj}\in C_q}N_{\text{call}}(m_{pi},m_{qj})\times Times_{ij}\quad(1)$$

其中,$L(C_p,C_q)$表示类C_p传递给类C_q的总流量,即边$\langle C_p,C_q\rangle$上的权值;$N_{\text{call}}(m_{pi},m_{qj})$表示方法$m_{pi}\in C_p$单次传递给方法$m_{qj}\in C_q$的流量,即传递的参数个数;$Times_{ij}$表示系统中方法$m_{pi}$对$m_{qj}$信息传递的次数.由此可得系统中类间传递关系的信息总流量为

$$TF=\sum_{C_i\in\mathcal{S}}\sum_{C_j\in\mathcal{S}}L(C_i,C_j)\quad(2)$$

定义 3. 基于信息流的类级有向网络(CDNMIF)模型.以类为粒度,CDNMIF 表示类与类之间信息传递关系的有向网络图,即$G=(V,E,\boldsymbol{Z})$.其中,$V=\{C_i\mid i\in\{1,2,\cdots,NC\}\}$为系统中类节点的集合;$E=\{\langle C_i,C_j\rangle\mid i,j\in\{1,2,\cdots,NC\},i\neq j\}$为类间信息传递的有向关系集合.若$m_{pi}\in C_p$,$m_{qj}\in C_q$,$p\neq q$,当且仅当$m_{pi}$,$m_{qj}$之间存在信息传递关系$m_{pi}\rightarrow m_{qj}$时,网络$G$中存在一条有向边$\langle C_p,C_q\rangle\in E$.$\boldsymbol{Z}=(z_{ij})_{NC\times NC}$表示为有向图$G$中节点间信息传递关系的权值矩阵,$z_{ij}=L(C_p,C_q)$.

图 1 和图 2 分别给出了一个代码片段和相应的类间信息传递关系示例图.如图所示,*Cc.b*()调用*Ce.c*(*x*)时向其传递了参数*x*,即*Cc.b*()→*Ce.c*(*x*);*Ca.b*()调用*Cb.c*()后返回给*Ca.b*()一个*return*值,即*Cb.c*()→*Ca.b*();*Cd.c*()使用了全局变量*b*的同时调用了*Ce.a*(*x*),且*b*在*Ce.a*(*x*)的作用域被更新,即*Ce.a*(*x*)向*Cd.c*()反馈了变量*b*的更新情况.因此,我们有*Cd.c*()→*Ce.a*(*x*)、*Ce.a*(*x*)→*Cd.c*().图 3 为示例代码抽象成的 CDNMIF 模型,其中类为节点,边为两个类之间的信息传递关系,类间传递的信息流量即为边的权值.

```
public Cass Ca {
    public int a;
    public void b() {
        Cb b = new Cb();
        a = b.c();
    }
    public void c() {
        Cc c = new Cc();
        char[] m = c.b();
    }
}

public Cass Cb {
    private int a(int x, int y) {
        int m = x×x−y×y;
        return m;
    }
    private int b(int y) {
        int m = y×y;
        return m;
    }
    public int c() {
        int x;
        int y;
        int z = a(x, y) + b(y);
        return z;
    }
}
```

```
public Cass Cc {
    public String a;
    public char[] b() {
        char[] str = a.toCcharCarray();
        int x = a.length();
        Ce e = new Ce();
        e.c(x);
        return str;
    }
    public int c(int x) {
        Cd d = new Cd();
        int m = x−d.d(x×x);
        return m;
    }
}

public Cass Cd {
    public int a;
    public boolean b;
    public void c(int x,int y) {
        d(x+y);
        b = true;
        Ce e = new Ce();
        e.a(a);
    }
    public int d(int y) {
        return y×a;
    }
}
```

```
public Cass Ce {
    private int d;
    public void a(int x) {
        Cd dd = new Cd();
        if (x != d) {
            dd.b = false;
        }
        Cc c = new Cc();
        dd.c(x, d);
        c.c(x);
    }
    public void b() {
        Cc c = new Cc();
        int n = c.c(d);
        c(n);
    }
    public void c(int x) {
        System.out.println(x);
    }
}
```

图 1　示例代码片段

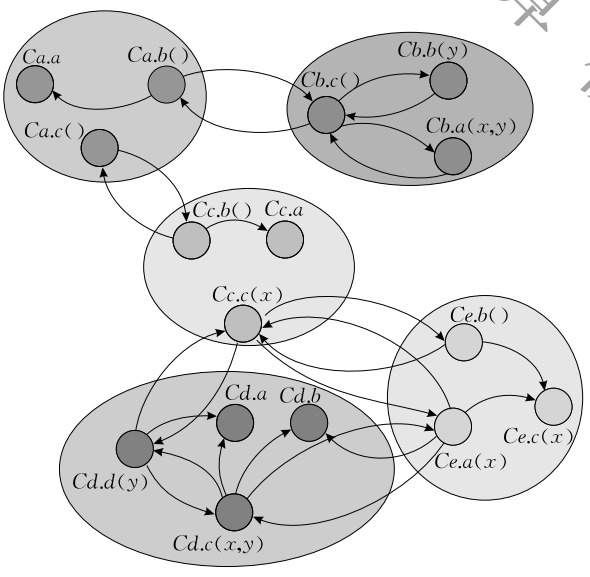


图 2　类间信息传递关系图

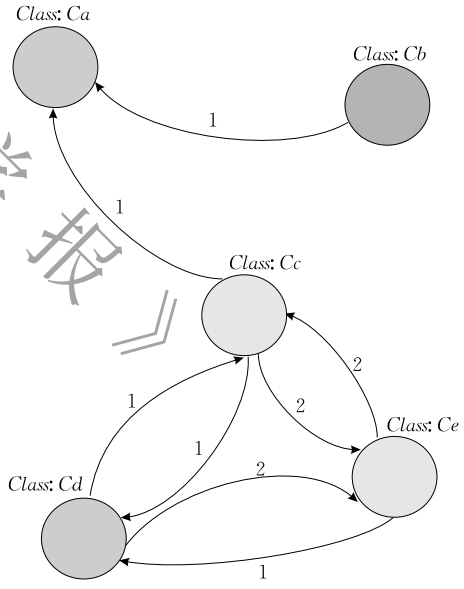


图 3　实例代码对应的 CDNMIF 模型

从 CDNMIF 的定义可以看出，组件之间的依赖关系可视为动态运行时信息流传递的管道，网络中某个节点的失效行为可直观地理解为由于管道阻塞而引起信息流动停止的现象. 因此，任意组件的风险因子都会对整个系统的可靠性造成影响，借助该模型可进一步分析组件在系统中逻辑位置的重要性.

4　RA-TCP 方法

4.1　概率风险评估模型

我国在 1990 年制定的 GJB900《系统安全性通用大纲》中对系统的风险有明确性定义：危险事件的

风险就是该事件发生的概率和损失程度的函数[34]. NASA-STD-8719.13A[35]为软件系统定义了多种风险类型:性能风险、容忍度风险、实用性风险、代价风险等等.而本文关注于软件的可靠性风险分析.软件可靠性,是指在规定时间内及规定的环境下,软件不引起系统发生失效的概率[36].综合上述定义,概率风险评估模型(Probabilistic Risk Analysis,PRA)可作为解决软件可靠性风险评估问题的有力手段.

在金融和工程领域,概率风险评估是一种辨识和评价复杂工程系统风险的结构化、集成化的逻辑分析方法,可被用于预测系统中的某个部件或者资产的启发式风险因子[37].利用 PRA 模型可以更精确地了解复杂系统的特性,找到其容易发生错误的位置,以此方式对系统的安全性进行预警.PRA 模型包括威胁性、脆弱性和后果三种评估指标,其风险的标准定义如下[38]:

$$\mathcal{R}=\mathcal{T}\times\mathcal{V}\times\mathcal{C} \tag{3}$$

其中,$\mathcal{T}$ 代表威胁性,表示尝试攻击的概率;$\mathcal{V}$ 代表脆弱性,表示攻击成功的概率;$\mathcal{C}$ 代表后果,表示通过成功的攻击所造成的后果及损失.

本文将 PRA 模型扩展到软件系统的风险分析中,用于识别系统中所存在的潜在风险.由软件错误运行或中止运行的事件序列、相应失效行为的发生概率及其对应的失效后果共同决定了某一组件的风险值.在研究测试用例优先级排序问题时,每一个测试用例都会覆盖多个类,而每个类都是由多个方法组成的.软件网络中,任何一个节点存在缺陷都会引起网络中不同程度的故障"涟漪效应".换言之,由于节点沿着链路将信息传递给与其有依赖关系的节点,进而造成网络中一系列的连锁故障.因此,根据每个组件的逻辑位置对系统总体风险的影响程度,将任意组件的标准风险$\mathcal{R}_i$的参数重新定义为

$\mathcal{T}_i$=威胁性,即组件内代码被执行的概率;

$\mathcal{V}_i$=脆弱性,即组件存在缺陷的概率;

$\mathcal{C}_i$=后果,即当组件存在缺陷时,给软件系统带来的预期损失.

结合 CDNMIF 模型,我们可以分别估算网络中节点和边的威胁性、脆弱性及后果值.

4.1.1 威胁性$\mathcal{T}$

假设所有的源代码均会被执行,而在实际功能操作过程中,用户的行为具有一定随机性,不同场景下,由于系统中的核心功能模块处于软件拓扑结构中的枢纽位置,因而被执行的概率较高.这里采用随机游走的方式对软件的动态运行状态进行模拟,找出模块之间为协作实现某个功能而传递信息的过程.建模过程遵循以下两个原则:

(1)增长特性:随着用户的操作,网络中参与功能执行的节点和边的规模不断扩大.

(2)优先执行特性:用户更倾向于执行系统中的核心功能模块.由于介数中心性较高的节点意味着系统中任意两个节点间信息传递的路径经过该节点的概率较高,即介数中心性越大的节点,其逻辑位置越重要[39].因此,当前功能执行节点的所有邻居节点中作为下一步功能跳转对象的概率满足:

$$\Pi(C_i)=\frac{BC_i}{\sum_{j=1}^{NG}BC_j},\ BC_i=\sum_{s\neq i\neq t}\frac{n_{st}^i}{g_{st}} \tag{4}$$

其中,NG 表示当前功能执行节点所有邻居节点的个数,BC_i表示当前执行类节点的邻居节点 C_i的介数中心性,g_{st}表示从节点 C_s到节点 C_t的最短路径数目,n_{st}^i表示从节点 C_s到节点 C_t的 g_{st}条最短路径中经过节点 C_i的路径数目.

为了保证用户操作的随机性、多样性以及收集执行路径的完整性,设计出如下功能游走过程:首先,随机选取任意功能的起始节点,即出度不为零且入度为零的类节点,视为用户操作的种子节点.由于功能的执行为自顶向下的过程,因此,沿着静态系统结构中类间的信息传递关系,以介数中心性优先为原则选取当前执行节点的邻居节点作为功能下一步跳转的路径.确定种子节点后,记录下种子节点通过链路随机游走至其他节点的路径,重复上述操作直至所有节点和边均被访问过且重复执行操作至少达到 LT 次时结束.如式(5)和(6)所示,可以求出任意类节点 C_i及边$\langle C_i,C_j\rangle$的威胁性,即为访问过该节点和边次数 TC_i和 TE_{ij}分别与所有节点和边被访问的总次数的比值.算法 1 对 CDNMIF 中所有类节点和边的威胁性计算过程进行了描述,时间复杂度为 $O(|E|)$,其中,$|E|$表示边的总数.这里,LT 取值 2000.

$$\mathcal{T}(C_i)=\frac{TC_i}{\sum_{j=1}^{NC}TC_j} \tag{5}$$

$$\mathcal{T}(C_i,C_j)=\frac{TE_{ij}}{\sum_{p=0}^{NC}\sum_{q=1}^{NC}TE_{pq}} \tag{6}$$

算法 1. 类节点和边的威胁性计算.

输入:CDNMIF:基于信息流的类级有向网络;

LT:运行次数;

t:当前运行次数

输出:$\mathcal{T}n$:所有节点的威胁性;

$\mathcal{T}e$:所有边的威胁性

1. 将网络中表示节点和边是否被访问过的标识状态

markN、*markE* 数组初始化为 FALSE；
2. 将存储节点和边被访问次数的数组 *timeN*、*timeE* 初始化；
3. *source*←入度为 0 且出度不为 0 的节点数组；
4. *goal*←出度为 0 且入度不为 0 的节点数组 *l*；
5. *t*←0；
6. WHILE(*t*<*LT*‖*markN*[*j*]∈*markN*!=TRUE‖*markE*[*j*]∈*markE*!=TRUE)
7. 执行式(4)获取节点 $S_i \in source$ 为种子节点；
8. 将从节点 S_i 随机游走至目标节点 $G_i \in goal$ 的节点路径存储到 *road* 队列；
9. *node*←*road* 队列中的节点集合；
edge←*road* 队列中的边集合；
10. FOR Each $ni \in node$ Do
11. *markN.ni*←TRUE; *timeN.ni*++；
12. END FOR
13. FOR Each $e_i \in edge$ Do
14. *markE.ei*←TRUE; *timeN.ei*++；
15. END FOR
16. *t*++；
17. END WHILE
18. FOR Each $i \in timeN$ Do
19. $\mathcal{T}ni = timeN[i]/sumN$; //*sumN*:数组 *timeN* 内的元素和
20. END FOR
21. FOR Each $i \in timeE$ Do
22. $\mathcal{T}ei = timeE[i]/sumE$; //*sumE*:数组 *timeE* 内的元素和
23. END FOR

4.1.2　脆弱性 $\mathcal{V}$

复杂系统科学的观点认为：网络拓扑结构的复杂程度决定了软件的脆弱性，即拓扑结构越复杂，其发生故障的可能性越大[30]. 本文利用 Henry 和 Kafura 提出用扇入和扇出的概念来衡量系统的复杂性[40]. 令 $leth_{it}$ 表示方法 m_{it} 所包含的代码行数；Fin_{it} 表示所有传入方法 m_{it} 中的参数个数与所有被方法 m_{it} 接收到的返回值个数之和；$Fout_{it}$ 表示所有方法 m_{it} 中传出的参数个数之和. 则网络中节点和边的脆弱性公式可分别表示为[40]

$$\mathcal{V}(C_i)=\sum_{m_{it}\in C_i} leth(m_{it})\times(Fin(m_{it})\times Fout(m_{it}))^2 \quad (7)$$

$$\mathcal{V}(C_p,C_q)=\sum_{m_{it}\in Mt_{pq}} leth(m_{it})\times(Fin(m_{it})\times Fout(m_{it}))^2 \quad (8)$$

其中，Mt_{pq} 为类 C_p 和 C_q 之间真正发生信息传递关系的方法集合，即将 CDNMIF 模型中连接任意类对 C_p 和 C_q 之间的边 $\langle C_p,C_q\rangle \in E$ 的脆弱性视为该类对之间传递信息的所有方法复杂度之和. 节点类 $C_i \in \mathcal{S}$ 的脆弱性等价于该类内部定义的所有方法的复杂度之和. 计算系统中所有类的脆弱性指标的时间复杂度为 $O(NM)$，这里，NM 为系统方法总数.

4.1.3　后果 $\mathcal{C}$

软件错误，又称故障，是系统在运行时由于缺陷造成的非正常状态的表现和反映[41]. 方法被执行时，在一定条件下偏离其预期设计的要求或规定的功能，我们定义这种现象为方法失效. 软件错误种类各异，对系统的破坏程度也不尽相同. 例如内存非法访问等错误能够导致软件系统的崩溃，但是大部分错误只是造成系统输出不满足用户的要求. NASA-STD-8719.13A 标准[35]指出，风险分析应当考虑任务失效发生时给系统带来损失的最坏情况. 因此，针对软件网络的“涟漪效应”，我们认为系统中任意方法 $m_{it} \in C_i$ 发生失效，错误都会以 100% 的概率沿着信息传递关系传播给直接或间接依赖它的所有方法，从而导致网络信息流的阻塞. 为简化问题研究，本文采用潘伟丰等人在文献[42]中的假设情况：(1)所有的错误仅仅会对软件的运行正确性造成影响；(2)错误仅仅会在信息传递过程中传播.

根据上述讨论，可将系统中由于方法 m_{it} 的失效引起的最大流量损失视为 m_{it} 的失效后果 $\mathcal{C}(m_{it})$，

$$\mathcal{C}(m_{it})=TF-\sum_{m_j\in \overline{M_t'}}\sum_{m_k\in \overline{M_t'}} f(m_j,m_k)\times Times_{jk} \quad (9)$$

$$\overline{M_t'}=M\backslash M_t' \quad (10)$$

其中，M 表示系统中所有方法的集合，M_t' 表示在静态系统中所有直接或间接可达方法 m_{it} 的方法集合，$\overline{M_t'}$ 表示集合 M 与 M_t' 的差集，即在方法 m_{it} 失效的情况下系统依然正常运转的方法集合；TF 如式(2)所示，表示系统正常工作状态下信息传递的总流量；$f(m_j,m_k)$ 表示方法 $m_j \in \overline{M_t'}$ 单次传递给方法 $m_k \in \overline{M_t'}$ 的信息流量，即参数个数. 接下来，结合算法 1 中模拟动态软件运行的信息，估算方法之间的通信频率：

(1)若方法 m_j 与 m_k 属于同一个类 C_p，则 m_j 对 m_k 信息传递的次数 $Time_{jk}$ 为算法 1 所述的随机游走过程中类 C_p 被访问到的次数 TC_p；

(2)若方法 $m_j \in C_p$，$m_k \in C_p$，$p \neq q$，则方法 m_j 对 m_k 信息传递的次数 $Time_{jk}$ 为模拟系统动态运行过程中边 $\langle C_p,C_q\rangle$ 被触发的次数 TE_{pq}.

那么类节点 C_i 内部定义的所有方法失效后果的平均值可用于刻画该类的失效后果：

$$\mathcal{C}(C_i)=\frac{\sum_{m_{it}\in C_i}\mathcal{C}(m_{it})}{NM_i} \quad (11)$$

进而，类间传递关系 $C_p \rightarrow C_q$ 的失效后果为类 C_p 和 C_q 间传递信息的所有方法失效后果之和：

$$\mathcal{C}(C_p,C_q)=\frac{\sum_{m_t \in Mt_{pq}} \mathcal{C}(m_t)}{|Mt_{pq}|} \tag{12}$$

通过结合高效回溯机制对网络中的节点进行深度遍历，计算所有类的失效后果的时间复杂度为 $O(|V|^2)$，即类节点总数的平方.

4.2 测试用例优先级计算

定义 4. 杠铃模型. 在软件运行过程中，每一步信息传递过程都由源点、信息管道和接收终点协作完成，这三个基本元素构成了“节点-链路-节点”的杠铃模型.

系统发生失效的根本原因可归结于某一步信息传递的杠铃模型发生失效. 而杠铃模型的失效，对应着 $2^3-1=7$ 种可能的状态事件：(1) $Event$(源点 F，链路 N，接收点 N)；(2) $Event$(源点 N，链路 F，接收点 N)；(3) $Event$(源点 N，链路 N，接收点 F)；(4) $Event$(源点 F，链路 F，接收点 N)；(5) $Event$(源点 F，链路 N，接收点 F)；(6) $Event$(源点 N，链路 F，接收点 F)；(7) $Event$(源点 F，链路 F，接收点 F). 其中，N 表示组件元素工作正常，F 表示组件元素发生失效. 从程序代码的角度，这里的节点、链路的失效事件均表示其输出结果偏离了功能的预期要求.

为考虑所有导致系统失效的状态事件，定量分析测试用例所覆盖的风险因子信息，本文借助于杠铃模型，将每个测试用例所覆盖的功能路径分解成若干“节点-链路-节点”的形式，即将功能路径精化为一步步的信息传递操作，进而可利用状态事件故障树的形式语义描述复杂系统的失效因果链. 以图 3 所示的网络模型为例，若一个测试用例覆盖的类信息为类 Ca、Cb、Cd、Ce，则图 4 即为这个测试用例所覆盖的杠铃模型，边权值为源点与接收节点之间传递的信息量.

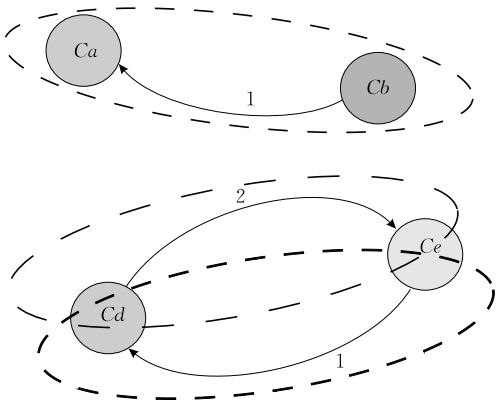


图 4 杠铃模型示例

故障树(FT)等同于事件树(ET)，是一种特殊的倒立树状因果关系图，它用事件符号、逻辑门符号和转移符号描述系统中各种事件之间的因果关系，是概率风险评估中的核心方法[43]. 逻辑门的输入事件是输出事件的“因”，逻辑门的输出事件是输入事件的“果”. 由此，通过故障树分析即可获得一个杠铃模型在不同条件下所有状态的事件集合. 杠铃发生失效的可能性组合方式如图 5 所示，其显示的是一个杠铃中的节点与链路均被执行条件下杠铃发生故障的事件树，OR 逻辑门意味着如果被其连接的组件 A 或 B 或者都失效，错误便会沿着故障树传播.

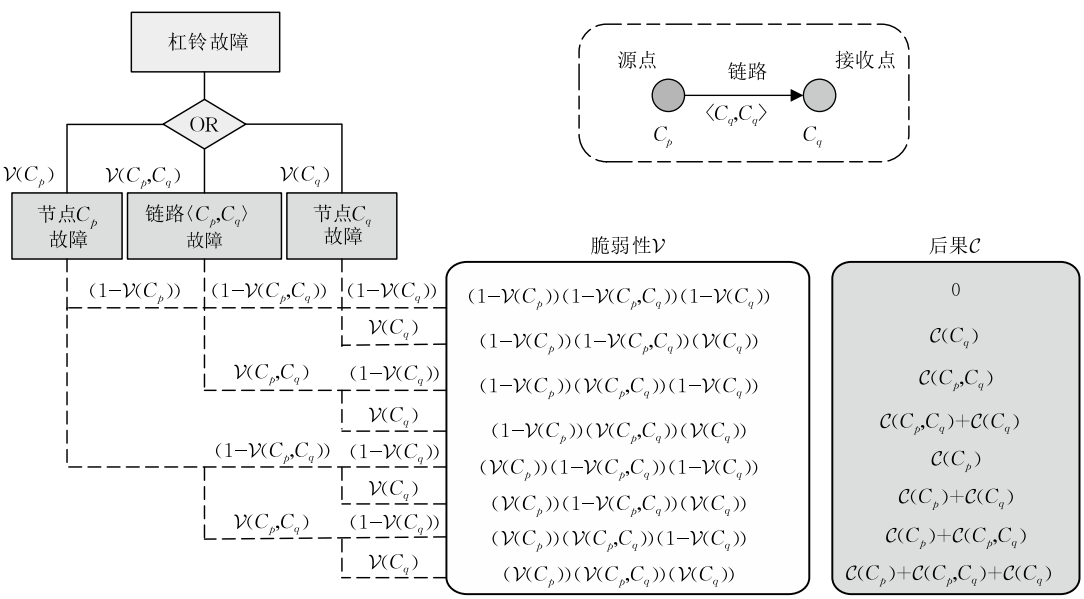


图 5 杠铃中的节点与链路均被执行条件下的事件树($\mathcal{T}(C_p)\cdot\mathcal{T}(\langle C_p,C_q\rangle)\cdot\mathcal{T}(C_q)$)

图 5 中，$\mathcal{V}(C_p)$，$\mathcal{V}(\langle C_p,C_q\rangle)$及$\mathcal{V}(C_q)$分别表示节点 C_p，链路$\langle C_p,C_q\rangle$或节点 C_q存在缺陷的概率；$\mathcal{T}(C_p)$，$\mathcal{T}(\langle C_p,C_q\rangle)$及$\mathcal{T}(C_q)$分别表示节点 C_p，链路$\langle C_p,C_q\rangle$或节点 C_q被执行的概率；$\mathcal{C}(C_p)$，$\mathcal{C}(\langle C_p,C_q\rangle)$及$\mathcal{C}(C_q)$分别表示节点 C_p，链路$\langle C_p,C_q\rangle$或节点 C_q存在缺陷时造成的后果. 假设节点与链路均被执行，即威胁性为$\mathcal{T}(C_p)\cdot\mathcal{T}(\langle C_p,C_q\rangle)\cdot\mathcal{T}(C_q)$，当节点 C_p没有缺陷、链路$\langle C_p,C_q\rangle$没有缺陷、节点 C_q存在缺陷的情况下其脆弱性为事件树中对应事件的概率乘积，即$(1-V(C_q))(1-V(\langle C_p,C_q\rangle))\,V(C_q)$，后果为$\mathcal{C}(C_q)$. 若用“+”表示组件不存在缺陷，“−”表示组件存在缺陷，则将上述组合事件应用到概率风险评估模型中可得到其风险值：

$$\begin{aligned}&Risk(C_p^+,\langle C_p,C_q\rangle^+,C_q^-)=\\&(1-\mathcal{V}(C_p))\cdot(1-\mathcal{V}(\langle C_p,C_q\rangle))\cdot\\&\mathcal{V}(C_q)\cdot\mathcal{T}(C_p)\cdot\mathcal{T}(\langle C_p,C_q\rangle)\cdot\mathcal{T}(C_q)\cdot\mathcal{C}(C_q)\end{aligned}$$

考虑到图 5 中所有可能的事件组合，我们将概率风险评估模型应用到上述组合事件，即第 i 个杠铃中威胁性为$\mathcal{T}(C_p)\cdot\mathcal{T}(\langle C_p,C_q\rangle)\cdot\mathcal{T}(C_q)$的情况下的风险值为

$$\begin{aligned}\mathcal{R}'_i=&\mathcal{T}(C_p)\cdot\mathcal{T}(\langle C_p,C_q\rangle)\cdot\mathcal{T}(C_q)\cdot(1-\mathcal{V}(C_p))\cdot\\&(1-\mathcal{V}(\langle C_p,C_q\rangle))\cdot\mathcal{V}(C_q)\cdot\mathcal{C}(C_q)+\mathcal{T}(C_p)\cdot\\&\mathcal{T}(\langle C_p,C_q\rangle)\cdot\mathcal{T}(C_q)\cdot(1-\mathcal{V}(C_p))\cdot\mathcal{V}(\langle C_p,C_q\rangle)\cdot\\&(1-\mathcal{V}(C_q))\cdot\mathcal{C}(\langle C_p,C_q\rangle)+\mathcal{T}(C_p)\cdot\mathcal{T}(\langle C_p,C_q\rangle)\cdot\\&\mathcal{T}(C_q)\cdot(1-\mathcal{V}(C_p))\cdot\mathcal{V}(\langle C_p,C_q\rangle)\cdot\mathcal{V}(C_q)\cdot\\&(\mathcal{C}(\langle C_p,C_q\rangle)+\mathcal{C}(C_q))+\mathcal{T}(C_p)\cdot\mathcal{T}(\langle C_p,C_q\rangle)\cdot\\&\mathcal{T}(C_q)\cdot\mathcal{V}(C_p)\cdot(1-\mathcal{V}(\langle C_p,C_q\rangle))\cdot(1-\mathcal{V}(C_q))\cdot\\&\mathcal{C}(C_p)+\mathcal{T}(C_p)\cdot\mathcal{T}(\langle C_p,C_q\rangle)\cdot\mathcal{T}(C_q)\cdot\mathcal{V}(C_p)\cdot\\&(1-\mathcal{V}(\langle C_p,C_q\rangle))\cdot\mathcal{V}(C_q)\cdot(\mathcal{C}(C_p)+\mathcal{C}(C_q))+\\&\mathcal{T}(C_p)\cdot\mathcal{T}(\langle C_p,C_q\rangle)\cdot\mathcal{T}(C_q)\cdot\mathcal{V}(C_p)\cdot\mathcal{V}(\langle C_p,C_q\rangle)\cdot\\&(1-\mathcal{V}(C_q))\cdot(\mathcal{C}(C_p)+\mathcal{C}(\langle C_p,C_q\rangle))+\mathcal{T}(C_p)\cdot\\&\mathcal{T}(\langle C_p,C_q\rangle)\cdot\mathcal{T}(C_q)\cdot\mathcal{V}(C_p)\cdot\mathcal{V}(\langle C_p,C_q\rangle)\cdot\\&\mathcal{V}(C_q)\cdot(\mathcal{C}(C_p)+\mathcal{C}(\langle C_p,C_q\rangle)+\mathcal{C}(C_p)),\end{aligned}$$

整理后可得

$$\begin{aligned}\mathcal{R}'_i=&\{\mathcal{V}(C_p)\cdot\mathcal{C}(C_p)+\mathcal{V}(\langle C_p,C_q\rangle)\cdot\mathcal{C}(\langle C_p,C_q\rangle)+\\&\mathcal{V}(C_q)\cdot\mathcal{C}(C_q)\}\cdot\mathcal{T}(C_p)\cdot\mathcal{T}(\langle C_p,C_q\rangle)\cdot\mathcal{T}(C_p)\end{aligned}\tag{13}$$

即$\mathcal{R}'_i$的风险值为组件的威胁性乘以构成它的各个节点、链路的脆弱性与后果的乘积和. 进一步考虑威胁性的多种可能情况，任意组件均有被执行和未被执行两种可能性，因此要分析三个组件的威胁性组合共存在 $ES=2^3=8$ 个不同执行条件下的事件树，综合考虑所有执行条件，可得该杠铃模型的风险值为

$$\begin{aligned}\mathcal{R}_i=\sum_k^{ES}\mathcal{R}'_{ik}=&\mathcal{V}(C_p)\cdot\mathcal{C}(C_p)\cdot\mathcal{T}(C_p)+\\&\mathcal{V}(\langle C_p,C_q\rangle)\cdot\mathcal{C}(\langle C_p,C_q\rangle)\cdot\mathcal{T}(\langle C_p,C_q\rangle)+\\&\mathcal{V}(C_q)\cdot\mathcal{C}(C_q)\cdot\mathcal{T}(C_q)\end{aligned}\tag{14}$$

进而，用$\mathcal{R}(T_j)$表示测试用例 T_j所覆盖全部杠铃的风险值总和为

$$\mathcal{R}(T_j)=\sum_{i=1}^{NT}\mathcal{R}_i=\sum_{i=1}^{NT}\sum_{k=1}^{ES}\mathcal{R}'_{ik}\tag{15}$$

其中，NT 表示测试用例 T_j所覆盖的杠铃个数. 通过上述对测试用例覆盖风险值的分析与计算，可根据风险值的高低对测试用例进行优先级排序. 将覆盖信息风险等级较高的测试用例视为关键测试用例，进而尽早检测出风险级别较高的错误，提高系统可靠性和测试效率.

综上所述，RA-TCP 的整体框架如图 6 所示，其详细描述如下：

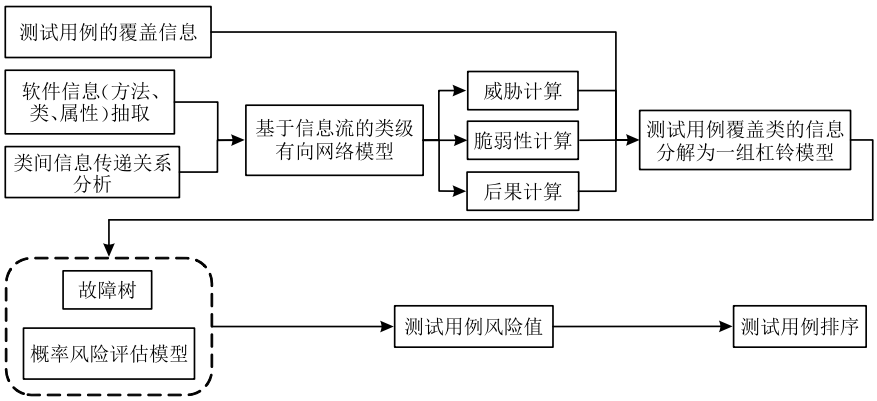


图 6　RA-TCP 框架

算法 2.　基于风险分析的集成测试优先级序列生成.

输入：待分析的软件系统编译后的字节码文件(Jar 包)；
　　测试用例覆盖信息

输出：TPO：测试用例序列

1. 对软件的字节码进行分析，获取软件中所有的类、方法和属性的信息，进而抽取软件系统中的类间信息传递关系；
2. 根据类间信息传递关系，将软件系统映射为相应的 CDNMIF 模型；

3. 计算网络中所有节点和边的威胁性、脆弱性及后果值；
4. 将每一个测试用例覆盖的信息分解为一组“节点-链路-节点”的杠铃；
5. 利用故障树分析所有可能的组合事件，结合 PRA 模型计算每个测试用例所覆盖的风险值；
6. 根据测试用例覆盖的风险值降序排列获取测试用例的优先级序列 TPO.

5 评估指标

为了评估测试用例的缺陷检测速率，我们使用 Elbaum 等人[3]所提出的 $APFD_c$ 作为评估指标对测试用例优先级排序技术进行评测. 该指标将软件中所出现的错误根据其风险指标赋予不同的严重程度，再结合测试用例的执行开销情况得到 $APFD_c$ 的公式：

$$APFD_c=\frac{\sum_{i}^{|\mathcal{F}|}\left(ef_i\times\left(\sum_{j=TF_i}^{|T|}ct_j-\frac{1}{2}\times ct_{TF_i}\right)\right)}{\sum_{i=1}^{|T|}ct_i\times\sum_{i=1}^{|\mathcal{F}|}ef_i} \tag{16}$$

其中，$T=\{t_1,t_2,\cdots,t_{|T|}\}$ 表示软件中的测试用例集，且 $|T|$ 为测试用例集 T 中所包含的测试用例个数；$\mathcal{F}=\{f_1,f_2,\cdots,f_{|\mathcal{F}|}\}$ 表示软件中的错误集，且 $|\mathcal{F}|$ 为错误集 $\mathcal{F}$ 中所检测出的错误数；ct 表示测试用例 $t_i\in T$ 的执行开销；ef 表示错误 $f_i\in\mathcal{F}$ 的严重程度；TF_i 表示在利用测试用例优先级技术所获取的排序序列 ST 中，首次检测到错误 f_i 的测试用例所处的位序.

假设测试用例的执行开销均为 1，在错误集 $\mathcal{F}$ 中，第 i 个错误的严重程度 f_i 表示与包含错误的类之间具有信息传递关系的类集合所构成的杠铃模型风险之和. 由上述公式可知，$APFD_c$ 的结果介于 0 和 100% 之间，$APFD_c$ 的值越大，表明错误的检测速率越高.

6 实验分析

6.1 实验设计

本文采用 4 个 Java 开源软件作为实验分析的数据集：Jmeter、Apache-ant、Joda-time 和 JFreechart. 其中，软件 Jmeter 和 Apache-ant 选取自 SIR①(Subject Infrastructure Repository)，软件 Joda-time 和 JFreechart 来自开源代码库. SIR 测试数据库提供各个软件版本的测试用例及植入错误信息，开源软件 Joda-time 和 JFreechart 包含测试用例套件，参照文献[44]提供的方法，通过分析原始开发人员上传代码的 bug 修复日志，挖掘出各个版本的真实错误情况. 上述 4 个软件是对测试用例优先级排序技术进行实证研究的过程中常被选取的实验对象[45-48]，表 1 为收集的不同软件在不同版本下的统计信息②. 若按照软件缺陷的正交分类法[49]对上述实验数据所包含的错误进行分类，平均 15.1% 的错误为赋值类型错误，84.9% 为接口类型错误.

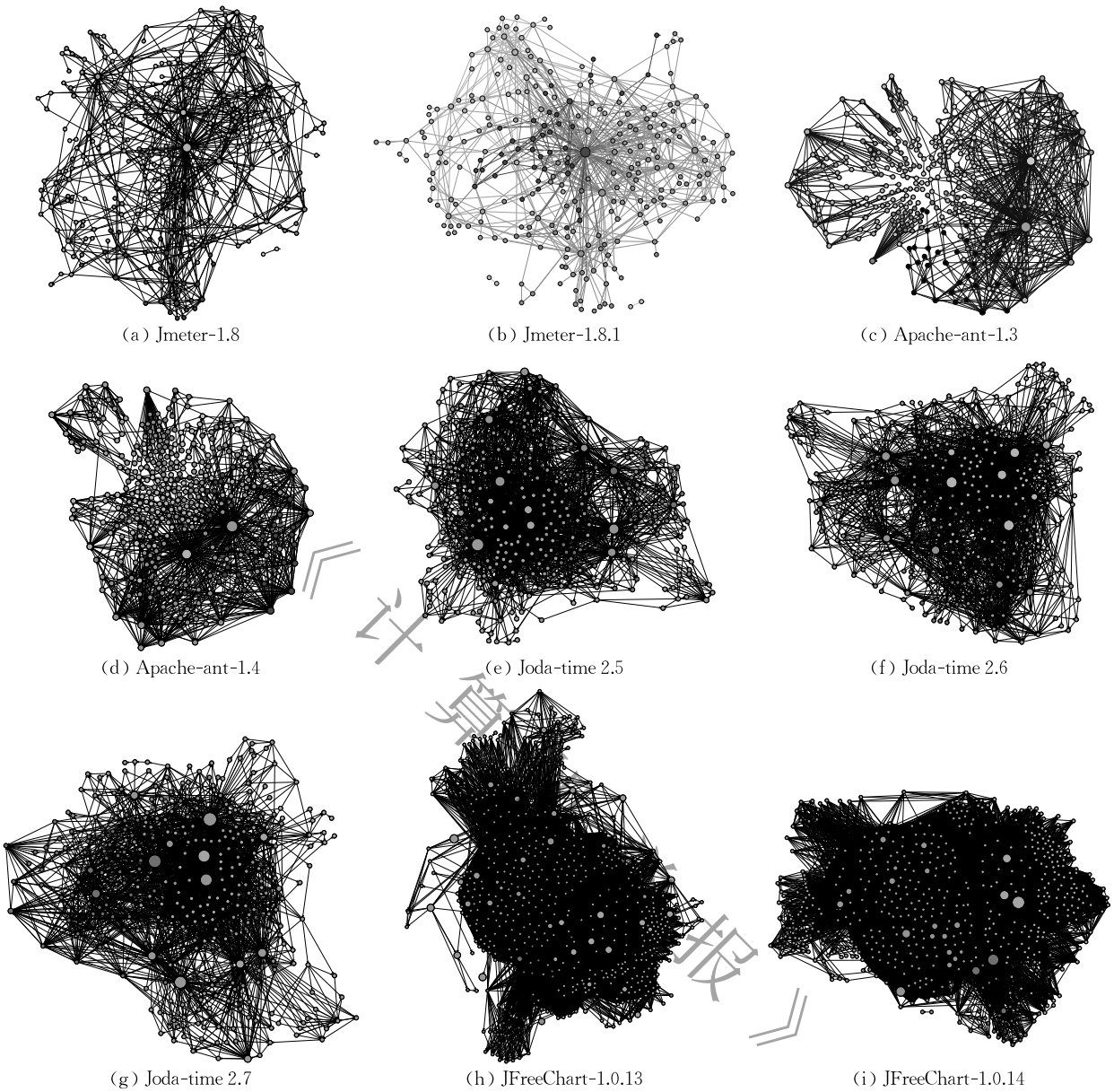
表 1 实验对象统计数据

实验对象	版本	类数	测试用例数	包含错误数
Jmeter-1.8	V1	291	23	19
Jmeter-1.8.1	V2	275	25	20
Apache-ant-1.3	V1	331	34	11
Apache-ant-1.4	V2	358	51	21
Joda-time-2.5	V1	175	149	22
Joda-time-2.6	V2	175	141	20
Joda-time-2.7	V3	178	139	19
JFreeChart-1.0.13	V1	609	356	19
JFreeChart-1.0.14	V2	619	360	10

本文从 4 个软件中选取 9 个不同的版本作为实验数据，借助于工具 Dependency Finder③ 抽取出类间方法和属性的依赖关系数据④，通过分析这些关系数据，分别对每个软件建立 CDNMIF 模型，所抽象出的网络模型如图 7 所示.

实验过程中，利用工具 djUnit 获得测试用例的覆盖信息. djUnit⑤ 可以为每个测试用例提供测试覆盖报告. 通过解析测试覆盖报告便可获得每个测试用例覆盖类的信息⑥. 为了证明算法的有效性和稳定性，除了考虑文献[23]和[27]中提及的 7 种粗粒度的测试用例优先级排序算法外，也加入了基于附加分支覆盖的细粒度优先级排序算法与 RA-TCP 进行对比分析，并验证了排序效果与风险各项指标的相关性. 8 种对比算法的描述说明如表 2 所示. 其中，T2 算法为随机测试序列，这里取其 50 次独立实验的平均结果进行比较；T3 算法是指在已知测试用例检测出的错误位置的情况下，将测试用例按照错误检测率最大的序列进行排序. 然而，这个顺序是在实际预测过程中无法达到的，因此，本文将其作为各

① http://sir.unl.edu/portal/index.php
② 实验系统的测试用例列表数据及植入或真实错误信息：https://github.com/wangying8052/-Regression-Testing-Raw-Data
③ http://depfind.sourceforge.net/
④ 类间方法和属性的依赖关系数据：https://github.com/wangying8052/Relationships
⑤ http://works.dgic.co.jp/djunit/
⑥ 测试用例覆盖类的信息：https://github.com/wangying8052/Coverage



(a) Jmeter-1.8　(b) Jmeter-1.8.1　(c) Apache-ant-1.3

(d) Apache-ant-1.4　(e) Joda-time 2.5　(f) Joda-time 2.6

(g) Joda-time 2.7　(h) JFreeChart-1.0.13　(i) JFreeChart-1.0.14

图 7　9 个软件抽象出的 CDNMIF 模型

表 2　测试用例优先级排序算法

排序技术	算法名	算法描述
T1	Untreated	测试用例的初始顺序
T2	Random	将测试用例随机排序
T3	Optimal	最大化测试用例套件中的故障检测率，按降序执行
T4	Total classes coverage	按测试用例覆盖类的数量降序执行
T5	Additional class coverage	基于反馈信息，按照测试用例覆盖类的数量降序执行
T6	Total different methods coverage	测试用例按上一版本所覆盖的方法新增加的数量降序执行
T7	Additional different methods coverage	基于反馈信息，按照测试用例按上一版本所覆盖的方法新增加的数量降序执行
T8	Additional branch coverage	基于反馈信息，按照测试用例覆盖分支数量降序执行

种排序方法有效性的上限. 由于上述对比算法均非开源程序，故根据文献[23]和[27]的描述，对其进行仿真实现，获取实验结果. 实验平台如下：3.7 GHz CPU，12 GB 内存，1 TB 硬盘，Windows 8.1 以及编译环境 Eclipse 4.5.0.

6.2　实验结果统计与分析

6.2.1　系统风险结果分析

利用 RA-TCP 算法求得不同软件版本间测试

用例的风险覆盖率分布情况如图 8 所示. 系统中只有少数测试用例覆盖了高风险等级的功能路径. 假设将风险指数超过 0.1 的测试用例视为具有较高风险等级. 则高风险等级的测试用例占下列系统中测试用例总数的比例分别为: Jmeter-V1 中包含 25%, Jmeter-V2 中包含 4%, Apache-ant-V1 中包含 17%, Apache-ant-V2 中包含 12%, Joda-time-V1 中包含 26%, Joda-time-V2 中包含 29%, Joda-time-V3 中包含 24%, JFreeChart-V1 中包含 7%, JFreeChart-V2 中包含 6%. 平均每个软件中仅含有 17%的高风险等级测试用例, 故测试用例的风险分布情况符合 Pareto 定律, 即系统中 80%的高风险错误由 20%的关键测试用例所覆盖.

以图 8(a)为例, 在软件系统 Jmeter-V1 中, 第 13 个测试用例占据最高的风险比例. 通过对源代码的分析发现, 测试用例 InterleaveControl $Test 覆盖了 9 个杠铃, 且随着该测试用例的执行, 可以检测出类 InterleaveControl, HTTPSampler 和 PowerTable-Model 内所包含的 7 个错误, 约占系统中错误总数的 36.8%.

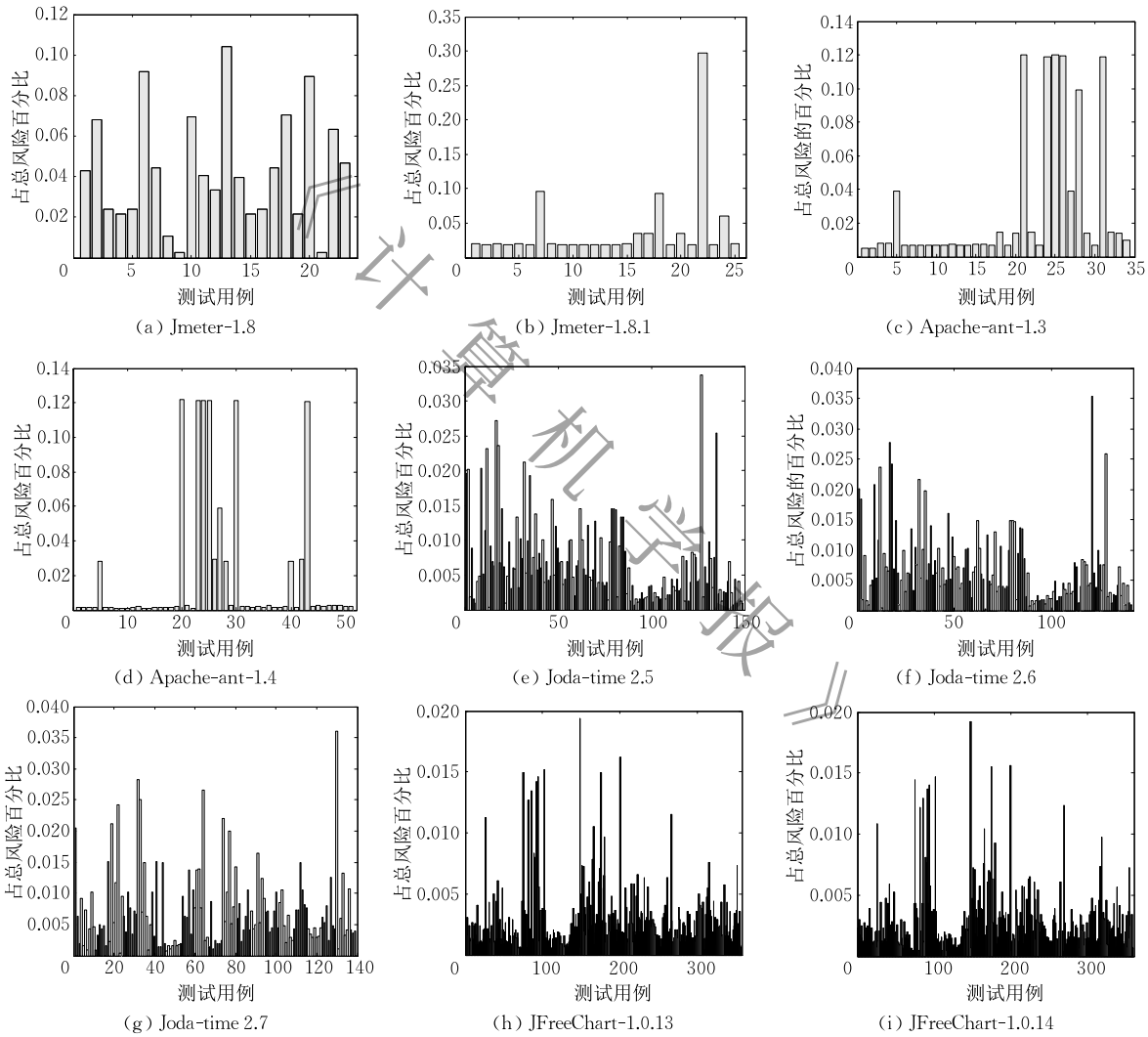


图 8 测试用例覆盖风险的指数分布

同样的, 如图 8(b)所示, Jmeter-V2 中第 22 个测试用例 ProxyControl $Test 的风险权重接近 0.3, 导致上述结果的主要原因是此测试用例对功能路径覆盖率为系统最高值, 因而检测到错误的可能性更大. 对比之下, 图 8(b)中第 1 个测试用例 GzipTest 在 Apache-ant-V1 中所占的风险权重最低, 仅覆盖了 2 个风险指数较低的杠铃. 从整体上看, 软件中近 80%的风险可被 6 个关键测试用例所覆盖, 即测试用例执行一半的情况下, 系统可靠性便陡然提升. 综上所述, 根据测试用例覆盖风险的比例进行排序, 能够优先检测出严重级别较高的错误, 提高测试效率.

6.2.2　有效性分析

为了验证 RA-TCP 的有效性，本文在 4 个软件不同版本间分别计算了 RA-TCP 与其他 8 种排序技术的错误检测率，$APFD_c$ 指标统计结果如表 3 所示，其对应的箱形图如图 9 所示. 图 10 是 RA-TCP 与其它 8 种排序算法在不同软件间 $APFD_c$ 值的对比情况. 实证结果表明，不同的软件基于 RA-TCP 所获取的 $APFD_c$ 值最接近最优排序方案 T3 的结果. 其中，由于 T1 没有考虑测试用例的结构复杂性而直接利用测试用例的初始顺序执行用例，导致 RA-TCP 与 T1 算法相比优势最为明显，其 $APFD_c$ 值提高了 23.6%. RA-TCP 比 T2 算法的 $APFD_c$ 值提高了 15.4%，比 T4 算法的 $APFD_c$ 值提高了 6.1%，比 T5 算法的 $APFD_c$ 值提高了 7.2%，比 T6 算法的 $APFD_c$ 值提高了 3.8%，比 T7 算法的 $APFD_c$ 值提高了 5.6%，甚至比细粒度的分支覆盖优先级算法的准确率提高了 1.6%. 由此可见，RA-TCP 算法具备较强的错误检测能力.

表 3　所有排序算法 $APFD_c$ 值统计

软件	度量	排序算法								
		RA-TCP	T3	T1	T2	T4	T5	T6	T7	T8
Jmeter-1.8	$APFD_c$	0.900	**0.930**	0.860	0.710	0.820	0.810	0.890	0.910	0.897
	执行时间/s	4.854	0.001	0.010	0.035	0.034	0.051	0.023	1.374	1.431
Jmeter-1.8.1	$APFD_c$	0.940	**0.948**	0.596	0.755	0.903	0.932	0.936	0.936	0.942
	执行时间/s	4.072	0.001	0.009	0.031	0.031	0.047	0.022	1.369	1.400
Apache-ant-1.3	$APFD_c$	0.864	**0.939**	0.421	0.517	0.821	0.864	0.845	0.788	0.832
	执行时间/s	7.242	0.002	0.022	0.071	0.075	0.096	0.055	2.809	2.873
Apache-ant-1.4	$APFD_c$	0.979	**0.984**	0.960	0.910	0.960	0.967	0.940	0.960	0.963
	执行时间/s	7.976	0.003	0.027	0.080	0.087	0.117	0.065	2.877	2.938
Joda-time-2.5	$APFD_c$	**0.936**	**0.936**	0.466	0.811	0.749	0.654	0.721	0.650	**0.936**
	执行时间/s	11.560	0.004	0.047	0.093	0.104	0.153	0.094	3.361	3.494
Joda-time-2.6	$APFD_c$	**0.941**	**0.941**	0.703	0.888	0.782	0.703	0.936	0.861	0.907
	执行时间/s	13.376	0.004	0.051	0.102	0.116	0.148	0.108	3.297	3.013
Joda-time-2.7	$APFD_c$	**0.946**	**0.946**	0.753	0.830	0.946	0.946	0.927	0.918	**0.946**
	执行时间/s	13.692	0.005	0.049	0.099	0.105	0.164	0.112	3.328	3.164
JFreeChart-1.0.13	$APFD_c$	**0.923**	**0.923**	0.833	0.771	0.918	0.916	0.913	0.921	0.902
	执行时间/s	22.679	0.008	0.125	0.307	0.428	0.531	0.642	9.976	9.860
JFreeChart-1.0.14	$APFD_c$	**0.948**	**0.948**	0.662	0.801	0.932	0.937	0.929	0.928	0.911
	执行时间/s	24.975	0.009	0.159	0.328	0.453	0.513	0.753	9.341	9.124
Average	$APFD_c$	0.931	0.944	0.695	0.777	0.870	0.859	0.893	0.875	0.915

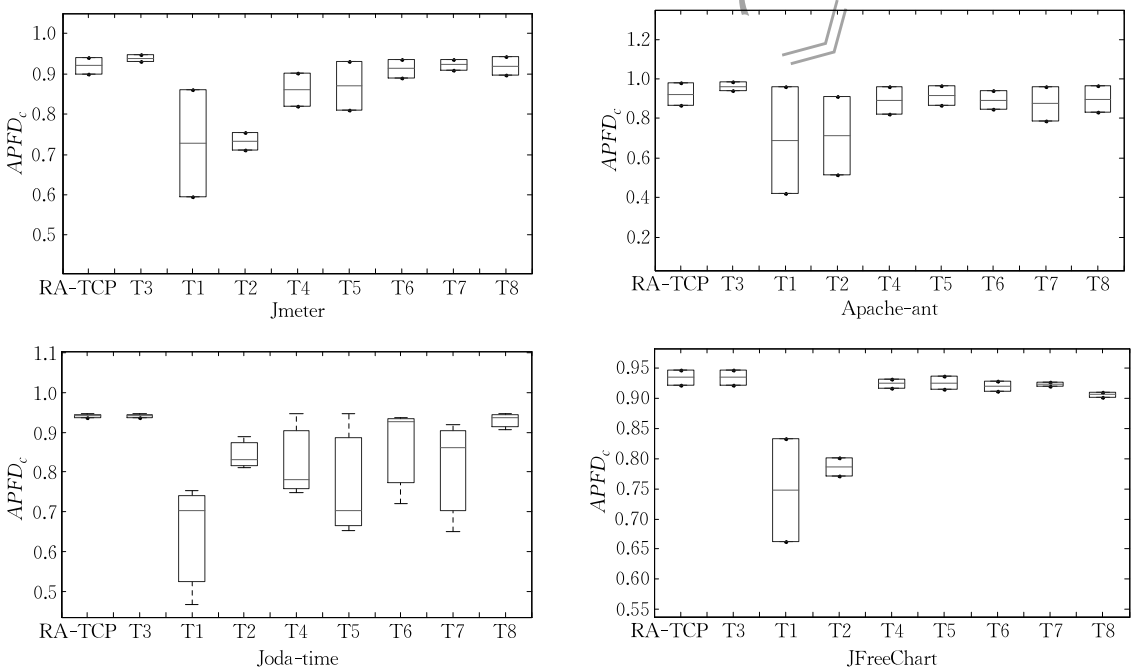


图 9　所有排序算法 APFD$_c$ 值箱形图

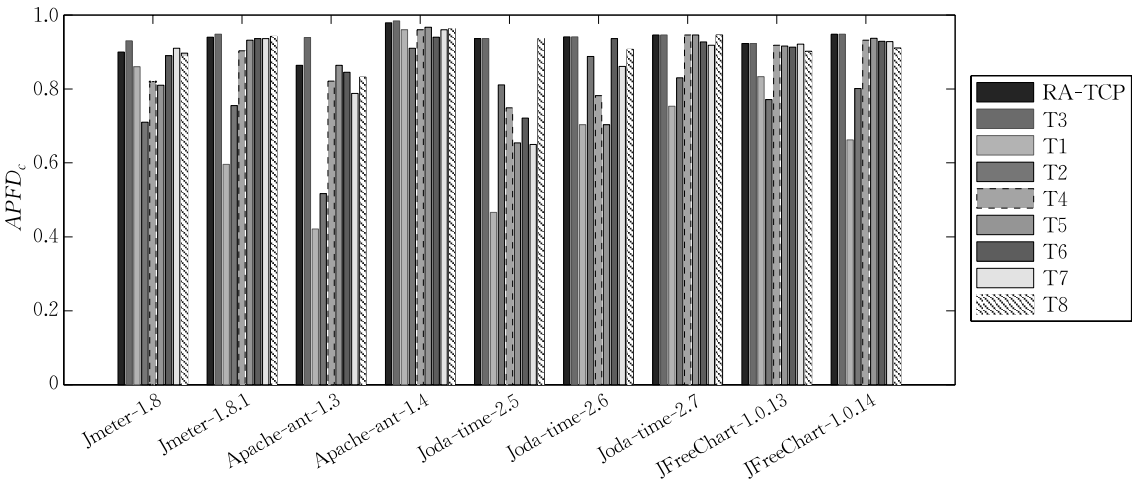


图 10 所有排序算法在不同软件间的 $APFD_c$ 值

研究发现 5 个软件版本通过不同的排序算法所获取的 $APFD_c$ 值都达到了最优. 而对于软件 JFreeChart,RA-TCP 的 $APFD_c$ 值明显高于其他排序算法. 通过对软件源码和测试用例及植入错误类型的分析,软件 JFreeChart-V1 中第 148 个测试用例 CategoryPlotTests 杠铃的覆盖率最高,其中 8 个杠铃包含缺陷. 以包含缺陷类 CategoryAxis 和 CategoryPlot 构成的杠铃模型为例,组成杠铃的节点、链路和节点的执行概率分别为 0.13,0.06,0.12,即通过该杠铃的信息传递管道是网络中的主要功能执行路径,相对频繁被执行. 同时,RA-TCP 算法对包含在该杠铃内部的接口和赋值类型的错误较为敏感,由于该两种错误类型所导致的失效将使得功能执行路径发生阻塞或错误运行,且将错误传播到了软件网络中其他的 22 个类,损失了 35%的信息流量,造成故障的波及范围较大. 因此,覆盖高风险杠铃的测试用例应该被优先执行.

从算法的执行时间来看,粗粒度优先级排序算法 T1,T2,T4,T5,T6 用时较短且十分相近;T7 算法需要进行版本间的代码的差分操作,T8 算法采用插桩法分析测试用例覆盖的源代码语句的分支情况,故该两种算法用时相对较长. 抽取系统的依赖关系并对其建立网络拓扑模型,进而分析其节点的风险因子,尽管上述复杂的计算步骤占用了 RA-TCP 方法一定的运行时间,但是对于系统和测试用例规模都相对较大的 Joda-time 和 JFreeChart 系统而言,其执行时间与 T7 和 T8 算法相差不大,计算效率尚可接受.

6.2.3 稳定性分析

为了验证 RA-TCP 的稳定性,本文对不同软件的 8 种排序算法所获取的 $APFD_c$ 值的均方差进行对比分析,结果如表 4 所示. 综合看来,RA-TCP 算法所得 $APFD_c$ 的均方差最接近于 T3 算法的结果且平均值不大于其他 8 种排序技术. 特别是在系统 Joda-time 和 JFreeChart 中,稳定性达到最优状态.

表 4 所有排序算法在不同软件间的 $APFD_c$ 值均方差

排序技术	Jmeter	Apache-ant	Joda-time	JFreeChart
RA-TCP	0.0008	0.0066	2.50e-05	0.0003
T3	0.0002	0.0010	2.50e-05	0.0003
T1	0.0348	0.1453	0.0235	0.0146
T2	0.0010	0.0772	0.0016	0.0005
T4	0.0034	0.0097	0.0111	0.0001
T5	0.0074	0.0097	0.0245	0.0002
T6	0.0011	0.0045	0.0148	0.0001
T7	0.0003	0.0148	0.0199	2.50e-05
T8	0.0010	0.0086	0.0004	4.05e-05

进一步利用单因素方差分析[51]的方法比较在上述四个软件系统中 RA-TCP 与最优排序方案 T3 算法间的 $APFD_c$ 值,同时,计算出同样通过度量软件节点测试重要性而进行测试用例优先级排序的 CSN-TCP[11]和 Apros[50]两种算法与 T3 算法间 $APFD_c$ 值的方差分析结果. 如表 5 所示,显著性水平设置为 0.05,结果行表示两种算法间是否有显著性差异. 由表 5 可知,RA-TCP 与 T3 算法间 $APFD_c$ 值在两个软件中的 P 值远大于 0.05,说明两种算法间没有显著性差异. 通过效应量的统计判断 RA-TCP 与 T3 算法间 $APFD_c$ 值的差异大小可知,两个软件的效应量均较小,且在软件 Joda-time 和 JFreechart 中为 0,即两算法间差异较小. 同理,CSN-TCP[11]、Apros[50]与 T3 算法间均无显著差异,其 P 值均大于 0.05,且两算法与 T3 算法间 $APFD_c$ 值在两个软件中的 P 值均小于 RA-TCP,效应量均大于

RA-TCP.即与其他相似的优先级排序技术相比，RA-TCP 与 T3 算法的差异性更小，结果更相近.从算法的稳定性和有效性的结果来看，RA-TCP 算法适用于检测赋值和接口类型的错误，同时作用于不同规模的测试用例套件和软件系统皆可取得较好结果.

表 5　基于 RA-TCP 与 T3 算法间 $APFD_c$ 值的方差分析

软件	度量	RA-TCP	CSN-TCP[11]	Apros[50]
Jmeter	P 值	0.334	0.279	0.245
	效应量	0.201	0.659	0.713
	结果	N	N	N
Apache-ant	P 值	0.458	0.365	0.221
	效应量	0.321	0.526	0.688
	结果	N	N	N
Joda-time	P 值	1	0.542	0.408
	效应量	0	0.508	0.646
	结果	N	N	N
JFreeChart	P 值	1	0.437	0.259
	效应量	0	0.541	0.632
	结果	N	N	N

6.2.4　相关性分析

为了分析 RA-TCP 算法的有效性与测试用例所覆盖类的威胁性、脆弱性、后果及风险之间的关系，本文利用 4 个软件 9 个版本中基于 RA-TCP 算法与 T3 算法所获取的 $APFD_c$ 比值作为最优比，进而分析最优比与$\mathcal{T}$、$\mathcal{V}$、$\mathcal{C}$、$\mathcal{R}$ 间的 Pearson 相关性，得到结果如表 6、表 7 所示.结果表明，$APFD_c$ 最优比与测试用例所覆盖类的威胁性、脆弱性和后果均存在弱正相关性，与风险值存在较强正相关性，由于测试用例的风险值$\mathcal{R}$由$\mathcal{T}$、$\mathcal{V}$、$\mathcal{C}$参数的乘积所决定，因此$\mathcal{R}$与最优比的相关性更明显.通过分析可知，随着测试用例所覆盖类的威胁性、脆弱性、后果及风险的同步增长，RA-TCP 算法所获取的测试用例序列可以具有更高的缺陷检测效率.即 RA-TCP 算法的缺陷检测能力受软件系统的拓扑结构及类内部的代码语句复杂程度影响，结构越复杂，风险因子越大，排序算法的效果越显著.

表 6　不同软件间的软件结构参数

	Jmeter-1.8	Jmeter-1.8.1	Apache-ant-1.3	Apache-ant-1.4	Joda-time-2.5	Joda-time-2.6	Joda-time-2.7	JFreeChart-1.0.13	JFreeChart-1.0.14
$APFD_c$ 比	0.900/0.930	0.940/0.948	0.864/0.939	0.979/0.984	0.936/0.936	0.941/0.941	0.946/0.946	0.923/0.923	0.948/0.948
$\mathcal{T}$	0.008588	0.008592	0.010272	0.011521	0.010963	0.009830	0.010834	0.009964	0.011675
$\mathcal{V}$	0.003217	0.031107	0.039098	0.053170	0.060934	0.003120	0.029674	0.0371425	0.051075
$\mathcal{C}$	0.005179	0.004926	0.002215	0.002875	0.004787	0.004524	0.004278	0.002649	0.003289
$\mathcal{R}$	0.043478	0.040000	0.029412	0.019608	0.074115	0.041674	0.058300	0.048030	0.058520

表 7　Pearson 相关性

	$\mathcal{T}$	$\mathcal{V}$	$\mathcal{C}$	$\mathcal{R}$
$APFD_c$ 比	0.2435	0.1257	0.3556	0.3615

6.3　效度威胁

借助于 Dependency Finder 工具可对 Java 语言的软件系统进行解析，提取出类间及方法和属性之间的不同粒度的依赖关系，鉴于此，本文的实验对象皆选取的是 Java 语言编写的开源软件.需要澄清的是，所提出的 RA-TCP 算法对 C++等其他面向对象语言编写的软件同样适用.

djUnit 为每个测试用例生成的测试覆盖报告只包含类的信息，为获取其对杠铃模型的覆盖情况，本文将其结合 Dependency Finder 提取出的测试用例类与其他类的依赖关系，通过分析最终得到更细粒度的覆盖信息.由于所得测试用例覆盖信息的正确性对实验结果有较大影响，因此，本文提供该部分原始实验数据的下载链接，以供研究人员重复实验时验证.

由于实验涉及的 7 种粗粒度测试用例优先级排序算法和基于附加分支覆盖的细粒度优先级排序算法均为非开源程序，需要重新编写实现后获取实验结果.为保证实验结果的准确性，实现过程中，采用与文献中相同的参数及算法框架.同时为保证执行时间度量的公平性，所有对比算法皆基于 Java 语言实现，在 Eclipse 4.5.0 编译环境中运行.

7　结　论

本文通过软件系统中类间信息流的传递关系构建类级有向网络模型，将测试用例所覆盖的结构信息抽象成一组“节点-链路-节点”的杠铃模型，结合故障树理论分析具有信息传递关系的两个类之间发生失效的所有可能方式，利用概率风险评估方法来度量测试用例所覆盖的风险值，从而获取测试用例的优先级序列，保证软件系统中的风险指数较高的错误与缺陷可以被尽早被发现，提高测试效率.通过将本文提出的 RA-TCP 算法与现有算法进行对比

分析,证明 RA-TCP 算法具有较高的错误检出率,且其算法的有效性与稳定性均高于其他算法.根据 Pearson 相关性的分析结果显示,测试用例的风险值与类的威胁性、脆弱性和后果均存在正相关性.

参 考 文 献

[1] Kung D C, Gao J, Pei H, et al. On regression testing of object-oriented programs. Journal of Systems & Software, 1996, 32(1): 21-40

[2] Harrold M J. Reduce, reuse, recycle, recover: Techniques for improved regression testing//Proceedings of the 25th IEEE International Conference on Software Maintenance (ICSM). Alberta, Canada, 2009: 5

[3] Elbaum S, Malishevsky A, Rothermel G. Incorporating varying test costs and fault severities into test case prioritization//Proceedings of the 23rd IEEE International Conference on Software Engineering (ICSE). Washington, USA, 2001: 329-338

[4] Wong W E, Horgan J R, London S, et al. A study of effective regression testing in practice//Proceedings of the 8th IEEE International Symposium on Software Reliability Engineering (ISSRE). Albuquerque, USA, 1997: 264-274

[5] Islam M M, Marchetto A, Susi A, et al. A multi-objective technique to prioritize test cases based on latent semantic indexing//Proceedings of the 16th IEEE European Conference on Software Maintenance and Reengineering (CSMR). Szeged, Hungary, 2012: 21-30

[6] Hao D, Zhang L, Zang L, et al. To be optimal or not in test-case prioritization. IEEE Transactions on Software Engineering, 2015, 42(5): 490-505

[7] Yacoub S M, Ammar H H. A methodology for architecture-level reliability risk analysis. IEEE Transactions on Software Engineering, 2002, 28(6): 529-547

[8] Hettiarachchi C, Do H, Choi B. Risk-based test case prioritization using a fuzzy expert system. Information & Software Technology, 2015, 69(C): 1-15

[9] Cui S, Bin S, Sun G. Research of structural risks of object-oriented software based on ripple degree of software network. International Journal of Multimedia and Ubiquitous Engineering, 2016, 11(1): 221-228

[10] Wang X F, Chen G. Complex networks: Small-world, scale-free and beyond. IEEE Circuits & Systems Magazine, 2003, 3(1): 6-20

[11] Li Z, Harman M, Hierons R M. Search algorithms for regression test case prioritization. IEEE Transactions on Software Engineering, 2007, 33(4): 225-237

[12] Rothermel G, Untch R H, Chu C, et al. Prioritizing test cases for regression testing. IEEE Transactions on Software Engineering, 2001, 27(10): 929-948

[13] Do H, Mirarab S, Tahvildari L, et al. The effects of time constraints on test case prioritization: A series of controlled experiments. IEEE Transactions on Software Engineering, 2010, 36(5): 593-617

[14] Elbaum S, Gable D, Rothermel G. Understanding and measuring the sources of variation in the prioritization of regression test suites//Proceedings of the 7th IEEE International Software Metrics Symposium (METRICS). London, England, 2001: 169-179

[15] Jeffrey D, Gupta N. Test case prioritization using relevant slices//Proceedings of the 30th IEEE Annual International Computer Software and Applications Conference (COMPSAC). Chicago, USA, 2006: 411-420

[16] Mei H, Hao D, Zhang L, et al. A static approach to prioritizing JUnit test cases. IEEE Transactions on Software Engineering, 2012, 38(6): 1258-1275

[17] Yoo S, Harman M, Tonella P, et al. Clustering test cases to achieve effective and scalable prioritisation incorporating expert knowledge//Proceedings of the 18th International Symposium on Software Testing and Analysis (ISSTA). Chicago, USA, 2009: 201-212

[18] Pan Wei-Feng, Li Bing, Ma Yu-Tao, Liu Jing. Test case prioritization based on complex software networks. Acta Electronica Sinica, 2012, 40(12): 2456-2465(in Chinese)
(潘伟丰, 李兵, 马于涛, 刘婧. 基于复杂软件网络的回归测试用例优先级排序. 电子学报, 2012, 40(12): 2456-2465)

[19] Xie X, Chen T Y, Kuo F C, et al. A theoretical analysis of the risk evaluation formulas for spectrum-based fault localization. ACM Transactions on Software Engineering & Methodology, 2013, 22(4): 1-31

[20] Felderer M, Ramler R. Integrating risk-based testing in industrial test processes. Software Quality Journal, 2014, 22(3): 542-575

[21] Schneidewind N, Hinchey M. Risk-driven software reliability and testing//Proceedings of the 2nd International Conference on Secure System Integration and Reliability Improvement (SSIRI). Yokohama, Japan, 2008: 183-184

[22] Amland S. Risk-based testing: Risk analysis fundamentals and metrics for software testing including a financial application case study. Journal of Systems and Software, 2000, 53(3): 287-295

[23] Chen Y, Probert R L, Sims D P. Specification-based regression test selection with risk analysis//Proceedings of the 2002 Conference of the Centre for Advanced Studies on Collaborative Research (CASCON). Toronto, Canada, 2002: 1-14

[24] Zimmermann F, Eschbach R, Kloos J, et al. Risk-based statistical testing: A refinement-based approach to the reliability analysis of safety-critical systems//Proceedings of the 12th European Workshop on Dependable Computing (EWDC). Toulouse, France, 2009: 1-8

[25] Li Bing, Ma Yu-Tao, Liu Jing, Ding Qi-Wei. Advances in the study on complex networks of software systems. Advances in Mechanics, 2008, 38(6): 805-814(in Chinese)

(李兵，马于涛，刘婧，丁琦伟. 软件系统的复杂网络研究进展. 力学进展，2008，38(6)：805-814)

[26] Valverde S，Cancho R F，Solé R V. Scale free networks from optimal design. Europhysics Letters，2002，60(4)：512-517

[27] Moura A，Lai Y C，Motter A E. Signatures of small world and scale free properties in large computer programs. Physical Review E，2003，68：046116

[28] Wheeldon R，Counsell S. Power law distributions in class relationships//Proceedings of the 3rd IEEE International Workshop on Source Code Analysis & Manipulation (SCAM). Amsterdam，Netherlands，2003：45-54

[29] Valverde S，Solé R V. Hierarchical small worlds in software architecture. Working Paper of Santa Fe Institute，2003，SFI/03-07-044

[30] Myers C R. Software systems as complex networks：Structure，function，and evolvability of software collaboration graphs. Physical Review E，2003，68(4)：352-375

[31] Wang Bei-Yang，Lü Jin-Hu. Software networks nodes impact analysis of complex software systems. Journal of Software，2013，24(12)：2814-2829(in Chinese)
(汪北阳，吕金虎. 复杂软件系统的软件网络结点影响分析. 软件学报，2013，24(12)：2814-2829)

[32] Wang Ying，Yu Hai，Zhu Zhi-Liang. A class integration test order method based on the node importance of software. Journal of Computer Research and Development，2016，53(3)：517-530(in Chinese)
(王莹，于海，朱志良. 基于软件节点重要性的集成测试序列生成方法. 计算机研究与发展，2016，53(3)：517-530)

[33] Pan Wei-Feng，Li Bing，Zhou Xiao-Yan，He Peng. Regression test case prioritization based on bug propagation network. Journal of Computer Research and Development，2016，53(3)：550-558 (in Chinese)
(潘伟丰，李兵，周晓燕，何鹏. 基于错误传播网络的回归测试用例排序方法. 计算机研究与发展，2016，53(3)：550-558)

[34] GJB8900-90. General Program for System Safety. Beijing：Military Standard Publishing Department of Commission of Science Technology and Industry for National Defense，1999 (in Chinese)
(GJB8900-90. 系统安全性通用大纲. 北京：国防科工委军标出版发行部，1999)

[35] NASA Technical Std. NASA-STD-8719. 13A，Software Safety，1997

[36] Gokhale S S. Architecture-based software reliability analysis：Overview and limitations. IEEE Transactions on Dependable & Secure Computing，2007，4(1)：32-40

[37] Cooke R，Bedford T. Probabilistic Risk Analysis：Foundations and Method. Cambridge：Cambridge University Press，2001

[38] Jonkman S N，Gelder P，Vrijling J K. An overview of quantitative risk measures for loss of life and economic damage. Journal of Hazardous Materials，2003，99(1)：1-30

[39] Ma Yu-Tao，He Ke-Qing，Li Bing，Liu Jing. Empirical study on the characteristics of complex networks in networked software. Journal of Software，2011，22(3)：381-407 (in Chinese)
(马于涛，何克清，李兵，刘婧. 网络化软件的复杂网络特性实证. 软件学报，2011，22(3)：381-407)

[40] Henry S，Kafura D. Software structure metrics based on information flow. IEEE Transactions on Software Engineering，1981，SE-7(5)：510-518

[41] Maggie H，Katerina G. Common trends in software fault and failure data. IEEE Transactions on Software Engineering，2009，35(4)：484-496

[42] Pan Wei-Feng，Li Bing. Software quality measurement based on error propagation analysis in software networks. Journal of Central South University(Science and Technology)，2012，43(11)：4339-4348(in Chinese)
(潘伟丰，李兵. 基于软件网络错误传播分析的软件质量量度. 中南大学学报(自然科学版)，2012，43(11)：4339-4348)

[43] Bucci P，Kirschenbaum J，Mangan L A，et al. Construction of event-tree/fault-tree models from a Markov approach to dynamic system reliability. Reliability Engineering & System Safety，2008，93(11)：1616-1627

[44] Just R，Jalali D，Ernst M D，et al. Are mutants a valid substitute for real faults in software testing?//Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE). Hong Kong，China，2014：654-665

[45] Zhang L，Zhou J，Hao D，et al. Prioritizing JUnit test cases in absence of coverage information//Proceedings of the 25th IEEE International Conference on Software Maintenance (ICSM). Alberta，Canada，2009：19-28

[46] Fang C，Chen Z，Wu K，et al. Similarity-based test case prioritization using ordered sequences of program entities. Software Quality Control，2014，22(22)：335-361

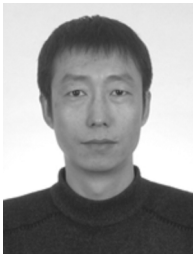
[47] Rachatasumrit N，Kim M. An empirical investigation into the impact of refactoring on regression testing//Proceedings of the 28th IEEE International Conference on Software Maintenance(ICSM). Trento，Italy，2012：357-366

[48] Hao D，Zhang L，Zhang L，et al. A unified test case prioritization approach. ACM Transactions on Software Engineering & Methodology，2014，24(2)：1-31

[49] Chillarege R，Bhandari I S，Chaar J K，et al. Orthogonal defect classification—A concept for in-process measurements. IEEE Transactions on Software Engineering，1992，18(11)：943-956

[50] Ma Z，Zhao J. Test case prioritization based on analysis of program structure//Proceedings of the 15th Asia-Pacific Software Engineering Conference (APSEC). Beijing，China，2008：471-478

[51] Zhang L，Hou S S，Guo C，et al. Time-aware test-case prioritization using integer linear programming//Proceedings of the 18th International Symposium on Software Testing and Analysis(ISSTA). Chicago，USA，2009：665-669



YU Hai, Ph. D., associate professor, M. S. suprvisor. His research interests include complex networks, chaotic encryption, software testing, software refactoring, and software architecture.

YANG Yue, M. S. candidate. Her research interests include complex network, and software testing.

WANG Ying, Ph. D. Her research interests include software refactoring, software testing, software reliability, software architecture and complex network.

ZHANG Wei, Ph. D., associate professor, M. S. supervisor. His research interests include signal processing, multimedia coding and software architecture.

ZHU Zhi-Liang, Ph. D., professor. His research interests include chaotic enoryption, complex network, software testing and software refactoring.

Background

Regression testing must be conducted to confirm that recent program changes have not adversely affected existing features and new tests must be conducted to test new features. Testers might rerun all test cases generated at earlier stages to ensure that the program behaves as expected. However, as a program evolves the regression test set grows larger, old tests are rarely discarded, and the expense of regression testing grows. Test case prioritization seeks to provide a way to run test cases with the highest priority earliest. To reduce the cost of regression testing, software testers may prioritize their test cases so that those which are more important, by some measure, are executed earlier in the regression testing process. For example, testers might wish to schedule test cases in an order that achieves code coverage at the fastest rate possible, exercise features in order of expected frequency of use, or exercises subsystems in an order that reflects their historically demonstrated propensity to fail. Numerous empirical studies have shown that prioritization techniques can significantly improve rate of fault detection. Those studies, however, raised several additional questions: (1) Can prioritization techniques be effective when targeted at specific components with high risk factors; (2) are the information transfer relationships between components taken into full account in the prioritizing process; (3) can the incorporation of risk analysis for software system into prioritization techniques improve their effectiveness?

According to our empirical results, the optimal technique is slightly better than the additional technique with ignorable difference for achieving optimal coverage. However, the optimal technique is significantly worse than the additional technique for most target programs in terms of fault detection. Moreover, although both the optimal technique and the additional technique significantly outperform the ideal technique in terms of coverage, the latter significantly outperforms the former two techniques in terms of fault detection. Therefore, in test-case prioritization, it is not worthwhile to pursue optimality by taking the coverage as an intermediate goal.

This paper proposes a Risk Analysis-based Test Case Prioritization (RA-TCP) algorithm based on the transmission of information flows among software components. Firstly, we map software system into a class-level information flow-based directed network model. Then, class relationships covered by each test case are represented by a set of barbell models. Finally, combining with probabilistic risk analysis and fault tree theory, we assign a priority to each test case by calculating the sum of risk indexes of all the barbell models covered by it. Experimental results demonstrate that the higher risk index the test case has, the better fault coverage is. RA-TCP increases the detection rate of faults with serious risk indicators. By comparing with the other state of art prioritization algorithms, we can draw the following conclusions:

(1) RA-TCP has a higher error detection rate and performs stable across different systems.

(2) RA-TCP technique can maximize the system risk reduction rate. Especially, this advantage will be more obvious if the test case suite covers more high-risk barbell motifs of the system.

(3) The prioritization techniques based on historical code change information are more appropriate to regression testing, while RA-TCP which based on complexity metric, can be applied to solve the non-regression testing problem.

The research was supported by the National Natural Science Foundation of China (Grant Nos. 61374178, 61402092, 61603082), The online education research fund of MOE research center for online education, China (Qtone education, Grant No. 2016ZD306) and the Ph. D. Start-up Foundation of Liaoning Province, China (Grant No. 201501141).