集成测试中的类测试顺序生成技术述评

张艳梅[1),2),3)] 姜淑娟[1),2)] 张 妙[1)] 鞠小林[4)]

[1)](中国矿业大学计算机科学与技术学院 江苏 徐州 221116)

[2)](广西可信软件重点实验室(桂林电子科技大学) 广西 桂林 541004)

[3)](南京大学计算机软件新技术国家重点实验室 南京 210093)

[4)](南通大学计算机科学与技术学院 江苏 南通 226019)

摘 要 对于面向对象程序,一个常见的问题是确定集成测试中的类的测试顺序,称为类集成测试顺序的确定问题.类测试顺序的确定问题是面向对象软件集成测试中的关键难点之一.首先,简单介绍类测试顺序确定问题的背景.其次,概括描述类集成测试顺序问题以及其产生的原由,并介绍类间依赖关系(包括类间静态依赖关系和动态依赖关系)和抽象类的定义与特点,接着对类测试顺序的确定问题进行分类并评析两种分类方式,包括破除环路的方式和估算测试桩代价的方式.其中,破除环路主要采用基于图论和基于搜索这两类方法,估算测试桩代价主要根据评价所构造的测试桩的个数多少和所构造的测试桩的总体复杂度大小这两个指标.再次,对现有解决类集成测试顺序问题的典型技术进行分类,分为基于图论和基于搜索技术两大类.然后,全面系统地分析这些相关技术的研究现状、特点等,之后还介绍了已有典型的基于图论和基于搜索技术技术在实验过程中各自所使用的评测数据集等.最后,指出未来的研究方向,并对该文进行总结.

关键词 软件测试;类测试顺序;集成测试;破除环路;测试桩代价

中图法分类号 TP311 **DOI号** 10.11897/SP.J.1016.2018.00670

Survey of Class Test Order Generation Techniques for Integration Test

ZHANG Yan-Mei[1),2),3)] JIANG Shu-Juan[1),2)] ZHANG Miao[1)] JU Xiao-Lin[4)]

[1)](*School of Computer Science and Technology, China University of Mining and Technology, Xuzhou, Jiangsu* 221116)

[2)](*Guangxi Key Laboratory of Trusted Software, Guilin, Guangxi* 541004)

[3)](*State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing* 210093)

[4)](*School of Computer Science and Technology, Nantong University, Nantong, Jiangsu* 226019)

Abstract In the context of object-oriented software, a common problem is the determination of test orders for the integration test of classes, known as the class integration and test order problem. Determination of class integration test order is a key and difficult point in object-oriented software integration testing. Firstly, the background of class integration test order problem is simply introduced. Secondly, the class integration test order determination problem is outlined and the origin of class integration test order problem is analyzed, and then some related definitions such as inter-class dependencies (including inter-class static dependencies and inter-class static dependencies) and abstract class are introduced. Inter-class dependencies include inter-class static dependencies and inter-class dynamic dependencies, and abstract class has the characteristic of cannot be instantiated, which can affect the class integration test order. The solutions of class integration test order determination problem are classified to two ways, which are the cycle-

收稿日期:2016-06-08;在线出版日期:2017-05-19.本课题得到国家自然科学基金(61502497,61673384)、广西可信软件重点实验室开放课题(kx201530,kx201609)、南京大学计算机软件新技术国家重点实验室开放课题(KFKT2014B19)、中国博士后科学基金项目(2015M581887)和徐州市科技计划项目(KC15SM051)资助.**张艳梅**,女,1982年生,博士,讲师,中国计算机学会(CCF)会员,主要研究方向为软件分析与测试等.E-mail:ymzhang@cumt.edu.cn.**姜淑娟**(通信作者),女,1966年生,博士,教授,博士生导师,中国计算机学会(CCF)高级会员,主要研究领域为编译技术、软件工程等.E-mail:shjjiang@cumt.edu.cn.**张 妙**,女,1992年生,硕士,主要研究方向为软件分析与测试等.**鞠小林**,男,1976年生,博士,副教授,主要研究方向为软件分析与测试、软件故障定位等.

breaking method and the measurement of the test stub cost. For the cycle-breaking method, it mainly adopts two kinds of methods: one is the method based on graph theory and another one is the method based on search. Where, the method based on graph theory and can be divided into two types: (1) only the inter-class static dependencies are considered, breaking inheritance, aggregation, association and use relations directly to eliminate the loops; (2) the inter-class static and dynamic dependencies are considered, breaking inheritance, aggregation, association, use and dynamic dependencies directly to eliminate the loops. The method based on search first determines an initial population which is consisted by class integration test orders, and then under the premise of meeting certain conditions (e. g., minimum the test stub cost), it deals with the population through evolutionary operation, and generates the optimal class integration test order. At present the method based on search only considers the inter-class static dependencies, ignoring the dynamic dependencies. For the measurement of the test stub cost, it includes the number of test stub and the overall complexity of test stub. Thirdly, the research status and the characteristics of these typical techniques completely and systematically are analyzed; where the existed typical techniques for solving the class integration test order determination problem are classified, which include the methods based on graph and the methods based on search and then some evaluation dataset used respectively by typical techniques such as graph-based methods and search-based methods in empirical studies are also introduced. Where, for the typical methods based on the graph techniques, we introduce them from five types such as the methods that minimize the number of general stub (class stub), the methods that minimize the number of special stub, the method of special situation (acyclic and considering dynamic dependencies), the methods that minimize the overall test complexity, and the methods by slicing technical. For the typical methods based on the search techniques, we introduce them from two types such as the solutions by multi-objective optimization algorithms based on linear weighting and the solutions by the optimization algorithms based on Pareto model. Finally, the future research directions of class integration test order determination problem are proposed and the paper is concluded.

Keywords　software testing; class test order; integration test; cycle-breaking; test stub cost

1　引　言

类集成测试顺序的确定是面向对象软件集成测试中的关键难点之一. 不同的测试顺序往往导致不同的测试代价，一个合适的测试顺序可以大大降低测试成本. 因此，我们面临的一个实际问题是准确确定一个最优的类测试顺序，使其满足所花费的测试成本尽可能低. 也就是说，我们需要定义一个指标，用于比较测试顺序，寻找一个可以识别最优测试顺序的算法.

李必信教授等在 2011 年对类集成测试顺序问题进行了系统的总结[1]. 我们以该综述的分析为基础，补充吸收了大量近年来国内外学者取得的最新研究成果，进行了较为全面的分析和梳理. 本文的主要贡献可总结如下：

系统地分析了类集成测试顺序问题近些年来取得的研究成果，借助“CNKI”，“Web of Science”，“IEEE”，“ACM”，“Springer”等数据库，通过关键词对相关问题进行检索. 搜集分析 63 篇在国内外权威期刊和会议等发表的相关文献.

本文第 2 节简单描述类测试顺序确定技术研究问题，并介绍类测试顺序确定问题的两种分类方式；第 3 节对现有解决类集成测试顺序问题的典型技术进行分类，并全面系统地分析这些相关技术，还介绍所使用的评测数据集等；第 4 节指出未来的研究方向；第 5 节最后对本文进行总结.

2　类测试顺序的确定问题描述

2.1　问题陈述

类簇由一组类组成，这组类具有一定的联系，比

如,方法或数据接口就能使它们之间产生某些联系,通常构成复杂的控制依赖、数据依赖等.类簇级测试(也称为类间集成测试)是面向对象测试的四大层次之一[2].它的测试对象是类簇,即针对类簇进行测试.我们知道,面向对象程序一般是在对象间进行消息传递,进而使其具有一定的联系.其中,一条消息从一个对象发送到另外一个对象通常会导致连带作用,即由于消息的传递最终会生成一条方法调用链.这样,表示调用关系的调用链便构成了一个网状结构.面向对象程序中,类集成测试顺序问题指的是为类簇中的各个类确定它们被测试的先后顺序.因此,进行测试的时候,第一个测哪个类,和确定类集成测试顺序的有效方法是需要我们解决的两个关键.

如果类簇之间没有形成环路,逆向拓扑排序的方式便于确定类簇的测试顺序;如果形成环路,则逆向拓扑排序不能满足需要,而是需要我们先打破环路,然后再确定类簇的测试顺序[3].因此,打破环路是解决类测试顺序问题的一个首要关键问题.只有打破环路以后,才能利用逆向拓扑排序等方法进而得到类测试顺序[4].

类测试顺序的确定问题,可以转化为基于图论的方法或者基于搜索的方法.其中,对于基于图论的方法中的图指的是类图.类图是一个有向图,表示类簇及它们之间的关系,可以表示为 $G(V,E)$.其中 V 表示类集合,E 表示类间关系的边集.类集成测试顺序问题指的是通过分析有向图的结构,对类图中用于表示类的所有节点进行排序,最终形成一个类测试序列.

例如,图 1 是不含环路的类图,可以通过逆向拓扑的方式确定测试顺序.测试顺序为(A,B,C).图 2 中 A、B、C 首尾相指构成一个环路,则必须找到首先测试的类(即打破环路 A,B,C),然后通过逆向拓扑来确定所有类的测试顺序.

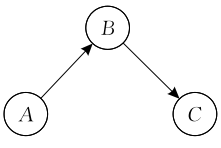

图 1 不含环路的类图

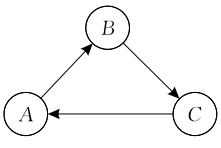

图 2 含有环路的类图

基于图论的类测试顺序确定的基本思想是:首先测试类图中没有任何前驱的类,然后测试该类的后继,直到测试完所有的类,这样测试方式可以降低测试桩的代价,降低测试成本.

基于搜索的类测试顺序确定方法的基本思想是:首先需要确定一个表示类集成测试顺序的初始种群,然后在满足一定条件(如尽可能减少测试桩代价)的前提下,通过进化操作对这一种群进行处理,进而生成类集成测试顺序.

2.2 类间的依赖关系

面向对象程序的类间依赖关系包括静态和动态关系.其中,静态关系是根据静态结构抽取出的用于表达类间依赖关系.而动态依赖关系则是面向对象程序运行时所形成的依赖关系.

2.2.1 类间的静态依赖关系

面向对象程序中的类间静态依赖关系有如下五种:继承(Inheritance)、组合(Composition)、聚合(Aggregation)、关联(Association)和依赖关系(Dependency)[5].

(1)继承关系(Inheritance)

继承是父类和子类之间的一种依赖关系.指子类共享父类(或祖先类)的属性和操作.具体地说,父类定义它子类的公有属性和操作等,而子类除了重新定义父类中操作的实现方法,还可以定义它本身的属性和操作.例如,对于两个类 A 和 B,类 A 继承类 B 时,类 A 是类 B 的子类,类 A 可以继承类 B 的成员(如数据成员和成员方法等).

(2)组合关系(Composition)

类 A 由类 B 组合而成,类 A 包含类 B 的全局对象,并且类 B 对象在类 A 创建的时刻而创建.同时,组合类的构造函数中包括另外一个类的实例化.它是整体与部分的关系,但部分不能离开整体而单独存在.如:实例化大雁类 A 以前,要先实例化翅膀类 B.翅膀类 B 不可以脱离大雁类 A 而独立存在.类 A 和类 B 同生共灭,紧密耦合在一起.

(3)聚合关系(Aggregation)

当对象 A 被加入到对象 B 中,成为对象 B 的组成部分时,对象 B 和对象 A 之间为聚集关系.聚合是关联关系的一种,是较强的关联关系,强调的是整体与部分之间的关系,且部分可以离开整体而单独存在.例如:雁群由大雁聚合而成,当雁群解散时,大雁还可以单独存在.

(4)关联关系(Association)

对于两个相对独立的对象,当一个对象的实例与另一个对象的一些特定实例存在固定的对应关系时,这两个对象之间为关联关系.发生关联关系的两个类,其中的一个类成为另一个类的属性,这时关联

关系的源类将增加属性.关联关系既可以是单向的，也可以是双向的.当关联关系是双向的时侯，两个类的属性均会增加.

(5)依赖关系(Dependency)

依赖关系(Dependency)也称为使用关系(Use).对于两个相对独立的对象，当一个对象负责构造另一个对象的实例，或者依赖另一个对象的服务时，这两个对象之间主要体现为依赖关系.当类 A 使用类 B 时，类 B 可以体现为局部变量、方法参数、静态方法的调用.

根据以上对类间静态依赖关系(如组合、聚合、关联和依赖关系)的定义可以发现，类间的组合、聚集、关联和依赖等静态依赖关系主要通过源类调用目标类成员变量(*MD*)、源类调用目标类方法参数(*PD*)、源类调用目标类方法返回值(*RD*)、源类调用目标类的方法(包括构造方法)(*ID*)来进行判断.

图 3 是示例程序 SimpleHospital[6].它包含 Hospital、Doctor、Patient 这 3 个类.类间的依赖关系如表 1 所示.其中表中的数字表示行号.

```
1. public class Hospital {
2. Doctor physician, surgeon;
3. public Hospital() {
4.    physician=new Doctor(this);
5.    surgeon=new Doctor(this);
6. }
7. void acceptPatient(Patient patient) {
8.   if (patient.isPhysical())
9.      physician.addPatient(patient);
10.   if (patient.isSurgical())
11.     surgeon.addPatient(patient);
12. }}
13. public class Doctor {
14. Hospital hospital;
15. Set<Patient>patients;
16. public Doctor(Hospital hospital) {
17.   hospital=hospital;
18.   patients=new Set<>();
19.}
20. void addPatient(Patient patient) {
21.   patients.add(patient);
22.}}
23. public class Patient {
24. bool b_physical, b_surgical;
25. public Patient(bool b_p, bool b_s) {
26.   b_physical=b_p;
27.   b_surgical=b_s;
28. }
29. boolean isPhysical() {
30.    return b_physical;
31. }
32. boolean isSurgical() {
33.    return b_surgical;
34. }}
```

图 3　示例程序 SimpleHospital

表 1　示例程序的属性依赖和方法依赖

	Hospital	Doctor	Patient
Hospital	—	(*MD*,2),(*MD*,2),(*ID*,9),(*ID*,11)	(*PD*,7),(*ID*,8),(*ID*,10)
Doctor	(*MD*,14),(*PD*,16)	—	(*PD*,20)
Patient	—	—	—

由表 1 可以发现，类 Hospital 包含 3 个属性依赖、4 个方法依赖，共 7 个依赖关系，类 Doctor 包含 3 个属性依赖关系，类 Patient 无依赖关系.

根据类间静态依赖关系的定义，我们可以得到图 3 所示示例程序的类间静态依赖关系，如表 2 所示.

表 2　示例程序的类间静态依赖关系

	Hospital	Doctor	Patient
Hospital	—	关联	关联
Doctor	聚集	—	关联
Patient	—	—	—

由表 2 可以发现，类 Hospital 和类 Doctor 是关联关系，类 Hospital 和类 Patient 是关联关系，类 Doctor 和类 Hospital 是聚集关系，类 Doctor 和类 Patient 是关联关系.

对于基于图论的方法，类簇及类间的关系一般抽象为对象关系图(ORD)[1].ORD 使用有向图 $G(V,E)$ 表示.其中，V 由用于表示类的节点构成，E 由用于表示类间关系(继承关系 I、组合关系 Cp、聚集关系 Ag、关联关系 As 和使用关系 Us)的边构成.

可以发现，ORD 体现的是类间的静态依赖关系.静态依赖关系的定义如下.

定义 1[4,7].　静态依赖集合.对于任意一个类图或 ORD 中的两个类 A 和 B，类 A 静态依赖的集合包括类 B 以及类 B 依赖的源类，即包含传递依赖的类，当且仅当 ORD 存在一条从 A 到 B 的有向边.

图 4 是一个 ORD 的示例.图中的节点代表类，实心有向边代表类间的继承关系、聚集关系和关联关系等.对于类 $C1$ 到 $C2$ 的有向边：

(1)“I”边代表 $C1$ 是 $C2$ 的子类；

(2)“Ag”边代表 $C1$ 是 $C2$ 的一个聚集类($C1$ 包含一个或多个 $C2$ 对象)；

(3)“As”边代表 $C1$ 与 $C2$ 关联.

UML 类图是表示类间关系的一种常用的模型.对于类集成测试顺序的确定问题，为了方便，通常的方法一般是将 UML 类图转换为对象关系图.这样就需要对类间关系进行映射.文献[8]提出了一种从 UML 类间关系映射为类测试顺序问题中用到的继承关系、聚集关系和关联关系的方法.其中：

①将 UML 类图中的泛化映射到对象关系图中的继承关系,表示为“I”;②组合是关联关系的一种特例,但是它体现的关系强于聚合关系,属于强聚集.构成组合关系的整体与部分是不可分的,整体生命周期一旦结束,那么部分的生命周期也不会继续存在,而是会随之结束,表示为“Ag”.总体来说,聚合和组合都表示为“Ag”;③将 UML 类图中的关联关系映射到对象关系图中的关联关系,表示为“As”;另外,UML 类图中一个简单的聚合关系是一种特殊的关联,也表示为“As”.总体来说,关联、简单的聚合和依赖都表示为“As”.

为了进一步理解类间关系的概念,我们通过图 3 所示的示例程序 SimpleHospital 进行说明.该示例程序对应的 ORD 图如图 4 所示.该 ORD 图包含类 Hospital、Doctor 和 Hospital 这 3 个类和 4 个类间关系.其中,类 Hospital 和类 Doctor 是关联关系(As),类 Hospital 和类 Patient 是关联关系(As),类 Doctor 和类 Hospital 是聚集关系(Ag),类 Doctor 和类 Patient 是关联关系(As).

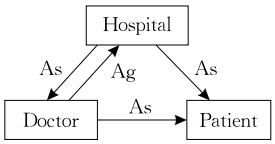

图 4 ORD 示例

定义 2. 测试桩[9].对被测试模块或组件在工作时依赖的模块或组件的模拟称为测试桩,也称测试存根.

例如,假设一个系统有 64 个类,这些类的 16 个已经集成,在其余 48 个中,假设类 $C1$ 依赖于类 $C2$,即 $C2$ 被集成后,$C1$ 才可以使用 $C2$.如果 $C1$ 在 $C2$ 之前集成,则需要构造一个用于模拟 $C2$ 行为的测试桩.

根据测试桩的定义我们可以推断,测试桩代价与静态依赖的联系强度有着直接关系.当两个类之间是强依赖关系,如果打破环路时删除该强依赖关系,则需要为构成该强依赖关系的服务类构造测试桩,也就是需要模拟该服务类的对象行为,而模拟该服务类的对象行为需要理解所有与之相关的类的对象行为,因此,类间依赖关系越强,所需构造的测试桩越复杂.那么,为了避免构造复杂的测试桩,一般只选择删除弱依赖关系,即关联关系或使用关系.在该限制条件的前提下,遵循保证测试桩代价尽可能低原则的情况下,选择删除合适的关联或使用等弱依赖关系.

2.2.2 类间的动态依赖关系

多态性是面向对象程序的一个重要特性,忽略这一特性得到的测试顺序是不准确的.因此,需要考虑多态性对打破环路的影响,更确切地说,需要考虑对最终生成的类集成测试顺序的影响.

在程序静态阶段(编码阶段),并不能确定定义的变量的方法调用和指向的具体类型,只有到程序运行期间才能确定.因此,在绑定引用变量到不同的类实现中时,不用修改源代码.同时,不改源代码的前提下,引用调用的具体方法和所绑定的具体内容都会有所改变,程序便能够选择多个运行状态,这体现的就是多态性.

动态依赖关系是受多态性的影响,在程序运行期间形成的类间依赖关系.

定义 3[8]. 动态依赖关系.如果类 A 继承类 B,且重写类 B 的虚方法,同时,类 B 是类 C 的服务类,即类 C 关联类 B 或者类 C 是 B 的聚集类,且调用了类 B 中已被类 A 重写的虚方法,那么在程序运行期间,类 C 动态依赖于类 A.如果类 A 继承类 B,类 B 是类 C 的服务类,即 C 或者和 B 相关联或者是 B 的聚集类,那么在程序运行期间,类 C 可能会动态依赖于类 A[8].

类间静态关系可以推导出动态关系.以图 5 所示示例为例,教师 teacher 类关联学生 student 类,student 类是本科生 undergraduate 类的父类,在程序执行时,teacher 类可能动态依赖 undergraduate 类.同理,成绩 score 类也可能动态依赖 undergraduate 类.类间动态依赖关系用标有 Dy 的虚线边表示.

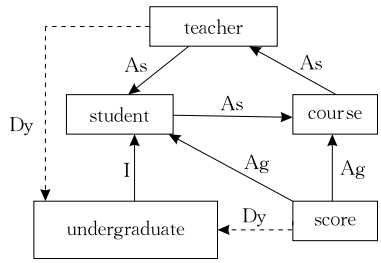

图 5 包含动态依赖关系的 ORD(EORD)示例

Kung 等人[3],Briand 等人[10],Kraft 和 Lloyd 等人[11]认为构建测试桩的难易程度(代价 $Cost$)存在以下关系:$Cost$(继承关系)≫$Cost$(聚集关系)>$Cost$(关联关系)=$Cost$(动态依赖关系).因此,存在继承或聚集关系的两个类的依赖程度较强,而存在动态依赖或关联关系的两个类的依赖程度较弱.

对于考虑动态依赖关系的类集成测试顺序问题,在打破环路时需要考虑动态依赖关系与测试桩

代价之间的关系.同 2.2.1 节的分析,为了避免构造复杂的测试桩,一般选择删除弱依赖关系.这里,弱依赖关系包括动态依赖关系与关联关系.因此,在该限制条件的前提下,遵循保证测试桩代价尽可能低的原则的情况下,选择删除合适的动态依赖关系与关联关系.

2.2.3　抽象类

通常在编程语句中用 abstract 修饰的类是抽象类.C++中,带有纯虚拟函数的类是抽象类,不能生成对象;Java 中,带有抽象方法的类是抽象类,也不能生成对象.

一个类测试顺序中包含的抽象类有时会同时承担父类和服务类的角色,而面向对象中抽象类的一个典型特点就是其本身不能进行实例化,而实例化是由它的子类所完成.那么,可以发现该抽象类就不再具有服务类的作用,而只能作为一个父类.此时是由该抽象类的子类通过实例化来完成该抽象类的服务作用.因此,抽象类不可实例化的特点对确定类测试顺序有一定影响,忽略抽象类不可实例化的特点得到的类测试顺序是不准确的.

因此,在打破环路的过程中,就需要考虑抽象类不可实例化的特点对打破环路的影响.其次,在消除环路之后,抽象类不可实例化的特点使得其不能被独立测试,导致部分测试序失效[8],这就要求我们对该无效测试序进行调整,使其恢复有效,进一步提高测试顺序的准确度.

2.3　问题分类

类集成测试顺序问题分类方式有多种,本节从破除环路的方法和衡量测试桩代价的方法这两个角度进行分类.

2.3.1　破除环路的方法分类

对于打破环路的方法,主要采用两类方法进行:一种是基于图论的方法.基于图论的方法又可以分为两种类型:(1)只考虑类间静态依赖关系的情况,打破环路时直接断开图中继承、聚集、关联、使用关系;(2)同时考虑类间的静态依赖关系和动态依赖关系,打破环路时直接断开图中继承、聚集、关联、使用和动态依赖关系;第二种是基于搜索的方法,即首先确定一个表示类集成测试顺序的初始种群,然后在满足一定条件(如尽可能减少测试桩代价)的前提下,通过进化操作对种群进行处理,从而生成最佳的类集成测试顺序.目前基于搜索的这类解决方案考虑的只有静态关系,忽略了动态关系.类间关系构成的环路打破方法分类如图 6 所示.

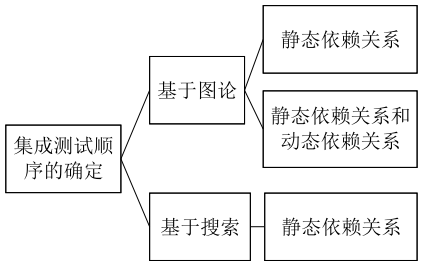

图 6　打破环路的方法分类

2.3.2　衡量测试桩代价的评测指标

已有文献主要通过判断生成一个类集成测试顺序所花费的代价的高低来评价一个类测试顺序确定方法的好坏,而生成一个类集成测试顺序需要打破依赖环路.打破依赖环路的过程就是从有向图中逐步去除那些形成环的有向边的过程,即消除类之间的依赖关系,直至图中不存在环为止[3].这一过程意味着需要开发测试桩.即删除依赖关系时,需要构建一定的测试桩.由测试桩的定义可知,测试桩模拟的是目标类所依赖的源类的相关模块的内容.该相关模块往往描述一个类对象的行为,而一个类对象行为又往往与多个类对象(或模块)有关,因此,在构造测试桩的时候需要花费较高的成本来模拟所有这些相关的类对象行为(或模块).

通过以上分析可知,生成一个类测试顺序所花费的代价高低可以通过测试桩代价来进行衡量.那么,在消除所有环路时,应该满足构造测试桩所花费的代价尽可能小.由于这样的代价往往不能直接被度量或估计,我们需要借助于一定指标.

评价测试桩代价高低的指标,存在一个一般的指标[9,12]:其一是集成测试过程当中需要构造的测试桩的总数目;其二是构造测试桩的总体复杂度.

(1)构造的测试桩数目

解决类集成测试顺序问题时需要降低测试桩数目,以达到降低测试开销的目的.因为创建测试桩是生成一个类测试顺序所花费的主要开销,体现在如下方面:

测试桩需要根据功能需求,对目标类所需的服务类进行功能模拟.创建测试桩的过程可能比被模拟的源代码复杂,测试桩也可能不完全正确,错误率可能高于源代码;此外,由于测试桩的创建需要测试人员对代码的理解,因此不太可能完全实现自动化生成测试桩.

测试桩又分为一般的测试桩(也称为“类桩”)和特效桩,通常情况下我们都简称为测试桩.事实上,几个客户类使用一个服务类时,构造的测试桩的数

目至少等同于客户类的数目.需要构造测试桩的类的数目乘以其客户类的数目是一个更准确的评价标准,称之为特效桩的数目[11].例如:有三个类类 A、类 B 和类 C.其中,类 C 有两个方法 $p()$ 和 $q()$,类 A 调用类 C 的方法 $p()$,类 B 调用类 C 的方法 $q()$.当需要测试类 A 和类 B 的时候,类 C 尚未被测试,此时应该为类 C 构建测试桩.如果是一般的测试桩,只需要为类 C 构造一个测试桩,类 A 和类 B 可以共用;而如果是特效桩,则需要分别为类 A 和类 B 所调用的不同方法 $p()$ 和 $q()$ 构建测试桩,即需要构建两个特效桩.

(2)构造的测试桩总体复杂度

影响测试桩复杂度的因素比较多,包括属性复杂度 $A(i,j)$,方法复杂度 $M(i,j)$,返回值复杂度 $R(i,j)$ 和传递的参数复杂度 $P(i,j)$ 等[5,13].通常采用计算属性复杂度、方法复杂度、返回值复杂度和传递的参数复杂度的几何平均值来定义一个测试桩复杂度的计算公式,用于衡量测试桩复杂度.其中,属性复杂度 $A(i,j)$ 在测试桩复杂度的计算公式中的权重用 W_A 表示,方法复杂度 $M(i,j)$ 的权重用 W_M 表示,返回值复杂度 $R(i,j)$ 的权重用 W_R 表示,传递的参数复杂度 $P(i,j)$ 的权重用 W_P 表示.W_A、W_M、W_R 和 W_P 不同的值决定属性复杂度,方法复杂度,返回值复杂度和传递的参数复杂度在测试桩复杂度的计算公式中所占的比重.若类 i 和类 j 之间方法的复杂度 $M(i,j)$ 权重 W_M 高于属性的复杂度 $A(i,j)$ 的权重 W_A,在类 i 和类 j 之间测试桩复杂度上,对于成本函数 $M(i,j)$ 的影响高于 $A(i,j)$.

为了处理存在的上述问题,将 $A()$ 和 $M()$ 定义的测试桩复杂度的计算公式进行标准化[9].例如依据数学方法,对 $Cplx()$ 进行标准化,记为 $\overline{Cplx()}$.矩阵 $Cplx(i,j)$ 表示类 i 和类 j 构成的边的复杂度,行和列都是类,类 i 依赖于类 j,需要计算 $Cplx_{\min}=\mathrm{Min}\{Cplx(i,j),i,j=1,2,\cdots\}$,$Cplx_{\max}=\mathrm{Max}\{Cplx(i,j),i,j=1,2,\cdots\}$,则

$$\overline{Cplx()}=\frac{Cplx(i,j)}{Cplx_{\max}-Cplx_{\min}} \tag{1}$$

矩阵 $Cplx(i,j)$ 中最小的复杂度的值等于 0.式(1)可以写为

$$\overline{Cplx()}=\frac{Cplx(i,j)}{Cplx_{\max}} \tag{2}$$

Briand 等人[9]认为属性复杂度和方法复杂度足以衡量测试桩复杂度.因此,在他们的方法中,对于存在依赖关系的两个类 i,j,如果需要构造一个测试桩,则测试桩复杂度 $SCplx(i,j)$ 通过标准化的加权几何平均计算,如式(3)所示:

$$SCplx(i,j)=(W_A\times\overline{A(i,j)^2}+W_M\times\overline{M(i,j)^2})^{1/2} \tag{3}$$

其中 $W_A+W_M=1$;W_A、W_M 为属性复杂度、方法复杂度的权重,取值在[0,1]之间.

为了更精确的度量测试桩复杂度,一些研究[5]还考虑了影响测试桩复杂度更多的因素,如返回类型复杂度和传递的参数复杂度.此时,测试桩复杂度 $SCplx(i,j)$ 的计算方法如式(4)所示:

$$SCplx(i,j)=(W_V\times\overline{V}(i,j)^2+W_M\times\overline{M}(i,j)^2+W_R\times\overline{R}(i,j)^2+W_P\times\overline{P}(i,j)^2)^{1/2} \tag{4}$$

其中 $W_A+W_M+W_R+W_P=1$;W_A、W_M、W_R、W_P 为属性复杂度、方法复杂度、返回值的复杂度和传递的参数复杂度的权重,取值在[0,1]之间.

对于一个给定的测试顺序 o,则测试顺序的整体复杂度为[5,9,13]

$$Ocplx(o)=\sum_{k=1}^{d}SCplx(k) \tag{5}$$

式(5)中,d 代表类集成测试顺序中类的个数,k 代表类.$SCplx(k)$ 是对于一个测试顺序中,单个一个类的测试桩复杂度.对于一个类集成测试顺序,$OCplx(o)$ 为生成一个测试顺序 o 总共需要构造的相应测试桩的复杂度,即为求得类集成测试顺序所花费的总体测试桩的代价.

由于构造不同的测试桩往往需要不同的测试代价,因此,构造的测试桩的数量越少,不能表明确定一个类测试顺序所花费的总体测试桩复杂度越低.测试桩的数量和总体复杂度分别用于度量构造测试桩难易度的指标时,后者更准确.

3 研究现状分析

本节分析类集成测试顺序问题的研究现状.介绍一些已有的典型方法,这些相关工作的分类如表 3 所示.该表包括分类(Cla.)、序号(No.)、文献(Ref.)、发表年份(Year)、文章类型(期刊/会议)、目标(Object)、采用的模型(Model)和算法(Algorithm).其中,模型包括类型(Type),类间关系的类型(Edge Types)、类间关系的来源(Origin);算法包括类型(Type)、允许删除的边的类型(Deleted edge)以及是否考虑了抽象类(Consider abstract class).而类间关系的类型(Edge Types)包括继承(I)、聚集(Ag)、

表 3　主要相关工作的分类

Cla.	No.	Ref.	Year	Journal(J)/Conference(C)	Object	Model			Algorithm		
						Type	Edge Types	Origin	Type	Deleted edge	Consider abstract class
GBA	1	Kung[3]	1995	JOOP(J)	MNGS	ORD	I, Ag, As	Code	—	As	N
		Tai[14]	1997	COMPSAC(C)				UML			
		Tran[15]	2000	ITR(J)							
	2	Hanh[16]	2001	ECOOP(C)	MNGS	TDG	I, Ag, As	UML	—	No constraint	N
		Hewett[17]	2008	SEKE(C)							
		Hewett[18]	2009	ASE(C)							
		Jaroenpiboonkit[19]	2007	APSEC(C)							
	3	Briand[10]	2001	ISSRE(C)	MNSS	ORD	I, Co, As	UML	—	As	N
		Briand[20]	2003	TSE(C)			I, Co, As	Code			
		Mao[2,21]	2005	CIT(C)			I, Ag, As	UML			
		Hashim[22]	2005	QSIC(C)			I, Co, Ag, As, De	UML		No constraint	
	4	Malloy[23]	2003	ISSRE(C)	MNSS	EORD	I, Co, As, De, Po, Ow	Code	—	No constraint	Y
		Kraft[11]	2006	JSS(J)			I, Co, As, De, Po, Ow	Code			N
		李都[24]	2008	CED(J)			I, Co, Ag, As, De, Po	UML			N
		Bansal[25]	2009	ICM2CS(C)			I, Co, As, De, Po, Ow, Fr, Ex	Code			N
		张艳梅[8]	2011	计算机学报(J)			I, Ag, As, Po	UML		As, Po	Y
		Zhang[26]	2011	SPE(J)							
	5	Labiche[27]	2000	ICSE(C)	MNSS	EORD	I, Ag, As, Po	UML	—	—	Y
		Li[7]	2005	JAECT(J)							
	6	Abdurazik[13]	2009	TCJ(J)	MOSC	WORD	I, Im, Co, Ag, As, De	Code	—	No constraint	N
		姜淑娟[5]	2011	计算机学报(J)		TDG	I, Co, As			As	
		赵玉丽[28]	2015	东北大学学报(J)							
		王莹[29]	2016	计算机研究与发展(J)		network	—			No constraint	
	7	刘颖莲[30]	2013	北京邮电大学(D)	—	DIGRAPH	—	UML	SLICING	No constraint	N
GSA	1	Hanh[16]	2001	ECOOP(C)	MOSC	TDG	I, Co, As	Code	GA	No constraint	N
		Briand[9]	2002	Technical report		ORD				As	
		Borner[31]	2009	VALID(C)			I, As, De			No constraint	
		Borner[31]	2009	VALID(C)			I, Ad, De		SA	No constraint	
		张艳梅[6]	2016	计算机学报(J)		—	I, Co, Ag, As		PSO	As	
	2	Cabral[32]	2010	ICTSS(C)	MOSC	ORD	I, Co, As	Code	PACO	As	N
		Vergilio[33]	2012	STTT(J)			I, Co, As		NSGAII, MTabu	As	
		Assunção[34]	2011	GECCO(C)			I, Co, Ag, As		NSGAII, SPEA2	Co, Ag, As	
		Assunção[35]	2014	IS(J)			I, Co, Ag, As		NSGAII, SPEA2, PAES	Co, Ag, As	
其它	1	Wang[36]	2010	CSACW(C)	MOSC	EWORD	I, Im, Co, Ag, As, De	Code	RIA	No constraint	N
	2	Guizzo[37]	2015	GECCO(C)		—	—		HHA		

关联(As)、组合(Co)、依赖(De)、多态(Po)、单元(Ow)、友元(Fr)、异常(Ex)、实现(Im).

针对类测试顺序的确定这一问题，目前国内外的学术界已经提出了一系列相关技术.我们将这些典型技术分为三类，分别是：基于图论的方法(GBA)，基于搜索的方法(GSA)和其它(Other).

对于基于图论的方法，我们又将其分为七种类型，其依据是：首先根据衡量测试成本的标准进行分类，即判断其是最小化测试桩的个数(包括一般测试桩(类桩)的个数(MNGS)和特效桩的个数(MNSS))还是最小化总体测试桩的复杂度(MOSC)；其次，根据描述类间关系的模型进行分类，即对象关系图

(ORD)(只考虑静态依赖关系)、扩展的对象关系图(EORD)(增加动态依赖关系)、加权对象关系图(WORD)、测试依赖图(TDG)、网络(network)和有向图(DIGRAPH)等.对于模型中类间关系的来源,主要包括两种方式:其中一部分可以通过 UML 获得,一部分是从源代码中所提取.

对于基于搜索的方法,大多数的方法是以构造的测试桩的总体复杂度作为花费的测试成本,因此,与基于图论的分类方式不同,我们不根据衡量测试成本的标准进行分类,而是根据采用的算法的特点进行分类,包括两种类型:分别是基于线性加权的多目标优化方法和基于帕累托模型的方法.其中,前者包括基于遗传算法的方法(GA)、模拟退火算法(SA)和粒子群算法(PSO).后者包括蚁群算法(PACO)、禁忌搜索算法(MTabu)和 NSGA-II、SPEA2、PAES 算法.

对于其它方法,主要是随机交互算法(RIA)和超启发式算法算法(HHA).

3.1 基于图论的方法

基于图论的解决方案中,类以及类之间的关系除了用类图或 ORD 表示外,还可以用测试依赖图(Test Dependency Graphs,TDG)来表示[3].TDG 也是有向图,图中的节点代表类,有向边代表类之间的依赖关系,通常用 $G(V,E)$来表示,其中 V 代表节点的集合,E 代表有向边的集合.本节中,根据衡量测试桩代价的两个评测指标,我们对基于图论的方法进行分类介绍.

根据 2.3.1 节的描述,我们知道基于图论的方法又分为两种情况:一是只考虑类间的静态依赖关系,二是同时考虑类间的静态依赖关系和动态依赖关系.其中,对于只考虑类间静态依赖的情况,确定类集成测试顺序的基本思想是:首先,将类间关系抽象成类图、对象关系图或者测试依赖图;然后,识别出图中的强连通组件,并打破其中的环路,在打破环路的过程中的限制条件是遵循满足所花费的测试桩代价尽可能小;最后,利用逆向拓扑排序等方法对所有类进行排序,进而得到最终的类集成测试顺序.流程图如图 7 所示.同时考虑类间静态依赖和动态依赖关系时,确定类集成测试顺序的基本思想是:首先,在类图、对象关系图或者测试依赖图等模型的基础上,增加动态依赖关系,构成同时包含静态和动态依赖关系的扩展的对象关系图;然后,识别出图中的强连通组件,并打破其中的环路,在打破环路的过程中的限制条件同样是遵循满足所花费的测试桩代价尽可能小;最后,利用基于测试级的方法对所有类进行排序,进而得到最终的类集成测试顺序.流程图如图 8 所示.

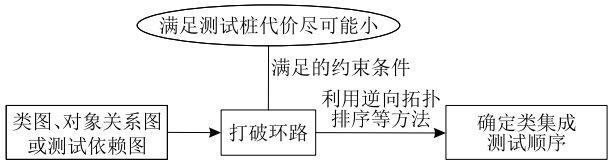

图 7 基于图论的类测试顺序生成流程图(静态依赖关系)

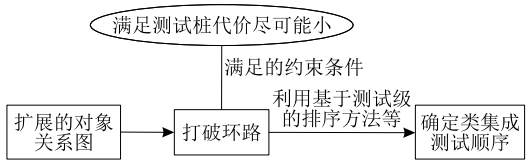

图 8 基于图论的类测试顺序生成流程图
(静态依赖关系和动态依赖关系)

3.1.1 最小化一般桩(类桩)的数量(MNGS)

以最小化一般桩(类桩)的数量作为衡量测试代价的方法主要考虑的是类间的静态依赖关系,分为两类:孩只考虑静态依赖关系(以 ORD 作为模型);虎只考虑静态依赖关系(以 TDG 作为模型).

3.1.1.1 静态依赖关系(以 ORD 作为模型)

以 ORD 作为模型,只考虑静态依赖关系时,具有代表性的分别是 Kung 等人[3]、Tai 和 Daniels[14]、Le Traon 等人[15]的算法.

(1) Kung 算法

Kung 等人[3]指出,类间关系构成的图中如果没有环路,可以通过对类簇进行简单的逆向拓扑排序的方法得到类测试顺序——一个众所周知的图论问题[38];Kung 等人[3]证明关联关系通常是类图中最弱的依赖关系.他们还认为,类图中的每个环路至少包含一个关联关系.因此,为了消除环路并且构造尽可能简单的测试桩,应打破关联关系.

Kung 等人[3]在确定类测试顺序时,首先将 ORD 中的类分为若干个簇,用单个节点代替有环 ORD 中的簇,得到一个无环 ORD,再对替换后的 ORD 应用拓扑排序获得每个簇的主测试顺序.簇指 ORD 中一组互相到达的类集合.分单元簇和非单元簇.其中,单元簇仅包含一个类;非单元簇包含多个类结点,可以通过去除一条或多条边的方式成为一个无环子图.非单元簇中的类测试顺序可以通过拓扑排序子图生成.而主测试顺序和所有非单元簇中的类测试顺序一起组成 ORD 图中所有类的测试顺序.Kung 等人[3]证明一个有环的 ORD 图中至少有一条关联边,因此,他们建议删除关联边来破除环路.

虽然 Kung 等人[3]给出了一种类测试顺序的确

定方法，但他们的这一方法存在以下问题：① 对于包含依赖环路的类图，他们没有提供精确的解决方案. 实际程序中一般都包含大量环路. 这时，不适合采用逆向拓扑的方法获得类测试顺序，而是需要先识别出其中的强联通分量，然后通过删除部分依赖关系来打破这些环路，从而使之成为无环图；② 当两个以上的依赖边具有相同的权重时，采用随机选择的策略，导致对不同删除边的选择会导致较大差异的结果.

(2) Tai 和 Daniels 算法

针对上述缺陷，Tai 和 Daniels[14] 提出了一种算法. 首先，只分析继承和聚集两种类型的强依赖边，这两种类型的依赖边构成有向无环图. 从该无环图中很容易获得类间的主要层次，即主要顺序；其次，属于同一主顺序的关联关系可能构成强连通组件 SCCs，这就需要断开某些关联边来确定次要顺序. 综上，所有类的集成测试顺序的确定方法是首先按主要顺序为子集确定次要顺序，然后在子集内部进行排序，从而生成整个类的集成测试顺序.

在确定次要顺序时，如何确定需要删除哪些关联边，我们可以遵循以下方案[39]：将构成环路的任何一条关联边，简写成 $e=\langle P,Q\rangle$，其权重定义为强连通组件 SCCs 当中的任何一条边(e)的源点入度(记为 indegree(P))与 e 的目标顶点的出度(即 outdegree(Q))之和：

$$\text{weight}(e)=\text{indegree}(P)+\text{outdegree}(Q) \quad (6)$$

再根据式(6)计算的结果，优先选择去掉权值大的边.

该算法的优点是在打破环路时继承和聚集两种类型的强依赖边不会被删除，但是缺点是可能导致一些没有必要的关联边被删除. 虽然绝大多数情况下，从一个稍小的主要层次指向一个稍大的主要层次的关联边并不会出现环路，但是这种情况下如果采用该排序策略就需要为其目标类构造测试桩，从而导致构造不必要的测试桩. 此外，该算法结果存在不确定性，受被删除边的选择所影响.

(3) Le Traon 算法

Tarjan[40] 提出了一种基于深度优先搜索(DFS)的强连通组件(SCCs)的计算算法. 针对 Kung 等人[3] 和 Tai 和 Daniels[14] 这两种算法的缺陷，Le Traon等人[15] 对 Tarjan 算法[40] 进行了改进，提出了一种类间测试顺序确定方法. 该算法选择要删除的依赖边按如下方式进行：在对 ORD 图进行深度优先搜索的时候，首先按照从小到大的顺序标识图中的各个结点，其中，标识号大的节点指向小的边称为逆向边. 对于任意一个强连通组件 SCC，打破其中的环路时，权重大的节点的入边被删除. 因此，节点的权重的计算方法在打破环路过程中起着关键作用.

Le Traon 等人[15] 的算法表明在打破环路时可以删除继承和聚集依赖关系，但是由于它们是强依赖关系，因此该算法会导致构造复杂的测试桩；此外，还存在两个层面上的不确定性：① 将类顶点作为考虑对象，按照深度遍历的话，起点的选取对构造测试桩所花费的代价影响较大；② 如果存在两个或更多个类间关系权重相等时，打破哪个依赖关系对计算结果也有较大影响.

以图 9 所示的 ORD 为例，采用 Kung 等人[3] 的算法、Tai 和 Daniels[14] 和 Le Traon 等人[15] 的算法所需构造测试桩的情况如表 4 所示. 可以发现，Kung[3] 的算法、Tai 和 Daniels[14] 的算法和 Le Traon 等人[15] 的算法以需要构造的类桩数量为打破环路的优化目标.

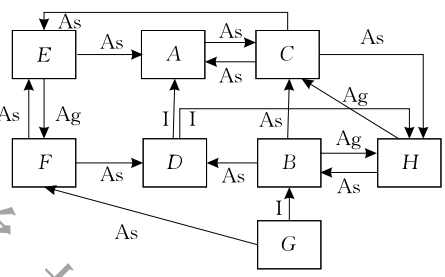

图 9　一个 ORD 示例

表 4　三种算法所需构造测试桩的情况

	Kung 算法	Tai 算法	Traon 算法			
			根节点是 H 节点			根节点是 G 节点
			选择节点 A	选择节点 E	选择节点 F	
类桩	CFHDB	CFHDB	HAF	HEA	HFA	BCF
类桩数目	5	5	3	3	3	3
特效桩	CFHDB	CFHDB	3H3A1F	3H2E1A	3H1F2A	1B2C1F
特效桩数目	5	5	7	6	6	4

以所需创建的类桩数目作为优化目标时，采用 Le Traon 等人[15] 的算法比采用 Kung 等人[3] 的算法以及 Tai 和 Daniels[14] 的算法有所减少；Le Traon 等人[15] 把类顶点作为考虑对象，计算结果依赖于所选的深度优先搜索的起始类顶点.

(4) 其它算法

后来，国内的东华大学李远杰[41] 对 Tai 和 Daniels[14] 算法和 Le Traon 等人[15] 算法进行了改进，提出了一种基于 ORD 的类集成测试顺序的改进算法. 该算法仅允许删除关联边. 与 Tai 和

Daniels[14]算法和 Le Traon 等人[15]的算法相比,需要的测试桩数目较少.

3.1.1.2 静态依赖关系(以 TDG 作为模型)

以 TDG 作为模型,只考虑静态依赖关系时,目前的类测试顺序确定算法中,具有代表性的分别是由 Hanh 等人[16]、Hewett 等人[17]、Hewett 等人[18]和 Jaroenpiboonkit 等人[19]提出的算法.

与文献[3,14]和文献[15]中典型的类测试顺序分配策略不同,Hanh 等人[16]将 TDG 作为一个依赖模型.他们提供了一种基于图论的算法 Triskell.通过选择和删除权重最高的节点打破环路,其中权重定义为节点所涉及到的环路的个数.

Hewett 等人[17]使用由 UML 类图创建的 TDG 作为依赖模型,使用一个快速算法来找到一个最优的测试序,满足一般的测试桩最少.后来,Hewett 等人[18]提出了一种基于软件组件测试依赖图的启发式算法自动生成软件组件测试序,通过最小化测试桩的数目找到一个最优类测试顺序.实验表明通过添加额外的启发式信息的方式改进了他们已有的算法,改进后的方法不使用依赖类型信息,与 Le Traon 等人[15]的方法相比,所需测试桩数目均最少,而 Hewett 等人[18]方法所花费的时间更少,显示了时间性能方面的优势.

Jaroenpiboonkit 等人[19]用 TDG 表示类间关系,用面向对象切片技术打破环路依赖,进而找到一个类测试顺序.他们提出了一种使用 TDG 和面向对象切片技术的类测试序的确定方法.该策略通过对类进行切片,通过部分测试代替删除测试桩构造关系来打破环路,花费了更少的测试桩.

3.1.2 最小化特效桩的数量(MNSS)

以特效桩的数量作为衡量测试代价的方法主要考虑的是类间的静态依赖关系,分为两类:孩只考虑静态依赖关系(以 ORD 作为模型);虎同时考虑静态依赖关系与动态依赖关系(以 EORD 作为模型).

3.1.2.1 静态依赖关系(以 ORD 作为模型)

针对 Le Traon 等人[15]的算法,Briand 等人[20]对它进行了一些改进:(1)避免 Le Traon 等人[15]的算法的第一个不确定性:忽略 SCCs 中各个节点的深度优先搜索标记顺序,并且打破环路时不将赖逆向边的数目作为依赖指标,致使 SCCs 的计算方法不影响权重的计算;(2)只将关联关系打破,并给出了一种新的计算关联边权重的方法:为了构造少量的测试桩,涉及环路较多的关联边被优先删除,其计算方法如式(7):对于某一个特定的 SCC 中的关联边 $e=\langle P,Q\rangle$:

$$\text{weight}(e)=\text{indegree}(P)\times\text{outdegree}(Q) \quad (7)$$

由于 Briand 的方法[20]将关联边作为对象且仅允许删除关联边,因此 Briand 的方法[20]不会删除诸如继承、聚集等强依赖关系,只需构造相对简单的测试桩.该方法虽然没有消除第二个不确定性,但是比 Le Traon 等人[15]的方法计算得到的权重更能反映真实值.

仍以图 9 所示的 ORD 为例,Briand 算法[20]所需构造测试桩的情况如下:类桩是 CFHB,即类桩数目是 4,特效桩是 1C1F1H1B,即特效桩的数目也是 4.可以发现,Kung 等人[3]的算法、Tai 和 Daniels[14]的算法和 Le Traon 等人[15]的算法以需要构造的类桩数量为打破环路的优化目标,而 Briand 等人[20]是以需要构造的特效桩数目为打破环路的优化目标.将所需构造的特效桩的数目作为优化目标时,Briand等人[20]的算法最优,在确定性上也有一定程度的提高.相对于类桩数量为优化目标,这种以特效桩数量为优化目标的方法在实际的类集成测试中更为合理.

国内,江西财经大学毛澄映教授[2,21]对 Tai 和 Daniels[14]算法以及 Briand 等人[20]的算法进行了改进,使用由 UML 类图创建的加权对象关系图(WORD)作为依赖模型,有三种类型的依赖关系:继承、聚合和关联.他们使用一个三元组(环路权重 CW、边向因子 DG、关联强度 RD)表示每个关联关系的权重.按照 CW、DG、RD 的顺序进行剪边操作.剪边策略如下:首先考虑剪去 CW 和 DG 值大的边,其次考虑减去 RD 值较小的边.如果三个值一样(虽然可能性很小),则剪去其中任意一条边.他们定义组成环的边的个数为 2 的环路为一个环偶对.他们首先选择和删除关联边的方法打破所有环偶对,然后使用基于图论的方法打破其它类型的环路.他们的算法更倾向于选择和删除权重最高的关联边打破环路.与 Tai 和 Daniels[14]算法比较,所需要的测试桩数目较少,并能保证不删除继承、聚集等强联系边.与 Briand 等人[20]的算法相比,他们所需创建的测试桩的数目相同,但是毛澄映教授[2,21]的算法求强连通组件 SCCs 的递归次数大为减少.

Hashim 等人[22]借鉴 Malloy 等人[23]的算法,提出了一种基于软件组件的测试顺序生成方法,通过最小化特效桩的数目找到一个最佳的类测试顺序.实验结果表明与 Malloy 等人[23]和 Briand 等人[20]的方法相比,所需测试桩数目均有明显减少.

国内，西安电子科技大学的高海昌教授[42]在UML类图的基础上，对传统的对象关系图进行了扩展，提出了类集成测试顺序改进算法，与Tai和Daniels[14]算法和Le Traon等人[15]的算法比较，需要的测试桩最少，与Briand等人[20]的算法比较，所需创建的测试桩的数目相同. 上海师范大学齐丽娜[43]提出了基于对象模式关系图的类测试顺序生成方法，她考虑了可能存在间接依赖关系的模块，提出了相对完善的测试顺序的确定方案，并构建了确定类集成测试顺序的排序树. 在集成测试时，采用该方法生成的类测试顺序能够有效减少所构造的测试桩数量，可以降低测试所花费的代价.

以上方法都没有考虑多态性的特点对类集成测试顺序的影响，具有一定的局限.

3.1.2.2　静态依赖关系与动态依赖关系(以EORD作为模型)

3.1.2.1节介绍的方法在解决类集成测试顺序问题时忽略了多态性或潜在的动态绑定特性而形成的动态依赖关系对类集成测试顺序的影响，即只考虑了静态依赖关系，导致类测试顺序准确度较低. 为了克服这一弊端，一些科研人员开始考虑了动态依赖关系对类测试顺序的影响. 具体分析如下：

国外，Malloy等人[23]既考虑了类间静态依赖关系和多态性而可能产生的动态依赖关系，又考虑了抽象类的影响. 他们[23]针对C++语言的特点，提出了一个由参数化的成本模型驱动的类测试顺序系统. 将类间关系扩展为关联、组合、依赖、继承、实现和多态等六种关系，其中，前五种类型的边常用于UML类图. 通过选择和删除权值最低的边来打破环路，与通常的方法不同的是，该方法可以删除任意类型的边来打破环路，即允许删除强依赖关系，其限制条件是需要满足所需构造的测试桩的个数尽可能少.

Kraft等人[11]提出了有趣的问题，即当ORD中存在环路时，应该删除哪些类型的边来打破环路. 对于他们的问题，其答案是打破环路时删除继承关系.

国内，北京邮电大学的李都[24]允许删除继承和聚合关系等强依赖关系，导致构造复杂的测试桩，而且在某些情况下显著增加了冗余测试桩. 装甲兵工程学院的周燕[44]不仅考虑了类间的静态依赖关系，而且还考虑了类间动态关系，更全面地分析了类间的依赖关系，该算法在一定程度上减少了集成测试过程中所需构造的测试桩的数目. 上海交通大学郑磊[45]以及太原理工大学的申小荣[46]改进了Kung等人[3]的拓扑排序算法，考虑了多态性和潜在的动态绑定特性可能产生的动态依赖关系.

但是以上的方法考虑的面向对象特点仍然不够全面，如缺少考虑抽象类的特点对类测试顺序的影响. 为了满足测试的完整性，不能忽略抽象类对类集成测试顺序的影响.

据我们所知，对于ORD中存在环路的类集成测试顺序问题，目前只有我们的工作[8,26]较全面地考虑了面向对象程序的特性，如继承和多态、抽象类不可实例化等，同时考察了这些特性对打破环路以及测试顺序的影响. 利用基于图论的算法，通过删除关联和多态等弱依赖关系打破环路，解决了基于图论中满足构造的测试桩的数目尽可能少这一条件的类集成测试顺序问题.

我们[8,26]认为在打破环路时，需要重点讨论抽象类的特点对所需构造的测试桩存在的影响. 我们以规则的形式来描述这种影响.

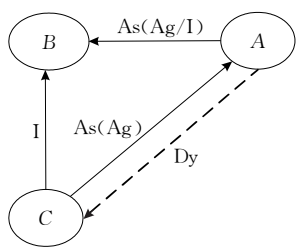

图10　示例WORD

规则. 假设A类是B类的客户类，B是C的父类(即类B既是一个服务类又是一个父类)，C是A的一个客户类. 那么A与C之间存在一条动态依赖边(如图10所示). 此时，需要为哪个类构造测试桩? 当A在B与C之前进行测试时，存在以下两种情况：

(1) 如果$A\rightarrow B$是聚集关系(Ag)，并且$A\rightarrow B$的类型为公有(Public)或保护(Protected)，① 如果B不是抽象类，则类B可以被实例化，C可以继承类B的数据成员. 此时需要为类B，类C，以及C的子类构造测试桩；② 如果类B是抽象类，则类B不能被实例化，然而它仍然说明类A所需要的服务，而这个说明仅仅是模拟创建类B的测试桩的一个信息片段. 在这种情况下，如果A在B与C之前进行测试，需要为类B构造测试桩. 而抽象类B失去了它作为服务类的角色. 因此，需要为类B、类C，以及C的子类构造测试桩. 综合①和②，不管B是否为抽象类，需要为类B，类C，以及类C的子类构造测试桩. 如果$A\rightarrow B$是聚集关系(Ag)，并且$A\rightarrow B$的类型为私有(Private)，则C不能继承B的数据成

员,因此,没有必要为 C 以及 C 的子类构造测试桩.因此,不管 B 是否为抽象类,只需要为类 B 构造测试桩.

(2)如果 $A\rightarrow B$ 为关联关系(As),与(1)中①②同理,需要为类 B,类 C 以及类 C 的子类构造测试桩.

综合(1)(2),可以将该规则归结为表 5 所示.其中,Y:是;N:否.

表 5 测试桩的构造规则

$A\rightarrow B$	$A\rightarrow B$ 的类型	Stub(B)		Stub(C)	Stub(C 的子类)
		B 是抽象类	B 不是抽象类		
Ag	Public	Y	Y	Y	Y
	Protected	Y	Y	Y	Y
	Private	Y	Y	N	N
As	—	Y	Y	Y	Y

打破环路以后,需要确定类集成测试顺序.抽象类不可实例化的特点导致其不能被独立测试,抽象类使得部分测试顺序是不可行的,需要考虑抽象类对类集成测试顺序的影响.我们的方法[4]是对抽象类不可实例化对类测试顺序的影响进行分别讨论:①忽略这种影响,根据测试级顺序分配策略得到类的测试顺序;②考虑这种影响,即在①的基础上调整类测试顺序.抽象类具有不可实例化的特点,在测试抽象类时,常用的策略是首先实例化抽象类的一个子类,然后,测试该子类的特征(继承特征),通过对该特征的测试来实现抽象类(父类)的测试.由此,可以将抽象类测试方案调整为:将抽象类移入到各自子类所在的测试级中,利用这种变化,那些处于不可行测试级而且以抽象类为源类的目标类可以转移至各自源类所属的测试级.

3.1.3 特殊情况(无环路+考虑动态依赖关系)

Labiche 等人[27]、Li 等人[7]和 Chen 等人[47]既考虑了类间静态依赖关系和多态性而可能产生的动态依赖关系,又考虑了抽象类的影响.然而,他们解决的是不存在环路的情况下如何对类进行排序的问题,即对于在静态关系中就已经存在环路的情况,他们并没有给出破除环路的有效途径.事实上,对于实际的面向对象程序来说,抛开动态依赖关系不考虑,静态依赖关系就早已经了构成环路,因此,有许多应用的局限性.

文献[44-46]中的这一类方法研究的重点也是提出在不包含环路的图(或者打破环路以后的环路图)的拓扑排序方法.然而这类方法能应用于存在环路的图.类间动态依赖关系是在程序在执行阶段产生,具有动态性和实时性,当删除类间静态关系时,一部分类间动态依赖关系仍然会存在,也有一部分动态依赖关系会消失.那么对于尚且存在的这部分动态依赖关系,连同静态依赖关系,可以采取文献[44-46]中的这一类方法生成类集成测试顺序.因此,对于存在环路的对象关系图,我们还需要重点分析考虑动态依赖关系时如何来打破环路,即需要分析动态依赖关系对打破环路的影响.特别地,还要知道何时需要删除动态依赖关系来打破环路,或者要知道哪些动态依赖关系会随着静态依赖关系的删除而消失.

我们知道,Kung 等人[3]的算法没有考虑程序的多态性,即没有考虑类间动态关系,是一种基于类间静态依赖关系的类测试顺序生成算法.以图 11 所示的包含动态依赖关系的 ORD 图为例,Kung 等人[3]的算法得到 4 层测试顺序,并且各层内的多个类没有先后顺序,如表 6 的第二列所示.

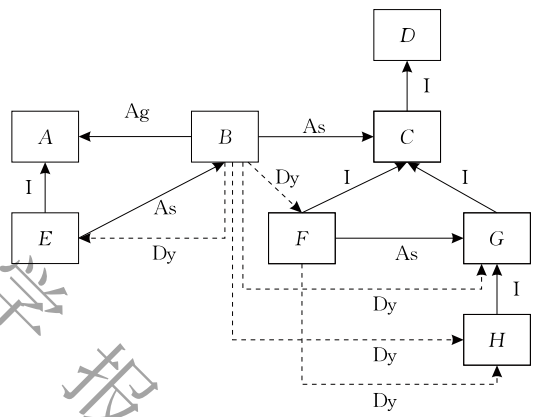

图 11 包含动态依赖关系的 ORD 图(EORD)

表 6 测试顺序表

	文献[3]	文献[44-46]
Test level	Class(es)	Class(es)
1	A,D(可互换)	D,A
2	C	C
3	B,G(可互换)	G,B
4	E,F,H(可互换)	E,H,F

文献[44-46]分析发现,类 F 与类 G 之间是关联关系,类 H 是类 G 的子类.考虑面向对象的多态性时,类 F 与类 H 之间会产生一定的依赖关系,具体来说,是 F 可能会动态依赖于类 H.因此表 6 中的第 4 层中的类 F 和类 H 不能进行单独测试,类 H 测试之后才能测试类 F.

文献[44-46]中的算法同样可以得文献[3]的 4 层测试顺序,但还需要进一步确定各层层内的测试顺序.其中,在第 1 层中,通过分析可知类 A 和类 D 之间不存在动态依赖关系,而类 A 和类 D 的入度

分别为 2 和 1，那么第 1 层的测试顺序应该是 D、A（不可互换）；由于第 3 层中的类 B 在运行时可能动态依赖于类 G，那么首先测试类 G，再测试类 B；第 4 层中，分析可知类 F 动态依赖类 H，那么首先测试类 H，再测试类 F. 因此，文献[44-46]得到的整体测试顺序为：D, A, C, G, B, E, H, F.

采用 Kung 等人[3]的算法、周燕[44]的算法、郑磊[45]的算法和申小荣[46]的算法所生成的类测试顺序情况如表 6 所示. 由表 6 的结果我们可以发现，在对类簇进行排序时，是否考虑程序运行过程中由于多态性而可能产生的类间动态依赖关系对类集成测试顺序有较大影响. 考虑多态性时，类间依赖关系分析更加全面，处于同一层的类不再被单独测试，而是采取一定的策略确定层内所有类的测试顺序，这样得到的类测试顺序更加准确.

3.1.4　最小化测试桩的总体复杂度(MOSC)

由 3.1.1 节与 3.1.2 节内容可以发现，目前大多数研究方法以构造测试桩的数目为评价测试成本的指标，以构造的测试桩总体复杂度为评价指标的方法相对较少. 但是，以测试桩的总数目作为问题的优化目标时，会造成类集成测试顺序精确度较低. 这是由于一般情况下不同测试桩的复杂度是不相等的，因此可以说，测试桩的数量多少与生成一个类集成测试顺序所花费的真实、准确地总体代价的高低不成正比. 测试桩的总体复杂度可以表示一个测试顺序需要花费的总体代价，以此作为问题的优化目标时精确度较高. 因此，应该以测试桩的总体复杂度来评价生成一个测试顺序需要花费的总体代价.

Abdurazik 和 Offutt[13]提出了一种基于耦合度量和图论相结合的技术. 首先，利用结点和边的权重模拟类以及类之间的关系；其次，根据定量的耦合度量来确定节点和边的权重；最后，打破环路时综合考虑边权重、节点权重以及环路的个数这三个因素. Abdurazik 和 Offutt[13]的方法对使用的属性复杂度和方法复杂度的权重取均值，即各为 0.5. 该方法需要构造更复杂的测试桩，但得到类测试顺序所花费的总体测试桩的复杂度最少.

我们提出了基于耦合度量的类测试顺序确定方法[5]. 该方法将基于图论的启发式算法与类间耦合度量相结合，用于实现在满足测试桩总体复杂度尽可能小的条件下打破环路，其中，耦合度量用于度量每个测试桩的复杂度. 与 Abdurazik 和 Offutt[13]的方法不同的是，我们[5]将属性复杂度，方法复杂度，返回值复杂度和传递的参数复杂度这 4 个都作为影响测试桩复杂度的因素，并对它们的权重 W_V，W_M，W_R 和 W_P 给出了计算方法，如式(8)～式(11)：

$$W_V = V/(V+M+R+P) \tag{8}$$

$$W_M = M/(V+M+R+P) \tag{9}$$

$$W_R = R/(V+M+R+P) \tag{10}$$

$$W_P = P/(V+M+R+P) \tag{11}$$

实验结果表明，采用文献[5]的方法生成一个类集成测试顺序所花费的总体复杂度有所降低，节约了测试成本.

该方法没有考虑抽象类的特点，实际上，在一个程序中抽象类普遍存在，因此在程序测试时忽略它会影响类间的依赖性，进而导致类测试顺序的准确度降低. 另外，已有的类间分析方法大多数忽略类间的动态依赖关系，然而，多态性是面向对象程序的典型特点之一，由其产生的动态依赖关系大量存在. 这就要求在确定类的测试顺序时，需要构造动态依赖关系的测试桩. 由前面分析已知，部分类间动态依赖关系会随着类间静态依赖关系的打破而消失. 对于没有消失的动态依赖关系，还需要我们解决以下三个问题：孩打破环路时，在哪种情况下除了删除类间静态依赖关系，还需要删除那些没有消失的类间动态关系；虎如何构造类间动态依赖关系的测试桩；虎如何计算这些没有消失的动态依赖关系的权重.

东北大学的赵玉丽等人[28]提出了一种基于复杂网络的类间集成测试顺序生成方法. 首先，建立面向对象软件的复杂网络模型，并根据软件网络的结构特征，分析类节点的影响力和复杂性，提出一种节点重要性的度量方法；然后，将错误被尽早发现的思想应用于类集成测试顺序的确定问题，综合考虑桩复杂度和节点重要性，在确保构造的测试桩复杂度尽可能小的前提下，优先测试重要节点. 该方法在确定类测试顺序时，为了花费尽可能少的测试桩并满足类集成测试的完整性要求，保证在测试每个类之前，已经测完其依赖的类.

这一方法[28]从一个新的角度来解决类集成测试顺序问题. 但是该方法没有考虑抽象类的特点，没有考虑类间的动态依赖关系，也没有区分类间依赖关系的强弱，导致得到的类测试顺序不够准确. 因此，我们可以借鉴该方法的思想，再结合考虑面向对象的继承、多态、抽象类的特点，给出解决类集成测试顺序问题的新途径.

东北大学的于海等人[29]提出了一种基于程序节点重要性的类集成测试序列生成方法. 该方法借鉴复杂网络思想，首先将程序抽象为网络，并结合类

的重要性度量方法,打破程序构成的网络中的环路,然后根据网络的无环链路进行逆向拓扑排序,生成类集成测试序列.在确定类的集成测试顺序之前,首先对类的重要性进行度量,得到出错可能性大、一旦出现错误对整个待测软件系统传播影响力强的类,这样可以保证该方法确定的类集成测试顺序使得重要的类节点被优先测试.在确定类集成测试顺序时,遵循尽可能早地发现错误的原则,即考虑了测试方法自身的揭错能力这一重要目的.实验结果表明该方法还能够使得构造的测试桩的总体复杂度较小.该方法的所考虑的测试方法自身的揭错能力是本方法的一个重要创新点.

3.1.5 基于切片技术的方法(SLicing)

北京邮电大学刘颖莲[30]结合利用切片技术,提出了考虑面向对象程序特点的类集成测试方法.首先,通过 UML 类图生成有向图,用于表示类间依赖关系;其次,计算该有向图中各个节点的权值,并依据权值大小对有向图进行切片;再次,计算各个切片的度数和切片的权值,如果度数超过给定的阈值,再折半拆分该切片,直至所有切片的度数均小于阈值为止;最后,按照切片的权值(即切片中包含的所有节点权值的总和)进行集成测试.其中,一个切片由多个表示类的节点构成,因此,切片的复杂度取决于其所包含的类节点的复杂度(一个类的复杂程度由类的成员变量个数、成员函数个数、调用其它类中的方法个数和被其它类调用的次数这四个指标来确定).类的集成测试优先级由各个类的复杂度来确定.

该方法为解决现有的类集成测试顺序的生成问题提供了一种全新的解决思路.与于海等人[29]的方法类似,该方法也对类的重要性进行了度量,得到出错倾向性大、一旦发生错误对整个待测软件系统传播影响力强的类,这样可以保证该方法确定的类集成测试序列使得重要的类节点被优先测试,考虑了测试方法自身的接错能力.但是,通过分析发现这一方法并没有区别类间依赖关系的强弱,而是都将其归为调用关系,这样得到的类测试顺序结果是不够准确.因此,还需要划分类间依赖关系的强弱,引入依赖边权值的概念,并将边权值用作计算类节点权值的一个因子.这样导致该方法不能够满足构造的测试桩的总体复杂度较小.

3.2 基于搜索的方法

基于图论的方法遵循删除最少的边打破尽量多的环路的原则来识别和消除环路.Briand 等人[9,12]提到基于图论的算法的缺点:在打破环路的时候,基于图论的解决方案假设每个测试桩的开发成本相同.但是 Briand 等人[9,12]认为存在这样的情况:删除两个依赖关系比只删除一个所花费的成本更低.此外,在基于图论的解决方案中,要实现充分考虑并估算需要构造的测试桩的具体成本(如类中属性的数量、调用方法的数量以及各种约束等各种因素)是相当困难的[17].因此,已有的基于图论的解决方案并没有充分解决类集成测试顺序的确定问题.

为了克服这个限制,Briand 等人[9,12]将类集成测试顺序的确定问题作为一个多目标优化问题.优化的目标函数有两个或两个以上时称为多目标优化.所谓的目标优化问题[33]是指采用一定的优化算法求解目标函数的最优解.所谓的多目标优化问题一般是将多个目标聚合成一个函数,进而形成单目标函数,从而求解到最优解.采用多目标优化算法确定类集成测试顺序的基本思想是:结合包含依赖信息的对象关系图、包含代价信息的代价模型以及类间的约束条件等信息,采用多目标优化算法获得类集成测试顺序的非支配解集,然后再根据用户或者系统的特定要求,生成满足条件的类集成测试顺序.这种方法的缺点是各目标权值的分配具有较大的主观性.

基于搜索的类测试顺序确定方法的总体思想是:首先设计初始种群,然后在满足一定条件的前提下,通过进化操作对这一种群进行处理,进而打破环路,得到最优的类集成测试顺序.具体来说:首先确定一个表示类集成测试顺序的初始种群;然后,设计适应度函数并计算每个个体的适应度值;最后,判断是否满足终止收敛条件或进化代数.如果不满足,通过进化操作对这一种群进行处理,生成新的种群,并重新计算每个个体的适应度值,重复该步骤,直到满足终止条件;如果满足,生成的类测试顺序即为最优的类测试顺序.其流程图如图 12 所示.

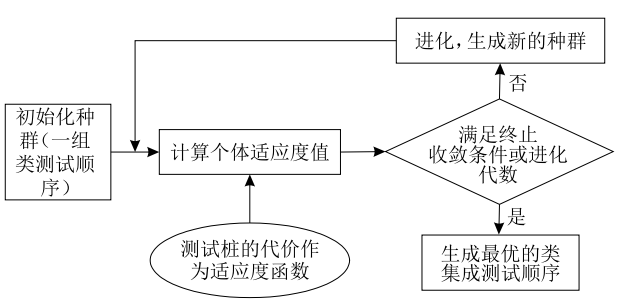

图 12 基于搜索的类测试顺序生成流程图(静态依赖关系)

对于基于搜索的方法,目前将所需构造的测试桩数目作为评价个体好坏的适应度函数的文献相对

较少.这是由于以测试桩数目作为问题的优化目标时精确度较低.因此,应该更倾向于以测试桩的总体复杂度作为适应度函数,来评价生成一个测试顺序需要花费的总体代价,进而评价类测试顺序生成方法的优劣.因此,本文从所采用的优化算法的不同进行分类描述,包括基于线性加权的多目标优化算法的方法和基于帕累托模型的优化算法的方法.

3.2.1　基于线性加权的多目标优化算法的方法

我们将基于线性加权的多目标优化算法的方法分为基于遗传算法的方法、基于模拟退火算法的方法和基于粒子群算法的方法.

3.2.1.1　基于遗传算法的方法(GA)

Briand 等人[9,12]提出了基于遗传算法的解决方案.他们的解决方案中,将耦合度量技术和遗传算法相结合,用于计算构造复杂的测试桩模块所花费的代价.对于遗传算法,需要解决如下几个问题:

(1)如何编码?

(2)如何进行交叉和变异操作?

(3)初始种群的规模?

(4)如何评价解是否最优?

对于问题(1),每一个个体用一组用标签表示的类表示.对于一个类簇$\{A,B,C,D\}$,一个个体可能为(B,C,A,D)序列,一个可能的种群可能为$\{(B\ C\ A\ D),(B\ C\ D\ A),\cdots\}$.

对于问题(2),需要针对原始种群及其后代进行交叉和变异操作,直到得到一个可以接受的(可能是最优的)解.变异操作指的是选择一个个体中的两个类并交换它们在该个体中的位置.例如,对于一对随机选择的基因(测试顺序中的类),$(B\ \boldsymbol{C}\ \boldsymbol{D}\ A)$变异为$(B\ \boldsymbol{D}\ \boldsymbol{C}\ A)$.交叉是一个相对复杂的操作,随机选择第一个个体(测试顺序)的基因(测试顺序中的类),从第二个个体中找到它们的位置,并将第二个个体中剩余的基因按它们在本个体中的顺序复制到第一个个体中.这样在创建一些新的子序列时保存了在原父代个体中的一些子序列.例如,如果我们假设有以下两个父代个体:

Parent 1:$(A\ B\ C\ D)$

Parent 2:$(D\ C\ B\ A)$

如果我们进一步假设选中第一个父个体中A和C,可以生成一个如下子代个体:

①我们得到(A-C-),其中,A和C被选中,'-'表示省略的类;

②按照B和D在父个体2中的顺序,将它们填充到省略的类的位置,得到$(A\ D\ C\ B)$.

对于问题(3),需要对遗传算法进行一定的参数设置,如种群的规模.Briand 等人[9]指出初始种群的规模在25～100之间.但是对于基于优化算法求解类集成测试序列的问题需要一定数量的初始种群来保证解的多样性.作为启发式算法,初始种群的规模一般是类数量的两倍到三倍.因此设定初始种群的规模是100.粒子群优化算法中粒子的初始速度是随机设定的,若粒子群随机设置的速度过快,容易跳过最优解,若粒子群的运动速度过慢,容易陷入局部最优解.

对于问题(4),他们针对一组真正的项目展开了一个实验,研究了四种不同的适应度函数:依赖关系的个数、方法耦合、属性耦合以及属性和方法的几何平均.他们实验证明在大多数情况下,将属性和方法的平均耦合作为适应度函数时产生最好的解决方案.由于 Briand 等人[9,12]的这一方法指出为每个耦合度量指标(适应度函数)分配权值的必要性,因此该方法具有主观性并且需要熟练掌握面向对象类的特点的相关知识.这导致其不适合复杂的软件系统.

Hanh 等人在文献[16]中除了采用基于图论的算法来打破环路,还采用了遗传算法来打破环路.首先确定个体和初始种群,其中一个个体可以表示为一个类集成测试顺序,初始种群可以表示为一组类集成测试顺序,然后在满足测试桩个数尽可能小这一条件的前提下,通过交叉变异等进化操作对这初始种群进行处理,进而产生满足的类测试顺序.其中,将所需构造的测试桩的个数作为评价个体好坏的适应度函数.

对于遗传算法来说,适当的评价函数的选择是一个关键任务,往往出现非常昂贵,费时耗力的复杂情况.此外,遗传算法是在一定范围的解空间上进行交叉、变异,并不能包含所有的解情况.进行交叉变异的操作在一定程度上会降低方法的效率.

当只考虑类之间的依赖关系时,Briand 等人[9,12]给出了形式化的证明:在仅考虑类之间依赖关系的情况下,基于遗传算法的类测试顺序生成方法比基于图论的解决方案更优.然而,大型面向对象程序错综复杂,存在多种因素影响类的测试顺序,因此难以设计合适的基于遗传算法的解决方案.而且,多种影响因素的耦合度量依赖于传统的手工分析,耗费大量的人力物力,可操作性不佳,因此在实际的工业领域应用较少.

国内,在面向对象软件集成测试中,针对类测试顺序生成问题,王正山[48]提出了一种应用混合遗传

算法的方案.该算法针对基本遗传算法的局部搜索能力比较弱且容易产生早熟的缺点,在基本遗传算法的基础之上增加局部搜索以增强局部搜索能力,以及使用缓冲池以减少运行时间.实验结果表明,该算法的求解质量方面优于 Briand 等人[9,12]的遗传算法.

Borner 等人[31]将 ORD 作为一种依赖模型.其中,ORD 由源代码创建,包括三种依赖关系:继承、关联和依赖.他们认为在执行集成测试时应考虑测试的焦点.他们使用模拟退火算法和遗传算法找到一个最优的测试顺序.该方法不仅考虑了模拟工作所花费的代价,而且考虑了测试的焦点.

3.2.1.2 基于模拟退火算法的方法(SA)

模拟退火是一种随机寻优算法,可以有效避免陷入局部极小,最终可以趋于全局最优的一种优化算法.其运行时间较短,已经被证明可以较好的应用于求解优化问题.采用模拟退火算法生成类集成测试顺序的基本思想是:在控制参数(当前温度 t)下,通过迭代来不断变换类的位置,并根据 Metropolis 准则得到最优的类测试顺序.利用贪心策略的思想变换类的位置,整个变换的过程可以看作是一个 Markov 链.文献[49]研究表明最优类集成测试顺序的生成需要进行多次变换,并且"仅当变换数至少是解空间规模的平方次时,模拟退火算法才能任意逼近平稳分布".对于类集成测试顺序的确定问题来说,解空间规模是类个数的阶乘,因此变换次数的规模近似于类个数的指数次.如果采用任意近似逼近的方式,生成收敛于全局的最优类集成测试顺序,模拟退火算法的执行时间将呈指数级增长方式.对于类个数较多的待测系统来说,运行时间难于承受,导致算法不可行.

然而,模拟退火算法可以通过允许任意选取初始类测试顺序、早期操作选择较优近似解的方式帮助减少工作量;同时,控制参数序列由冷却进度表决定,冷却进度表不断逼近模拟退火算法的渐进收敛状态,虽然难以获得全局最优解,但能够在有限时间内获得近似最优解.

3.2.1.3 基于粒子群算法的方法(PSO)

粒子群优化算法与遗传算法都是比较常用的人工智能算法,文献[50-51]证明许多组合优化问题可以使用粒子群优化算法解决,如旅行商问题,顺序排序问题,集合覆盖问题等.而类集成测试顺序的生成问题是一个优化问题,同时,文献[9,12]也已经证明遗传算法可以应用于类集成测试顺序的确定问题中,鉴于此,可以将粒子群优化算法应用到类集成测试顺序的确定问题中.

因此,我们利用粒子群优化算法尝试解决了类集成测试顺序问题,并与基于图论的方法、基于遗传算法的方法,和随机迭代方法进行了比较[6].该方法包含如下三个模块:

(1)位置区间的映射:对源程序中包含的所有类进行排列,将类测试顺序映射为粒子群中的粒子,从而使粒子的每一个位置对应一个类测试序列;

(2)最优粒子的生成:根据适应度函数计算每个粒子的速度和位置,然后通过粒子群优化算法选择粒子的最优位置和最优适应度;

(3)类测试序列的映射:根据映射关系,将选择的最优粒子映射为其对应的类测试顺序,则该测试顺序就是所求得的最优测试顺序.最优测试顺序使得构建测试桩的总体复杂度最小,测试花费的代价最小.

通过实验验证可以得到如下的结论:

(1)与基于图论的方法、基于遗传算法的方法和随机迭代方法相比,基于粒子群优化算法的方法可以一定程度找到测试桩代价更小的类集成测试序列;

(2)在基于图论的方法、基于遗传算法的方法和随机迭代方法中,随机迭代方法可以最快地找到最优类集成测试序列.对于规模较大的系统来说,与基于遗传算法的方法相比,在相同条件下(如种群数量、迭代次数),基于粒子群优化算法的方法可以更快找到更优的类集成测试顺序;

(3)与基于图论的方法、基于遗传算法的方法和随机迭代方法相比,基于粒子群优化算法的方法可以一定程度找到属性复杂度、方法复杂度更低的类集成测试顺序.

总体来说,我们通过实验验证了各种算法的适应性,具体如下:对于规模较小的系统,或者包类个数较少、且包含环路的个数也较少的系统,适用于基于图论的方法.这是由于环路少,容易划分 SCCs,便于直接从环路入手.对于规模较大的系统,或者类的个数较多,且包含环路的个数也较多的系统来说,不适用于基于图论的方法.主要有以下两点原因:首先基于图论的方法难以充分估计构建所需测试桩的具体成本[17],其中的影响因素包括类中属性的数量、类中方法调用的数量以及其他的各种约束等,由此基于图论的方法得到的解往往是次优的;其次,对于这类大规模系统,模拟图论的方法具有相当的复杂性,难以实现.

对于规模较小的系统，或者包类个数较少、且包含环路的个数也较少的系统，基于搜索的方法具有盲目性导致其不适用于解决这类系统的类集成测试顺序的问题. 而对于规模较大的系统，或者类的个数较多，且包含环路的个数也较多的系统来说，基于搜索的方法更适应于解决这类系统的类集成测试顺序的问题.

此外，由于粒子群算法的全局收敛性具有不确定性，因此也会出现个别较差的类测试顺序.

3.2.2　基于帕累托模型的优化算法的方法

我们将基于帕累托模型的优化算法的方法分为基于蚁群算法的方法、基于禁忌搜索的算法的方法和 NSGA-II、SPEA2 和 PAES.

3.2.2.1　基于蚁群算法的方法(PACO)

如 3.2.1.1 节所述，大型面向对象程序错综复杂，存在多种因素影响类的测试顺序，难以设计合适的基于遗传算法的解决方案. 而且，多种影响因素的耦合度量依赖于传统的手工分析，耗费大量的人力物力，可操作性不佳，因此在实际的工业领域应用较少. 针对以上问题，Cabral 等人[32]提出了一种基于帕累托蚁群克隆算法(Pareto Ant Colony System Algorithm，PACO)的解决方法，将 ORD 作为一种依赖模型. 其中，ORD 由源代码创建，包括四种依赖关系：继承、组合、关联和依赖. 他们使用了基于蚁群优化算法的多目标优化算法[52-53]来生成 Pareto 最优解集合(满足约束条件和所有目标函数的一组决策变量和相应各目标函数值的集合)，实现了属性复杂度和方法复杂度之间的平衡. 即一个类测试序 o 的测试桩复杂度取决于其属性和方法耦合. 按如下方式计算这两个复杂度：

(1) $ACplx(o)$(属性复杂度)：当引用/指向源类中的一些方法的参数列表中的目标类时，属性复杂度计算在这些目标类中局部声明的属性个数. 该信息表示为一个矩阵 $\boldsymbol{A}(i,j)$，其中，行和列表示类，i 依赖于 j.

$$ACplx(o)=\sum_{i=1,n}\sum_{j=1,n}\boldsymbol{A}(i,j);\ j\neq k \qquad (12)$$

其中，对于一个类测试序 o，n 表示类的总数，k 表示包含在类 i 之前的任何类.

(2) $MCplx(o)$(方法复杂度)：方法复杂度计算在被源类的方法调用的那些目标类中局部声明的方法的个数. 该信息表示为一个矩阵 $\boldsymbol{M}(i,j)$，其中，行和列表示类，i 依赖于 j.

$$MCplx(o)=\sum_{i=1,n}\sum_{j=1,n}\boldsymbol{M}(i,j);\ j\neq k \qquad (13)$$

其中，对于一个类测试序 o，n 表示类的总数，k 表示包含在类 i 之前的任何类.

在理想状态下，可以认为蚁群算法确定类集成测试顺序的方法收敛于全局最优. 但是对于最优类测试顺序并不唯一的实际待测系统来说，最优顺序和次优顺序可能有重叠的子顺序，这样当搜索过程中强化最优类测试顺序上的信息素时，也会强化次优的类测试顺序上相关部分的信息素，那么次优解还会以较高的概率被输出. 因此，采用蚁群算法不能保证找到问题的最优解，而是只能保证找到问题的较优解或可接受解.

类集成测试顺序问题本质上是一个优化问题，同时也是一个 NP-完全(难)问题. 蚁群算法作为一种群体智能搜索算法，为解决优化问题提供了一种有效途径. 同时，蚁群算法作为一种随机搜索算法，在 NP-完全(难)问题上的近似性能分析贴近实际应用，对算法的设计和应用等具有重要的指导意义.

3.2.2.2　基于禁忌搜索的算法的方法(MTabu)

Vergilio 等人[33]提出了两种算法：非占优排序多目标遗传算法(Non-dominated Sorting GA，NSGA-II[54-55])和禁忌搜索多目标优化(MTabu[56-57]). 其中，MTabu 不是进化算法，而是一种基于禁忌搜索的算法，是运筹学领域最适用的算法[56]. Vergilio 等人[33]使用 Briand 等人[9]的基准程序执行这两种多目标优化算法，并根据帕累托占优概念[58]来评估这些算法返回的解. 这两种多目标优化算法采用的耦合度量指标为：方法和属性的数量. 实验结果比较发现：对于所有的被测系统，与 PACO 和 MTabu 相比较，NSGA-II 算法更优. 这是由于 MTabu 一般不是作为独立算法完成某项任务，更主要的是用于防止陷入局部最优以及避免长期无效搜索. 因此它的主要作用是开辟了一条优化类集成测试顺序的新途径. 目前还难以证明其是否真正适用于解决类集成测试顺序的确定问题.

3.2.2.3　NSGA-II、SPEA2 和 PEAS

NSGA-II、SPEA2 和 PAES 这三种不同的多目标进化算法在局部寻找最优解的同时，保持种群多样性，力求得到近似最优类集成测试顺序.

其中，NSGA-II 是最著名的算法之一，使用的是多目标优化算法[59]，是目前用于解决类集成测试顺序问题的主要途径之一. Assunção 等人[34]对 NSGA-II 和改进的强度帕累托进化算法(SPEA2[60])这两种多目标优化算法进行了性能分析与比较. 对多个待测系统分别采用两个目标(属性复杂度和方法复杂

度)和四个目标(属性复杂度、方法复杂度、返回值复杂度和参数复杂度)确定类集成测试顺序.实验结果表明 NSGA-II 和 SPEA2 在两个目标下结果相同,SPEA2 在四个目标下结果更优.这一结果可以初步说明 SPEA2 比 NSGA-II 更适用于解决维数较多的问题.后来,Assunção 等人[35]又进一步对 NSGA-II、SPEA2 和 PAES 这三种算法进行了比较.实验结果表明对于比较复杂的系统,PAES 算法效果更优,这是由于 PAES 算法只保留了一个内部和一个外部种群,这样在有限的时间内只需要执行少量的种群变换操作.但是对于大多数实验对象来说,NSGA-II 效果最好.这是由于 NSGA-II 改进了交叉算子和变异算子,使得在满足类集成测试顺序求解需要的同时发现新个体,提高算法的执行效率.

3.3 其它方法

3.3.1 基于随机迭代算法的方法(RIA)

东南大学的李必信和王正山[36]使用一个扩展的对象关系图(EWORD)作为依赖模型.EWORD 由源代码创建,包括六种类型的依赖关系:继承、实现、组合、聚合、关联和使用.在类测试顺序生成问题中,在计算测试桩复杂度时引入了耦合度量的方法,他们认为继承和聚集等强依赖关系可以允许被断开,并在耦合度量过程中加入了随机交互算法.该技术首先为所有类型的边估计测试桩的复杂度,然后使用随机迭代算法打破环路.该算法使用最小反馈弧集的一些性质和模拟退火的思想,提高了其有效性.这一方法也是针对的类间静态关系进行的分析.

随机算法,在 NP-完全(难)问题上的近似性能分析贴近于实际应用,对指导算法的设计和应用等具有重要的指导意义.

3.3.2 基于超启发式算法的方法(HHA)

超启发式算法[61]是智能计算领域最近新出现的智能算法.该算法由高层策略与低层启发式算法组成.该算法的特点是高层策略操纵或管理一组低层启发式算法,两个层面协同工作,以获得新启发式算法,这些新启发式算法则被运用于求解各类 NP-完全(难)问题,共同实现超启发式算法.Guizzo 等人[37]提出用超启发式算法来解决类测试顺序问题.首先,初始化参数配置,初始化低层次启发算法以及种群;然后,评估初始种群的适应度;其次,迭代进化直到满足停止条件,生成最佳种群.其中,在迭代进化过程中,首先选择父代个体,然后选择低启发式算法(NSGA-II)并应用它生成新的解集,其次评价新的解集的适应度值.针对 7 个实验对象的结果表明超启发式算法比 NSGA-II 的效果要好.

超启发式算法的高层启发式方法部分几乎不依赖于问题的领域知识,低层启发式算法则与问题的领域知识紧密相关.启发式算法的应用目前已经非常广泛,而超启发式算法由于其历史短,主要还局限于一些常见的组合优化问题.

3.4 评测数据集

对于类集成测试顺序的问题,研究人员一般借助实验室或开源软件界的评测程序进行研究.我们对论文中采用的评测数据集进行了汇总,最终结果如表 7 和表 8 所示,所列系统是研究人员最为认可的评测数据集.ATM,ANT,SPM 和 BCEL 系统有合适的类(类个数在 19～45 之间),有不同数目的环路(ATM 的 30 个环路到 BCEL 的 416 091 个环路),DNS 含有的类最多(同样 BCEL 含有最多的依赖边),但是环路个数最少.JBoss、JHotDraw 和 MyBatis 系统的类个数在 150 和 331 之间.可以看出,所选的每个系统具有不同的类图,环路数,在不同的系统中二者的分布不同,系统具有不同的复杂度.

其中,绝大多数基于图论的论文均采用表 7 所示程序作为主要评测数据集.绝大多数基于搜索的论文均采用表 8 所示程序作为主要评测数据集.可以发现,表 8 没有对类间的依赖关系进行细分,只给出了总的依赖个数,这是由于基于图论的方法在打破环路时对于所删除的边的类型有一定的限制条件,比如,大多数只选择删除弱依赖关系(使用、关联、简单聚合等),而不允许删除强依赖关系(继承、组合、聚集等).而对于基于搜索的方法,在打破环路时对于所删除的边一般没有任何限制,即允许删除任何类型的依赖关系.这是因为在打破环路时如果对于所删除的边有所限制,如不允许删除继承边,将导致求解范围缩小,可能会遗漏最优解.

表 7 已有基于图论研究中使用的评测数据集

系统	类数	使用关系	关联和聚合数	组合数	继承数	循环数	代码行数
ATM[3]	21	39	9	15	4	30	1390
ANT[3]	25	54	16	2	11	654	4093
SPM[3]	19	24	34	10	4	1178	1198
BCEL[3]	45	18	226	4	46	416091	3033
DNS[3]	61	211	23	12	30	16	6710

表 8　已有基于搜索研究中使用的评测数据集

系统	语言	类数	依赖数	代码行数
ATM[9]	Java	21	67	1390
ANT[9]	Java	25	83	4093
SPM[9]	Java	19	72	1198
DNS[9]	Java	61	276	6710
BCEL①	Java	45	289	2999
JBoss②	Java	150	367	8434
JHotDraw③	Java	197	809	20273
MyBatis④	Java	331	1271	23535

4　未来研究方向

由本文所述内容可知：在集成测试中，类集成测试顺序的确定问题是一个值得研究的热门领域，具有较高的理论价值与较广泛的应用前景. 类集成测试顺序生成领域虽然在目前已经有了一些研究成果，但仍存在一些值得解决的问题，主要包括：

(1) 已有搜索算法的优化. 对于(单目标)粒子群优化算法，将粒子群优化算法应用到类集成测试顺序的生成问题中时，可能会带来一些问题，如当类的数量过大时，形成的类排序顺序成指数关系增长，大大增加解空间的规模，对于使用粒子群优化算法进行求解时，容易陷入局部最优解，这时就需要采取一定的策略避免陷入局部最优解.

我们可以在粒子群优化算法中引入奇异值分解的思想. 每隔若干代，预测进化方向，并在进化方向的正交方向生成新的解；对于新生成的解，我们处理其速度项，使其与正交方向保持一致；最后利用交替变量法对每一代中的最优个体进行局部搜索. 因为搜索区域不断的向正交方向延伸，因此可以增加全局搜索能力. 此外，由于速度项不断受到正交方向的影响，因此奇异值分解次数减少，降低了时间开销. 在全局搜索算法中引入局部搜索策略，对算法整体的全局搜索能力与局部搜索能力进行了合理的平衡，可以提高整体的搜索效率，从而提高最优类集成测试顺序的生成效率.

此外，我们还可以采用多目标粒子群优化算法来解决类集成测试顺序的生成问题. 这时可能会带来以下几个问题：

① 如何选择 pbest. 我们知道对于单目标优化来说，选择 pbest 的时候，只需要对比一下就可以选择出哪个较优. 但是对于多目标来说，两个粒子的比较并不能比较出哪个更优. 如果粒子的每个目标都要好的话，则该粒子更优. 若有些更好，有些更差的话，就无法严格的说哪个更好，哪个更差一些.

② 如何选择 gbest. 我们知道，对于单目标来说，种群中只有一个最优的个体. 而对于多目标来说，最优的个体有很多个. 而对粒子群优化算法来说，每个粒子只能选择一个作为最优的个体(领带者). 该如何选择？

(2) 其他元启发式搜索技术的应用. 元启发式搜索算法是启发式算法的改进，具有随机方法与局部搜索算法相结合的特点. 总体来看，元启发式算法具有应用范围广，花费较少(包括计算时间和占用空间)，易于维护、调整，期待一个较优解、甚至是最优可行解等特点，可以应用于求解优化问题. 典型的元启发式搜索算法包括遗传算法、模拟退火算法和爬山算法等. 可以根据具体的问题选择相应的算法进行求解. 其中，遗传算法和模拟退火算法已经被证明可以应用于求解类集成测试顺序的确定问题中.

爬山算法本质是一个贪心算法，通过多次选取随机的初始解，从一个解开始不断迭代向最优的解转移. 避免进行遍历，而是通过启发选择部分节点，从而达到提高效率的目的. 爬山算法在求解其他优化问题(如 TSP 问题)已经表现出来具有比较简单，效率高等特点. 因此，我们还可以采用爬山法等其它元启发式搜索算法来解决类集成测试顺序的确定问题. 但是，由于爬山算法不是全面搜索，因此最终结果可能不是最佳.

(3) 不同算法适应性的理论证明. 对于不同的搜索算法和图论的算法对不同的数据集的适应性，如什么类型的算法适合什么特点，多大规模的数据集等，已有的方法都是通过实验的方式验证来进行验证. 今后的工作中，可以尝试通过理论证明的方式来进一步证明我们所得到结论或者探索新的结论.

(4) 面向对象特点的考虑. 对于类集成测试序列的确定问题的研究，目前所采用基于搜索的方法都是在类簇中类间静态依赖关系的基础上进行的，并没有考虑以动态绑定、抽象类等为代表的类间动态依赖关系，然而类间的动态关系在实际的面向对象程序中广泛存在，忽略类间的动态依赖关系导致生成的类集成测试顺序的结果是不准确的. 因此，采用基于搜索的方法解决类集成测试顺序生成问题时需要进一步对程序中的动态依赖关系进行研究.

① http://archive.apache.org/dist/jakarta/bcel/old/v5.0/
② http://www.jboss.org/jbossas/downloads.html
③ http://sourceforge.net/projects/jhotdraw/
④ http://code.google.com/p/mybatis/downloads/list

(5)打破环路算法的完善.已有的破环算法是在构建测试桩的基础上加入回归测试来保证最终测试顺序的完整性.本文前面介绍的以测试桩的总体复杂度作为衡量测试代价的相关工作大多都考虑了基于关系类型(继承、组合、聚集、关联、使用等)以及属性和方法等的耦合度量.但是,已有的以构造的测试桩个数作为衡量测试代价的相关工作在打破环路时一般是假设删除每一条关联边所花费的代价是相同的,事实上所花费的代价往往是不同的,因此这种假设并不准确.因此,在未来的研究工作中,当以构造的测试桩个数作为衡量测试代价的指标时,可以将耦合度量技术用于关联边的度量,进一步优化类集成测试顺序.

另外,由于类集成测试顺序问题本质上是一个NP-完全问题,因此,存在着一定的不确定性(当存在多条边权值相同的依赖边时,随机选择删除其中的一条).如何降低或消除这种不确定性也是一个研究重点.在软件系统中,被频繁调用或调用某个类的特征次数较多的类通常具有较高的复杂性与较强的影响力,如果该类出现错误,则很有可能将错误传播到软件系统的其他类中,因此可以认为该类很重要,应该优先被测试.可以使用类的重要性判断测试的优先级.当存在多条边权值相同的依赖边时,我们可以度量构成相同权值的类的重要性.可以按照先测试重要性高的类,后测重要性低的类的原则来降低或消除这种不确定性.

(6)测试的充分性问题.在软件测试中,充分性准则会影响其揭错能力.目前已经存在几种充分性准则可以度量传统语言的软件测试过程,然而对于面向对象语言的软件测试过程,尚无广泛认同的充分性准则.例如,在测试过程中,假定按照生成的类测试顺序进行测试,是否能达到预想的测试覆盖率,可否对测试中出现的异常情况进行合理的解释;能否检测各模块没有错误连接;能否满足各项功能增长和性能增长的要求;对程序错误的输入有无正确的处理能力等;对软件缺陷发现的时间早晚有多大程度的影响等.

除了东北大学于海等人[29]的方法和北京邮电大学刘颖莲的方法[30]考虑了测试方法自身的揭错能力,其他方法在确定类集成测试顺序的同时,均未考虑软件系统中类节点的测试重要性问题,存在一定的局限性.复杂性高的类引入错误的概率相对较高,应该重点对其进行测试.相反,如果较晚测试那些容易发生错误的需要被重点测试的类节点,则会导致延迟发现错误,进而降低软件系统的测试效率.因此,高效的类集成测试顺序确定方法在软件测试领域具有重要的研究价值.Zhou 等人[62]利用软件复杂性的度量指标对面向对象系统的缺陷进行了预测,通过研究表明软件的复杂性与软件错误的数量、分布和严重级别有着密切的关系.但是缺陷在软件系统中的分布并不均匀,例如,一些简单的底层代码片段也可能出错,并且如果其在系统的多处直接或间接调用这些出错的代码,将会导致整个软件系统网络的崩溃[63].所以我们在确定类的集成测试顺序时不应该忽略错误传播对类集成测试顺序的影响.

因此,虽然降低测试成本是测试技术主要关注的问题,但是不能忽略测试方法自身的揭错能力.

(7)不同类簇间关系的记录.本文探讨的范围是同一类簇内的每个类的情况,后续的工作可以将类簇划分为几个不同的组.同时,为了便于进行后续的类级测试工作,我们可以为不同类簇的相关类添加额外的标记与信息,例如类之间的具体关系等.这将为以后的类级测试做指导作用.

(8)驱动程序的创建.桩模块是指模拟被测试的模块所调用的模块,而不是软件产品的组成部分.创建的驱动程序以主模块作为驱动模块,用桩模块代替与之直接相连的模块.因此,类间集成测试中除了创建桩模块花费一定代价,实际上类间集成时有时还需要创建驱动程序,需要考虑其创建代价,这也是未来研究的内容之一.

对上述问题进行深入的探索是具有一定研究意义的,对软件测试领域的理论与实践方面都有积极的影响.

5 结束语

集成测试中关键的一步是构建合理的类测试顺序,该步骤不仅会影响程序中错误被发现的时间,还会影响测试桩的设计与构建,间接影响了软件测试过程的测试成本,是类集成测试中最重要的问题之一.合适的测试顺序能够极大地节省测试成本.根据本文分析可以看出,基于搜索的类集成测试顺序的生成方法是近五年来学者们研究的重点,已经得到国内外学术界的广泛关注,并取得了一定的研究成果.但是仍有很多问题需要研究,例如现有的评测标准数据集本身的规模、特点,如程序中包含环路的数目、依赖关系的数目等对所采用的算法的影响以及测试方法自身的揭错能力.

本文对类集成测试顺序生成问题的背景、产生原由、分类方式、评测数据集等进行了总结，重点对已有的研究成果进行了分类并展开了系统地分析.对这一问题的深入研究，将帮助软件测试人员进一步提高工作效率.

致　谢　感谢各位审稿专家提出的宝贵意见！

参　考　文　献

[1] Wang Z S, Li B X, Wang L L, Li Q. A brief survey on automatic integration test order generation//Proceedings of the 23rd International Conference on Software Engineering Knowledge Engineering. Eden Roc Renaissance Miami Beach, USA, 2011: 7-9

[2] Mao C, Lu Y. Aicto: An improved algorithm for planning inter-class test order//Proceedings of the 5th International Conference on Computer and Information Technology. Shanghai, China, 2005: 927-931

[3] Kung D C, Gao J, Hsia P, et al. Class firewall, test order, and regression test of object oriented programs. Journal of Object-Oriented Programming, 1995, 8(2): 51-65

[4] Zhang Yan-Mei. Research on Testing Technology of Object-Oriented Programs Based on Dependency Analysis [Ph. D. dissertation]. China University of Mining and Technology, Xuzhou, 2012(in Chinese)
(张艳梅. 基于依赖性分析的面向对象程序测试技术研究[博士学位论文]. 中国矿业大学，徐州，2012)

[5] Jiang Shu-Juan, Zhang Yan-Mei, Li Hai-Yang, Wang Qing-Tan. An approach for inter-class integration test order determination based on coupling measures. Chinese Journal of Computers, 2011, 34(6): 1062-1074(in Chinese)
(姜淑娟，张艳梅，李海洋，王庆坛. 一种基于耦合度量的类间集成测试序的确定方法. 计算机学报，2011，34(6)：1062-1074)

[6] Zhang Yan-Mei, Jiang Shu-Juan, Chen Ruo-Yu, et al. Class integration testing order determination method based on particle swarm optimization algorithm. Chinese Journal of Computers, 2016, on line(in Chinese)
(张艳梅，姜淑娟，陈若玉等. 基于粒子群优化算法的类集成测试序列确定方法. 计算机学报，2016，在线发表)

[7] Li Xiao-Jiang, Li You-Lu, Chen Qi-An. An order-assigned strategy of integration test based on the dynamic dependency relation of classes. Journal of the Academy of Equipment Command & Technology, 2005, 16(1): 93-97(in Chinese)
(李小将，李佑禄，陈启安. 基于类的动态依赖关系的集成测试顺序分配策略. 装备指挥技术学院学报，2005，16(1)：93-97)

[8] Zhang Yan-Mei, Jiang Shu-Juan, Zhang Hong-Chang. An approach for class integration testing based on dynamic dependency relations. Chinese Journal of Computers, 2011, 34(6): 1076-1089(in Chinese)
(张艳梅，姜淑娟，张红昌. 一种基于动态依赖关系的类集成测试方法. 计算机学报，2011，34(6)：1076-1089)

[9] Briand L, Feng J, Labiche Y. Experimenting with genetic algorithms and coupling measures to devise optimal integration test orders. Carleton University, Canada: Technical Report SCE-02-03, 2002

[10] Briand L, Labiche Y, Wang Y. Revisiting strategies for ordering class integration testing in the presence of dependency cycles//Proceedings of the 12th International Symposium on Software Reliability Engineering. Hong Kong, China, 2001, 287-296

[11] Kraft N A, Lloyd E L, Malloy B A, Clarke P J. The implementation of an extensible system for comparison and visualization of class ordering methodologies. Journal of Systems and Software, 2006, 79(8): 1092-1109

[12] Briand L C, Feng J, Labiche Y. Using genetic algorithms and coupling measures to devise optimal integration test orders//Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering. Ischia, Italy, 2002: 43-50

[13] Abdurazik A, Offutt A J. Using coupling-based weights for the class integration and test order problem. The Computer Journal, 2009, 52(5): 557-570

[14] Tai-K C, Daniels F J. Interclass test order for object-oriented software//Proceedings of the 21st International Computer Software and Applications Conference. Washington, USA, 1997: 602-607

[15] Le Traon Y, Jéron T, Jézéquel J-M, Morel P. Efficient object-oriented integration and regression test. IEEE Transactions on Reliability, 2000, 49(1): 12-25

[16] Hanh V L, Akif K, Le Traon Y, Jézéquel J-M. Selecting an efficient oo integration testing strategy: An experimental comparison of actual strategies//Proceedings of the 15th European Conference on Object-Oriented Programming. Budapest, Hungary, 2001: 381-401

[17] Hewett R, Kijsanayothin P, Smavatkul D. Test order generation for efficient object-oriented class integration testing//Proceedings of the 20th International Conference on Software Engineering and Knowledge Engineering. San Francisco, USA, 2008: 703-708

[18] Hewett R, Kijsanayothin P. Automated test order generation for software component integration testing//Proceedings of the IEEE/ACM International Conference on Automated Software Engineering. Auckland, New Zealand, 2009: 211-220

[19] Jaroenpiboonkit J, Suwannasart T. Finding a test order using object-oriented slicing technique//Proceedings of the

14th Asia-Pacific Software Engineering Conference. Nagoya, Japan, 2007: 49-56

[20] Briand L, Labiche Y, Wang Y. An investigation of graph-based class integration test order strategies. IEEE Transactions on Software Engineering, 2003, 29(7): 594-607

[21] Lu Yan-Sheng, Mao Cheng-Ying. Method of inter-class test order determination for object-oriented cluster level testing. Journal of Mini-Micro Computer Systems, 2005, 26(6): 995-999(in Chinese)
(卢炎生, 毛澄映. 面向对象簇级测试中类间测试序确定方法研究. 小型微型计算机系统, 2005, 26(6): 995-999)

[22] Hashim N L, Schmidt H W, Ramakrishnan S. Test order for class-based integration testing of java applications// Proceedings of the 5th International Conference on Quality Software. Melbourne, Australia, 2005: 11-18

[23] Malloy B A, Clarke P J, Lloyd E L. A parameterized cost model to order classes for class-based test of C++ applications //Proceedings of the 14th International Symposium on Software Reliability Engineering. Denver, Colorado, 2003: 353-364

[24] Li Du. Towards test order selection strategy. Computer Engineering and Design, 2008, 29(4): 781-783(in Chinese)
(李都. 测试顺序选择策略研究. 计算机工程与设计, 2008, 29(4): 781-783)

[25] Bansal P, Sabharwal S, Sidhu P. An investigation of strategies for finding test order during integration testing of object oriented applications//Proceedings of International Conference on Methods and Models in Computer Science. Delhi, India, 2009: 1-8

[26] Zhang Yan-Mei, Jiang Shu-Juan, Yuan Guan, et al. An approach of class integration test order determination based on test levels. Software: Practice and Experience, 2015, 45(5): 657-687

[27] Labiche Y, Thévenod-Fosse P, Waeselynck H, Durand M H. Testing levels for object-oriented software//Proceedings of the 22nd International Conference on Software Engineering. Limerick, Ireland, 2000: 136-145

[28] Zhao Yu-Li, Wang Ying, Yu Hai, Zhu Zhi-Liang. An inter-class integration test order generation method based on complex networks. Journal of Northeastern University(Natural Science), 2015, 36(12): 1696-1700(in Chinese)
(赵玉丽, 王莹, 于海, 朱志良. 基于复杂网络的类间集成测试序列生成方法. 东北大学学报(自然科学版), 2015, 36(12): 1696-1700)

[29] 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)

[30] Liu Ying-Lian. Research of Object-Oriented Software Integration Testing Strategy [M. S. dissertation]. Beijing University of Posts and Telecommunications, Beijing, 2013(in Chinese)
(刘颖莲. 面向对象软件集成测试策略研究[硕士学位论文]. 北京邮电大学, 北京, 2013)

[31] Borner L, Paech B. Integration test order strategies to consider test focus and simulation effort//Proceedings of the International Conference on Advances in System Testing and Validation Lifecycle. Porto, Portugal, 2009:80-85

[32] da Veiga Cabral R, Pozo A, Vergilio S R. A Pareto ant colony algorithm applied to the class integration and test order problem//Proceedings of the 22nd IFIP WG 6.1 International Conference on Testing Software and Systems. Natal, Brazil, 2010: 16-29

[33] Vergilio S R, Pozo A, Garcia J C, et al. Multi-objective optimization algorithms applied to the class integration and test order problem. Software Tools for Technology Transfer, 2012, 14(4): 461-475

[34] Assunção W K G, Colanzi T E, et al. Establishing integration test orders of classes with several coupling measures// Proceedings of the 13th Annual Conference Companion on Genetic and Evolutionary Computation. New York, USA, 2011:1867-1874

[35] Assunção W K G, Colanzi T E, Vergilio S R, Pozo A. A multi-objective optimization approach for the integration and test order problem. Information Sciences, 2014, 267(20): 119-139

[36] Wang Zheng-Shan, Li Bi-Xin. Using coupling measure technique and random iterative algorithm for inter-class integration test order problem//Proceedings of the 34th Annual IEEE Computer Software and Applications Conference Workshops. Seoul, Korea, 2010: 329-334

[37] Guizzo G, Fritsche G M, Vergilio S R, et al. A Hyper-heuristic for the multi-objective integration and test order problem//Proceedings of the Genetic and Evolutionary Computation Conference. Madrid, Spain, 2015: 1343-1350

[38] Jungnickel D. Graphs, Networks and Algorithms. Augsburg, Germany: Springer Press, 1998: 781-782

[39] Mao Cheng-Ying. Research on Analysis and Testing Techniques of Object-Oriented Programs [Ph. D. dissertation]. Huazhong University of Science and Technology, Wuhan, 2006(in Chinese)
(毛澄映. 面向对象程序分析与测试技术研究[博士学位论文]. 华中科技大学, 武汉, 2006)

[40] Tarjan R. Depth-first search and linear graph algorithms. SIAM Journal on Computing, 1972, 1(2): 146-160

[41] Li Yuan-Jie, Zhou Guo-Zheng, Liu Feng-Chen. Improved inter-class test sequence algorithm. Computer Engineering, 2010, 36(8): 74-82(in Chinese)
(李远杰, 周国征, 刘凤晨. 一种类间测试顺序改进算法. 计算机工程, 2010, 36(8): 74-82)

[42] Gao Hai-Chang, Feng Bo-Qin, Li Yuan-Jie, Zeng Ming. Improved inter-class integration testing sequence algorithm based on extended ORD. Journal of Mini-Micro Computer Systems, 2007, 28(4): 725-728(in Chinese)
(高海昌，冯博琴，李远杰，曾明. 基于扩展 ORD 图的类间集成测试顺序改进算法. 小型微型计算机系统，2007，28(4)：725-728)

[43] Qi Li-Na. Research and Application of Class Integration Testing Strategy Based on Test Order [M. S. dissertation]. Shanghai Normal University, Shanghai, 2007(in Chinese)
(齐丽娜. 基于测试顺序的类集成测试方法研究与应用[硕士学位论文]. 上海师范大学，上海，2007)

[44] Zhou Yan, Song Jing-Hua. Study on object-oriented integrated testing sequence. Computer Measurement & Control, 2010, 18(9): 2014-2018(in Chinese)
(周燕，宋敬华. 面向对象的集成测试顺序的研究. 计算机测量与控制，2010，18(9)：2014-2018)

[45] Zheng Lei. Layered and Incremental Strategy for Objected-Oriented Integration Testing [M. S. dissertation]. Shanghai Jiaotong University, Shanghai, 2007(in Chinese)
(郑磊. 面向对象集成测试的分层增量测试策略[硕士学位论文]. 上海交通大学，上海，2007)

[46] Shen Xiao-Rong. Object-Oriented Cluster Level Test and Its Application [M. S. dissertation]. Taiyuan University of Technology, Taiyuan, 2005(in Chinese)
(申小荣. 面向对象类簇级测试及其应用[硕士学位论文]. 太原理工大学，太原，2005)

[47] Chen Q, Li X. An order-assigned strategy of classes integration testing based on test level//Proceedings of the 8th International Conference on Computer Supported Cooperative Work in Design. Xiamen, China, 2004: 653-657

[48] Wang Zheng-Shan. Application of hybrid genetic algorithm in object oriented software integration test. Computer Applications, 2008, 28(5): 1341-1343(in Chinese)
(王正山. 混合遗传算法在面向对象的研究中的应用. 计算机应用，2008，28(5)：1341-1343)

[49] Kang Li-Shan, Xie Yun, You Shi-Yong, et al. Non-Numerical Parallel Algorithms: Simulated Annealing Algorithm. Beijing: Science Press, 1994: 56-80

[50] Chen Xiang, Gu Qing, Wang Zi-Yuan, Chen Dao-Xu. Framework of particle swarm optimization based pairwise testing. Journal of Software, 2011, 22(12): 2879-2893(in Chinese)
(陈翔，顾庆，王子元，陈道蓄. 一种基于粒子群优化的成对组合测试算法框架. 软件学报，2011，22(12)：2879-2893)

[51] Yan Lu. Research and Application of Particle Swarm Optimization Algorithm [M. S. dissertation]. University of Electronic Science and Technology of China, Chengdu, 2013 (in Chinese)
(严露. 粒子群算法研究与应用[硕士学位论文]. 电子科技大学，成都，2013)

[52] Doerner K, Gutjahr W J, Hartl R F, et al. Pareto ant colony optimization: A metaheuristic approach to multi objective portfolio selection. Annals of Operations Research, 2004, 131(1-4): 79-99

[53] Dorigom M, Socha K. An introduction to ant colony optimization. Bruxelles, Belgium: Technical Report, TR/IRIDIA/2006-010, 2006

[54] Kalyanmoy D. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimisation: NSGA-II//Proceedings of the 6th International Conference on Parallel Problem Solving from Nature. Paris, France, 2000: 849-858

[55] Deb K, Pratap A, Agarwal S, Meyarivan T. A fast and elitist multi-objective genetic algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation, 2002, 6(2): 182-197

[56] Gendreau M, Potvin J Y. Tabu search//Gendreau M, Potvin J Y, Hillier F S eds. Handbook of Metaheuristics, International Series in Operations Research and Management Science. 2nd Edition. New York, USA: Springer, 2010

[57] Glover F. Future paths for integer programming and links to artificial intelligence. Computers & Operations Research. 1986, 13(5): 533-549

[58] Gong Mao-Guo, Jiao Li-Cheng, Yang Dong-Dong, Ma Wen-Ping. Research on evolutionary multi-objective optimization algorithms, Journal of Software, 2009, 20(2): 271-289(in Chinese)
(公茂果，焦李成，杨咚咚，马文萍. 进化多目标优化算法研究. 软件学报，2009，20(2)：271-289)

[59] Ishibuchi H, Sakane Y, Tsukamoto N, Nojima Y. Evolutionary many-objective optimization by NSGA-II and MOEA/D with large populations//Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics. San Antonio, Texas, USA, 2009: 1758-1763

[60] Zitzler E, Laumanns M, Thiele L. SPEA2: Improving the strength pareto evolutionary algorithm. Technical Report 103, Gloriastrasse 35, CH-8092 Zurich, Switzerland, 2001

[61] Burke E K, Hyde M, Kendall G, et al. A survey of hyper-heuristics. School of Computer Science and Information Technology, University of Nottingham, Nottingham: Technical Report NOTTCS-TR-SUB-0906241418-2747, 2009

[62] Zhou Yu-Ming, Leung H. Empirical analysis of object-oriented design metrics for predicting high and low severity faults. IEEE Transactions on Software Engineering. 2006, 32(10): 771-789

[63] 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)

ZHANG Yan-Mei, born in 1982, Ph. D., lecturer. Her research interests include software analysis and testing.

JIANG Shu-Juan, born in 1966, Ph. D., professor, Ph. D. supervisor. Her research interests include compilation techniques, software engineering, etc.

ZHANG Miao, born in 1992, M. S. Her research interests include software analysis and testing.

JU Xiao-Lin, born in 1976, Ph. D., associate professor. His research interests include software analysis and testing, fault localization, etc.

Background

In the context of object-oriented software, a common problem is the determination of test orders for the integration test of classes, known as the class integration and test order problem. Determination of class integration test order is a key and difficult point in object-oriented software integration testing.

In this paper, first, we simply introduce the background of class test order problem and point out the main contribution of this work. Then, we describe class test order problem, including the origin of class test order problem and the basic idea of generating for class test order; in addition, we also introduce two ways of classification, which are way of breaking loop and way of estimating test stub cost. Next, we classify the existing typical techniques for solving the class integration test order problem, and analyze the research status of these typical techniques completely and systematically; and then we also introduce some evaluation dataset used by typical techniques in empirical studies. Finally, we propose the future research directions of class integration test order and conclude the paper.

The main contribution of this work is the followings: (1) we completely and systematically analyze the research achievements of class integration test order problem in recent years, and collect and analyze about 50 related literatures of home and abroad published in the authoritative journals and conferences. (2) We summarize the evaluation indicators and evaluation dataset used by typical techniques in empirical studies. (3) We add a lot of research literatures at home and abroad in recent years besides the references cited in reference[1].

This work was supported in part by awards from the National Nature Science Foundation of China under Grant Nos. 61502497, 61673384, the Guangxi Key Laboratory of Trusted Software Nos. kx201530, kx201609, the State Key Laboratory for Novel Software Technology at Nanjing University under Grant No. KFKT2014B19, the China Postdoctoral Science Foundation Funded Project under Grant No. 2015M581887, and the Science and Technology Program of Xuzhou under Grant No. KC15SM051.