一种基于耦合度量的类间集成测试序的确定方法

姜淑娟 张艳梅 李海洋 王庆坛
(中国矿业大学计算机科学与技术学院 江苏 徐州 221116)

摘 要 类间测试顺序的确定是类集成测试中难以解决的一个关键问题. 类的测试序列不同,构造相应的测试桩需要花费的代价也会不同. 每一个测试桩复杂度度量的准确性决定最终打破环路所需构造测试桩的总体复杂度. 对于类间测试顺序问题,文章提出一种基于耦合度量的类间集成测试序的确定方法. 采用类间耦合度量与基于图的启发式算法相结合的方法,其中,前者用于度量每一个测试桩的复杂度,后者用于在保证测试桩总体复杂度尽可能小的条件下来打破环路. 首先提出一种在度量中使用的耦合权重的计算方法,对测试桩的复杂度进行新的耦合度量;然后给出一种有效打破环路的基于图的启发式算法;最后实现了类间测试序列自动生成工具——TOGOS. 实验结果表明:文中的方法较现有的方法总体复杂度有明显的降低,从而节约了测试成本.

关键词 集成测试;测试序列;测试桩复杂度;耦合度量;启发式算法
中图法分类号 TP311 **DOI 号**: 10.3724/SP.J.1016.2011.01062

An Approach for Inter-Class Integration Test Order Determination Based on Coupling Measures

JIANG Shu-Juan ZHANG Yan-Mei Li Hai-Yang WANG Qing-Tan
(*School of Computer Science and Technology, China University of Mining and Technology, Xuzhou, Jiangsu* 221116)

Abstract A key and difficult problem of inter-class integration testing is to determine the orders of tested classes. If the orders of tested classes are different, the corresponding costs of stubbing are also different. The overall complexity of stubbing is determined by the accurate measurement of the complexity of each stub. For the class test order problem, this paper presents an approach for inter-class integration test order determination based on coupling measures. The technique combines inter-class coupling measurement and graph-based heuristic algorithm. The former is used to assess the complexity of each stub and the latter is used to break cycles with the minimum overall complexity of stubbing. First, it proposes a computing method of coupling weight used in coupling measure, by which each test stub complexity is reevaluated, then, presents an effective graph-based heuristic algorithm for breaking the cycles of class diagram. Finally, the paper implements a test order automatic generating tool for object-oriented system—TOGOS. The experiment results show that the approach can make the overall stubbing complexity for breaking cycles reduced obviously, and lead to lower integration test cost.

Keywords integration testing; test order; test stub complexity; coupling measures; heuristic algorithm

收稿日期:2010-10-19;最终修改稿收到日期:2011-05-16. 本文得到国家自然科学基金(60970032)、教育部科学技术研究重点项目基金(108063)、江苏省自然科学基金(BK2008124)、江苏省"青蓝工程"、江苏省研究生培养创新工程项目基金(CX10B_157Z)资助.**姜淑娟**,女,1966 年生,博士,教授,博士生导师,主要从事程序设计语言、编译技术、软件工程等方面的教学与研究工作. E-mail: shjjiang@cumt.edu.cn. **张艳梅**,女,1982 年生,博士研究生,主要研究方向为软件分析与测试、异常处理等. E-mail: ymzhang@cumt.edu.cn. **李海洋**,男,1987 年生,硕士研究生,研究方向为软件分析与测试等. **王庆坛**,男,1989 年生,硕士研究生,研究方向为软件分析与测试等.

1 引　言

面向对象程序不像面向过程程序那样有明显的层次化模块结构，对象之间的联系通过消息传递的方式，一条消息会引起连锁反应进而形成一条方法调用链，体现调用关系的静态结构是一个错综复杂的网状结构. 因此，从哪里开始测试以及如何定义一个集成测试顺序是需要进一步研究的问题. 面向过程的基于功能分解的集成测试方法不能满足面向对象程序的集成测试的要求，需要研究满足面向对象程序特点的集成测试方法. 在面向对象软件中，类集成测试顺序问题是指为那些集成测试的类簇确定它们的测试顺序. 确定类簇的测试顺序时，对于在类间关系不构成环路的情况下，可以通过逆向拓扑排序来确定它们的测试顺序；在类间关系构成环路的情况下，则必须首先打破环路才能确定它们的测试顺序. 打破环路是指临时删除一条或多条关联边进而将该类簇构成的有环图变成无环图的过程.

确定类集成测试顺序是在实际的测试中必须要面临的问题，例如单元测试顺序可能受制于那些与其它项目共享的可利用资源，类集成测试顺序对软件开发和单元测试顺序存在一定的影响.

针对类集成测试顺序的问题，很多学者提出了不同的解决方法，总体可以归为两种：第一种是基于遗传算法的方法，即首先确定一个表示类测试顺序的种群，然后通过选择、交叉和变异对这一种群进行处理，并得到满足一定条件的测试顺序. 第二种是基于图论的方法、类簇以及它们之间的关系可以抽象为类图，一个类图是一个有向图，表示为 $G(V,E)$，其中 V 表示类节点集合，E 表示为类之间关系的边集，其中标签 I、Cp、Ag、As 和 Us 分别表示继承、组合、聚集、关联和使用关系. 类集成测试顺序问题就是对图中表示类的各节点进行排序，通过分析有向图的结构来确定满足条件的测试顺序.

类集成测试序列的不同，很大程度上影响着构造相应的测试桩需要花费的测试代价的大小. 测试桩复杂度是用来衡量构造一个测试桩的难易程度，测试桩的总体复杂度则是用来衡量为一个测试序列总体需要构造的相应测试桩的难易程度[1].

类集成测试顺序问题中，测试代价的衡量方法主要有两类，一是最小化集成测试过程当中需要建立的测试桩的数量[2-6]；二是最小化测试桩的总体复杂度[1,7-8]. 后者的精确度更高，因为不同的测试桩各自复杂度不同. 因此，测试桩的数量越少并不能表示一个测试序列需要花费的总体代价越低.

总体来说，为了实现测试桩的复杂度（$SCplx$）（即构成环路的各边权值）的精确度量，减少测试成本，我们需要最小化桩的总体复杂度来找到一个最佳的测试顺序. 因此，需要解决两个关键问题：测试桩的复杂度的计算方法以及打破环路的有效算法.

针对于第一个问题，即在度量测试桩的复杂度时，给出一种新的计算方法，即在进行耦合度量时，根据类间耦合程度分别对属性耦合和方法耦合赋予权值；对于第二个问题，首先找出类图中的强联通组件（$SCCs$），然后采用基于图论的启发式算法打破环路，弥补目前断开继承、组合和聚集等强联系关系增加测试桩的复杂度的缺陷，使最终所构造的测试桩的总体复杂度最小.

2 估算测试桩复杂度的耦合度量方法

通过最小化测试桩的总体复杂度来找到一个最佳的测试顺序，首先要解决的第一个问题是测试桩的复杂度的计算方法，只有在精确评价测试桩的复杂度的基础上才能准确地判断打破环路时所要删除的边集，进而找到最佳的测试顺序.

Briand[8] 在计算复杂度时，对于所使用的耦合度量方法中的属性耦合和方法耦合分配的是相同的权值，因此，考虑的是一种特殊的情况；而 Abdurazik 等人[1] 将耦合分为 9 种类型，不仅对边的权重进行定量的耦合度量，同时也对节点权重进行了度量，综合考虑边和节点的权重，进一步精确度量结果，但是这样增加了算法的复杂度，提高了测试成本.

因此，在度量测试桩的复杂度时，对于耦合度量方法中的属性耦合和方法耦合赋予权值，本文给出了一种新的计算方法.

2.1 相关概念

测试桩复杂度是用来衡量构造一个测试桩的难易程度，测试桩的总体复杂度则是用来衡量为一个测试序列需要构造的总体测试桩的难易程度.

在集成测试过程中，我们常常遇到这样的情况：当需要对某一个类 A 进行测试时，类 A 所依赖的另一个类 B 并没有经过测试. 如果很难在短时间内构建类 B，必定会影响到对类 A 的集成测试. 此时我们需要模拟的对象来替代类 B，我们称该模拟的对象为测试桩. 下面给出具体的定义.

定义 1.　测试桩. 如果类 $C1$ 的一个组件使用

一个或多个类 $C2$ 的服务组件，称为 $C1$ 依赖 $C2$. 增量集成过程中，当 $C1$ 集成时，但 $C2$ 尚未被集成，我们不得不模拟 $C2$ 的服务组件，这个模拟组件通常被称为一个测试桩.

一个测试桩并不是真正的对象，但是能够为待测对象提供感兴趣的数据或状态，这样，待测对象便能够顺利使用依赖对象，或者模拟事件.

定义 2. 代价函数. 计算强连通分量中构成环路的每条边的源类创建的测试桩复杂度的度量函数.

由于测试桩模拟的是待测对象所依赖的类，需要像所用到的真实类那样为待测对象提供感兴趣的数据或状态，因此构造一个测试桩的代价是相当大的. 所以应该尽量避免构造过于复杂类的测试桩. 创建每个测试桩花费一定的代价，用测试桩的总体复杂度衡量创建所有的测试桩花费的总代价. 特别地，当在系统中组件之间的依赖关系不产生环路时不需要创建测试桩. 当创建每一个测试桩花费的代价相同，最小化测试桩的复杂度等价于最小化测试桩的数量.

2.2 耦合度量方法

我们的目标是确定一个测试顺序，使它满足最小化测试桩的总体复杂度. 通过删除满足该条件的边集来打破类图中的环路，进而通过逆向拓扑排序找到一个最佳的集成测试顺序. Briand 等人[8]的耦合度量方法虽然不允许删除继承关系、组合关系和聚集关系这 3 类强联系关系，减少了构建测试桩的复杂度，但是他们为属性和方法分配相同的权值，这是一种特殊的情况，存在着局限性；Abdurazik 等人[1]为属性和方法重新分配了权值，但是他们的度量方法中不仅考虑边的权重，而且增加了节点权重，增加了算法的复杂度，提高了测试成本，因此，为了实现测试桩的复杂度的精确度量，减少测试成本，我们提出一种新的耦合度量方法.

耦合是指两个类相互依赖的一个量度，根据面向对象语言中类间的 5 种关系，对应 5 种耦合类型：继承耦合、组合耦合、聚集耦合、关联耦合、使用耦合.

对于每一种耦合类型，类间耦合信息大都包含 4 部分[1]：(1) 被访问的属性的个数；(2) 被调用的不同方法的个数(包括构造函数)；(3) 返回类型的个数；(4) 传递的参数的个数. 我们依据这 4 个参数来衡量服务类和客户类之间的依赖关系. 为简单起见，用符号“·”将耦合度量的 4 个参数集成到一个表达式，用它来表达两个类 C_i 和 C_i 之间的耦合度量.

式(1)用于表示两个类 C_i 和 C_j 之间的耦合度量(CM).

$$CM(C_i,C_j)=V\cdot M\cdot R\cdot P \tag{1}$$

其中，C_i 和 C_j 代表耦合的两个类，V 代表类 C_i 直接使用的类 C_j 的公有变量的个数，M 代表类 C_i 调用的类 C_j 中方法的个数，R 代表在 M 中出现的不同返回类型个数，P 代表在 M 中出现的不同参数个数.

2.3 估算测试桩复杂度

对于一个复杂度 $\boldsymbol{Cplx}()$，其标准化估算方法记为 $\overline{\boldsymbol{Cplx}()}$. 我们可以将其标准化形式为式(2).

$$\overline{\boldsymbol{Cplx}(i,j)}=\frac{\boldsymbol{Cplx}(i,j)}{\boldsymbol{Cplx}_{\max}-\boldsymbol{Cplx}_{\min}} \tag{2}$$

其中，$\boldsymbol{Cplx}(i,j)$ 表示测试桩复杂度信息矩阵，其中行和列分别表示每个类以及类 i 对类 j 的依赖，$\boldsymbol{Cplx}_{\min}=\mathrm{Min}\{\boldsymbol{Cplx}(i,j),i,j=1,2,\cdots\}$，$\boldsymbol{Cplx}_{\max}=\mathrm{Max}\{\boldsymbol{Cplx}(i,j),i,j=1,2,\cdots\}$.

如式(1)所示，在计算测试桩的复杂度时，耦合度量方法中使用 4 个耦合方法 $V()$、$M()$、$R()$ 和 $P()$，其中 $V()$ 代表调用的变量的个数，$M()$ 代表调用的方法的个数，$R()$ 代表返回类型的个数，$P()$ 代表传递的参数的个数，即一个测试桩的复杂度的估算方法如下[8]：

$$SCplx(i,j)=(W_V\times\overline{V}(i,j)^2+W_M\times\overline{M}(i,j)^2+W_R\times\overline{R}(i,j)^2+W_P\times\overline{P}(i,j)^2)^{1/2} \tag{3}$$

其中，W_V，W_M，W_R 和 W_P 表示权重，并且 $W_V+W_M+W_R+W_P=1$. $\overline{V}(i,j)$，$\overline{M}(i,j)$ 分别根据式(4)，(5)进行计算，$\overline{R}(i,j)$ 和 $\overline{P}(i,j)$ 分别根据式(6)和(7)进行计算. 其中式(4)～(7)的形式如下所示.

$$\overline{V(i,j)}=\frac{V(i,j)}{V_{\max}-V_{\min}} \tag{4}$$

$$\overline{M(i,j)}=\frac{M(i,j)}{M_{\max}-M_{\min}} \tag{5}$$

$$\overline{R(i,j)}=\frac{R(i,j)}{R_{\max}-R_{\min}} \tag{6}$$

$$\overline{P(i,j)}=\frac{P(i,j)}{P_{\max}-P_{\min}} \tag{7}$$

其中，$V(i,j)$、$M(i,j)$、$R(i,j)$ 和 $P(i,j)$ 的值通过开源工具 SOOT① 统计获得. $V_{\max}$、$V_{\min}$，$M_{\max}$、$M_{\min}$，$R_{\max}$、$R_{\min}$ 和 $P_{\max}$、$P_{\min}$ 分别表示统计数据构成的属性耦合矩阵、方法耦合矩阵、返回值耦合矩阵、参数耦合矩阵中的最大最小值.

对于式(3)中的权重 W_V、W_M、W_R 和 W_P，我们

① SOOT. A Java bytecode optimization framework. http://www.sable.mcgill.ca/soot/

给出一种计算方法：

$$W_V = V/(V+M+R+P) \quad (8)$$

$$W_M = M/(V+M+R+P) \quad (9)$$

$$W_R = R/(V+M+R+P) \quad (10)$$

$$W_P = P/(V+M+R+P) \quad (11)$$

对于给定的测试顺序 o，打破 d 条依赖边，则该测试序的总体测试桩的复杂度的计算公式如下：

$$OCplx(o) = \sum_{k=1}^{d} SCplx(k) \quad (12)$$

3　类间测试顺序的确定算法

对于面向对象软件集成测试顺序问题，我们利用 Tarjan 等人的算法[9]识别出类图中由类以及它们之间的依赖关系形成的强连通分量，通过删除那些至少能够构成一个强连通分量的依赖边，利用基于图的启发式算法打破环路.

3.1　打破环路的启发式算法

确定类间测试序的主要问题就是打破环路. Briand 等人[8]解决打破环路问题时，采用的是遗传算法，不仅要精心设置参数，而且算法需要被执行多次，而 Abdurazik 等人[1]综合考虑边的权重、节点权重以及环路的个数来选择边进行删除进而打破环路，增加了算法的复杂度，提高了测试成本.

因此，本文针对该问题给出了一种新的打破环路的方法，把有向边的权重以及使用相应有向边的环路数目作为打破环路的主要依据，其中把测试桩的复杂度作为有向边的权重. 针对确定使用有向边的环路数目的问题，我们提出了一种“查找树”的方法来查找 *SCCs* 中所有的环路. 首先分别将 *SCCs* 中包含的每一个子强连通分量 *SCCi* 中所有边以树的形式组织在一起，规则是将由类编号从小到大表示的各个类依次作为根节点，*SCCi* 中包含的边作为分支进行构造，每选择一个不同的根节点时构造一棵不同的查找树，构造一棵查找树时在一条路径中避免重复的节点，最后以与根节点相同的节点作为叶子节点，直到 *SCCi* 中的边不能满足树中叶子节点与根节点相同为止；然后对这些查找树进行前序遍历，去掉重复的路径，最后得到 *SCCi* 中的环路数目及所有环路及路径、*SCCi* 中各边涉及的环路的数目.

例如，以在第 5 节中实例分析将要使用的 ATM 系统为例，该实例是 Abdurazik 等人在文献[1]以及 Briand 等人在文献[8]中所使用的 ATM 系统. 该系统包含一个强连通分量 *SCC*{8,9,…,15}，该 *SCC* 包含的所有的边如表 4 第 2 列（由于篇幅所限，在这不单独列出）所示. 构造查找树时首先选择 *SCC*{8,9,…,15} 中类节点编号最小的 8 作为根节点进行构造，同理依次选择 9,10,11,…,15 分别作为根节点进行构造. 以 8 作为根节点为例所构造的查找树如图 1 所示.

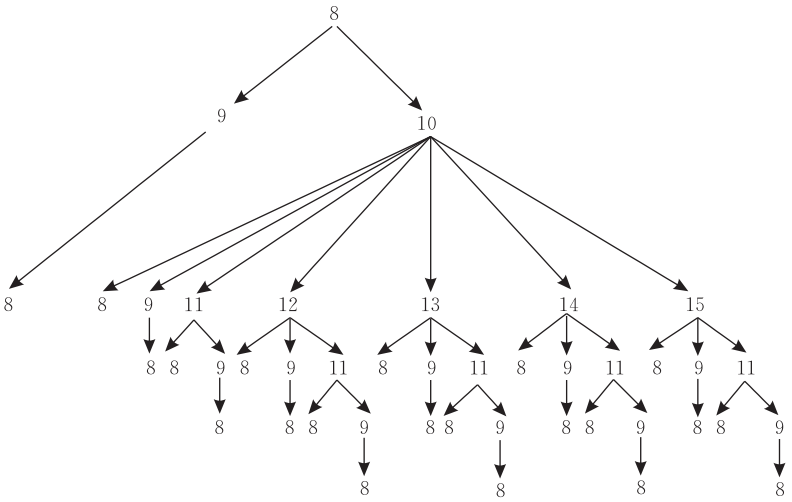

图 1　*SCC* 中以 8 为根节点的环路查找树

然后对查找树进行前序遍历，进而找出所有的环路，最后得到 *SCC* 中的环路个数、各个环路的路径、*SCC* 中各边涉及的环路个数. 本文所采用的环路消除算法如算法 1 所示.

算法 1.　Eliminate loops for *SCCi*.

输入：*SCCi*

输出：环路路径

```
Begin
  1. find all SCCs in class graph;
                          //生成类图中所有的 SCCs
  2. for (each scci(V_scci ; E_scci) ∈ SCCs) do
  3.    search loops(…);        //找出所有的环路
  4.    for (each association or use edge e∈E_scci) do
  5.       find the number of loops that use edge e;
```

```
                                    //找出使用边 e 的环路数目
6.    view the SCplx as edge weights in SCCs;
       //把测试桩的复杂度作为 SCCs 中边 e 的权重
7.    calculate loop-weight ratio;
                                    //计算边 e 的环路-权重比
8.  endfor
9. while (totalLoop!=0) do
10.   remove the edge with highest loop-weight ratio;
                          //删除环路－权重比最高的那条边
11.   totalLoop=totalLoop- number of loops broken;
                          //总的循环数目＝总的循环数目－
                          //被打破的环路的数目
12.   modify the number of loops for remaining edge;
       //修改使用剩余的边集中的边 e 的环路的数目
13.   recalculate the loop-weight ratio for remaining edge;
                        //对剩余的那些边重新计算环路-权重比
14.    if association or use edge e with the same loop-
         weight ratio;
          //有多条环路-权重比相同的关联、使用边
15.    then remove the e with more loops;
                        //删除使得打破环路个数多的那条边
16.     if the number of broken loops same;
                                    //打破的环路个数相同
17.       then remove any edge;
                                    //删除其中的任意一条边
18.     endif
19.    endif
20.  endwhile
21. endfor
End
```

我们的启发式算法的基本思想是首先生成类图中所有的 $SCCs$(行 1),然后针对每一个子 $SCCi$,打破其中的环路(行 2～20). 在打破环路的过程中首先删除环路-权重比最高的关联边或使用边(行 2～10)(Briand 等人[8]、Kung 等人[10]、Kraft 等人[11]都认为给不同类型的边创建测试桩的难易程度的高低和代价 $Cost(C)$ 均存在如下关系:C(关联边)＝C(使用边)＜C(聚集边)≪C(继承边). 因此,称继承关系和聚集关系为强联系关系,使用关系与关联关系均为弱联系关系. 为了减少测试代价,本文消除环路时避免删除强联系边),当有多条环路-权重比相同的关联边或使用边,则删除使得打破环路个数多的那条关联边或使用边(行 14～15),如使得打破的环路个数相同,删除其中的任意一条边(行 16～17).

3.2 确定类间测试顺序的算法

我们的目标是确定一个测试顺序,使它满足测试桩的总体复杂度最小. 在类间关系构成环路的情况下,需要根据耦合度量的测试桩复杂度,再结合环路消除算法来打破环路,进一步确定类测试顺序. 而类测试顺序由类的测试依赖性决定[12]. 测试依赖性表示一个类依赖于其它类的程度. 为了确定类测试顺序,本文从类测试依赖性入手,提出两个测试依赖性定理作为确定类测试顺序的依据.

定理 1. 当类 A 是类 B 的一个子类,或者类 A 是类 B 的一个聚合类,或者类 A 是类 B 的一个关联类,则在集成测试时,类 A 依赖于类 B,类 A 在类 B 之后进行测试.

证明. 类 A 继承于类 B 时,类 A 会继承类 B 的部分属性,从而类 A 依赖于类 B. 若类 B 中被类 A 继承的成员发生变化,或被类 A 继承的成员有直接或间接影响的成员发生变化时,将会影响类 A 的行为.

当类 A 是类 B 的一个聚合类,由于聚合是整体和个体的关系,即若干个类 B 聚合成一个类 A,则类 B 的改变必然会影响类 A,因此类 A 依赖于类 B.

当类 A 是类 B 的一个关联类,则类 A 能够访问类 B 的数据成员,或者类 A 传递一个消息到类 B. 因此,如果类 B 的数据成员发生变化,或者当类 B 接收类 A 发送的消息,并且类 B 的成员函数发生变化,则类 B 对消息的响应会有变化,返回给类 A 的结果也会发生变化. 因此,类 A 依赖于类 B.

因此,类 A 在类 B 之后进行测试. 得出定理 1 的结论. 证毕.

定理 2. 假设 3 个类 A、B、C,当 A 是 B 的一个子类,B 是 C 的一个服务类,即 C 关联于 B(或者依赖于 B)或者 C 是 B 的一个聚合类,并且当 C 不是 A 的聚合类时,则在集成测试时,C 依赖于 B,在不考虑多态性的情况下,C 在 B 之后测试,A,C 的顺序任意,即 B,C,A 或者 B,A,C.

证明. A 是 B 的一个子类,B 是 C 的一个服务类,由测试依赖性定理 1 得出,A 在 B 之后测试,C 在 B 之后测试. 对于 A、C 的测试依赖性关系有如下情况:

(1)若 C 关联于 B(或者依赖于 B). 如果 C 需要访问 B 中的某个数据成员,而 A 继承 B 的该数据成员. 由于我们没有考虑多态性,C 不会访问 A 中的数据成员,则 C 不依赖于 A. 如果 C 需要访问 B 中的某个数据成员,而 A 没有继承 B 的该数据成员. 尽管 A 继承 B,但 C 不会访问 A 中的数据成员,则 C 不依赖于 A. 但由于从 A 到 C 之间没有任何依赖关系,因此,A 也不依赖于 C.

(2)若 C 是 B 的一个聚合类，由于 A 是 B 的子类，而 C 是 B 的聚合类，当 C 不是 A 的聚合类时，C 不依赖于 A，A 也不依赖于 C.

由上述分析得出，则在集成测试时，C 依赖于 B，在不考虑多态性的情况下，C 在 B 之后测试，由于从 A 到 C 之间没有任何依赖关系，则 A，C 的顺序任意，即 B，C，A 或者 B，A，C. 得出定理 2 的结论.

证毕.

确定类测试顺序的算法如算法 2 所示.

算法 2.　Class Integration Test Order.

```
输入：class cluster
输出：class integration test order
Begin
 1. Gen_class graph precedence table();
                              //产生类图的优先级表
 2. Search SCCs;              //找出所有的 SCCs
 3. view each SCCi as one node, make the class graph
    acyclic;
 4. for each acyclic graph do //简化后的无环图
 5.    Find Reverse();        //找到它的逆向拓扑排序
 6.    for each SCCi∈SCCs do
 7.       Eliminate loops for SCCi;
                              //根据算法 1 消除 SCCi 的环路
 8.       GetAllNodeReverse(SCCi);
                              //根据两个定理得到 SCCi 中所有
                              //节点的逆向拓扑排序
 9.    endfor
10. endfor
11. output class integration test order;
                              //输出类集成测试序列
End
```

在该算法中，首先产生类图的优先级表(行 1)，并找出所有的 $SCCs$(行 2)，然后把每一个子 $SCCi$ 看成一个节点，使得整个类图成为无环图(行 3)，对于简化后的无环图，找到它的逆向拓扑排序(行 4～5)；接下来对于每一个子 $SCCi$，根据算法 1 消除 $SCCi$ 的环路，得到 $SCCi$ 中所有节点的逆向拓扑排序(行 6～8)；最后，对无环图进行逆向拓扑排序进而得到所有类集成测试序列(行 11).

3.3　类间测试顺序生成工具

我们根据上述类间测试顺序确定算法设计并实现了一个工具——TOGOS(Test Order Generator for Object-Oriented System)，其系统结构如图 2 所示.

该工具的输入信息是一个描述面向对象系统中类间关系的三元组列表，其中的三元组列表可以由面向对象系统的 UML 设计文档中的类图获得.

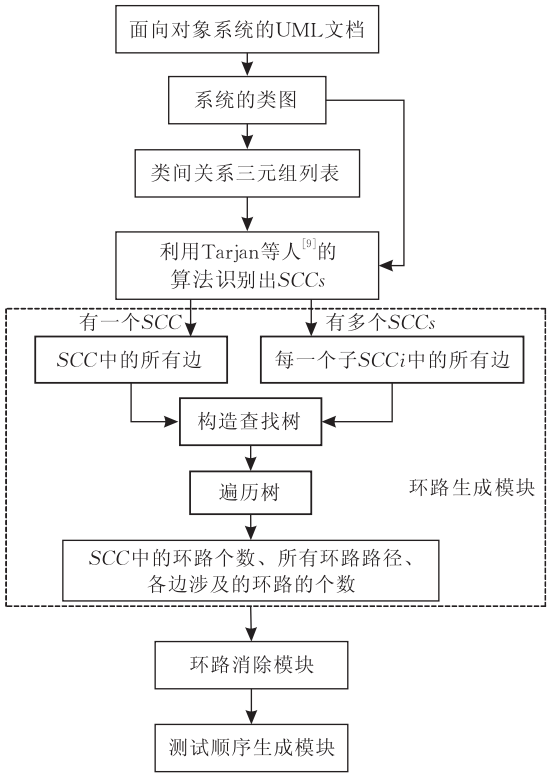

图 2　TOGOS 功能结构图

1)环路生成模块. 如图 2 中虚线框所示. 按照 3.1 节中环路查找方法该模块可以生成所有环路.

2)环路消除模块. 根据输入的关系三元组描述信息所表示的类图，采用 Tarjan 等人[9]的算法，依次找出该类图中的 $SCCi$，按照 3.1 节中的算法 1，通过断开一条或多条关联边(或者使用边)，将每一个子 $SCCi$ 的环路断开，直到类图中所有的环路消除为止，以得到新的无环类图.

3)测试顺序生成模块：按照 3.2 节中的算法 2，对无环图进行逆向拓扑排序输出类集成测试序列.

TOGOS 能够自动生成一个面向对象系统中的类集成测试序列，可减少测试的工作量.

4　实　验

在这一节中，我们通过一组基准程序来进行实验，以验证本文方法的有效性.

4.1　实验描述和过程

在实验中，我们采用了 3 个系统：(1)自动取款机(ATM)模拟系统；(2)ANT 系统；(3)DNS 系统. 其中，ATM 系统在许多软件测试研究中被作为一个基准程序来使用，它包含了 21 个类(忽略了与硬件设备相连的类)，67 条依赖边和大量的依赖环路，

其中,8 个类构成一个包含 30 个环路的 *SCC*[1,8]. ANT 系统——jakarta 项目 V1.4 版本的一部分(http://jakarta.apache.org).实验分析其中构成大量环路的 25 个类,其中,这 25 个类之间存在 83 条依赖边,构成 654 个依赖环路[8].DNS 系统(http://www.xbill.org/dnsjava/)可以提供网络域名服务,包含 61 个类、276 条依赖边,其中,10 个类构成 16 个环路[8].这 3 个系统的类图及其它详细信息可以参照 Briand 等人的文献[8].

SOOT[9]是一个能准确分析 Java 字节码的开源工具,可以提供基本的程序分析功能,例如,类间调用分析.为了统计公式(1)中 4 个参数的值,我们首先通过 SOOT 得到类间调用图,再对该调用图进行遍历,分析类间的调用关系,进一步得到目标数据.

4.1.1 ATM 系统

为了说明 TOGOS 工具的工作原理,我们给出实验的具体过程.首先以 ATM 系统为例进行分析,表 1 是 ATM 系统中包含的所有类.

表 1 ATM 系统

类编号	类	类编号	类
1	ReceiptPrinter	12	WithdrawalTransaction
2	Display	13	DepositTransaction
3	Keyboard	14	TransferTransaction
4	CardReader	15	InquiryTransaction
5	OperatorPanel	16	GUILayout
6	EnvelopeAcceptor	17	QuestionDialog
7	CashDispenser	18	ATMMain
8	ATM	19	ATMApplet
9	Bank	20	Money
10	Session	21	Status
11	Transaction		

首先,以式(1)的形式所表示的 ATM 系统中强连通分量 *SCC*{8,9,10,11,12,13,14,15}的各条边所对应的 *CM* 值如表 2 所示.每个 *CM* 中的 4 个耦合度量参数值是利用 SOOT 统计获得的.

表 2 *SCC*{8,9,10,11,12,13,14,15}各边的 *CM* 值

类编号	8	9	10	11	12	13	14	15
8		0.1.0.0	0.2.2.3					
9	0.7.1.3							
10	0.7.4.4	0.2.1.1		0.0.5.3	0.2.1.0	0.2.1.0	0.2.1.0	0.2.1.0
11	0.2.1.3	0.1.0.0	0.2.1.0					
12	0.4.3.11	0.4.3.11	0.2.2.0	0.0.0.0				
13	0.4.3.6	0.4.3.14	0.2.2.0	0.0.0.0				
14	0.3.2.6	0.3.3.12	0.2.2.0	0.0.0.0				
15	0.2.1.6	0.3.3.10	0.2.2.0	0.0.0.0				

然后,TOGOS 工具中环路生成模块可以自动生成 *SCC*{8,9,10,11,12,13,14,15}中所包含的 30 个环路,如表 3 所示.

表 3 *SCC*{8,9,10,11,12,13,14,15}中的环路

序号	边	序号	边
1	8→9→8	16	10→14→11→10
2	8→10→8	17	10→15→11→10
3	10→11→10	18	8→10→11→9→8
4	10→12→10	19	8→10→12→9→8
5	10→13→10	20	8→10→13→9→8
6	10→14→10	21	8→10→14→9→8
7	10→15→10	22	8→10→15→9→8
8	8→10→9→8	23	8→10→12→11→8
9	8→10→11→8	24	8→10→13→11→8
10	8→10→12→8	25	8→10→14→11→8
11	8→10→13→8	26	8→10→15→11→8
12	8→10→14→8	27	8→10→12→11→9→8
13	8→10→15→8	28	8→10→13→11→9→8
14	10→12→11→10	29	8→10→14→11→9→8
15	10→13→11→10	30	8→10→15→11→9→8

表 4 给出的是分别采用 Briand 等人[8]、Abdurazik 等人[1]以及本文方法对 *SCC*{8,9,10,11,12,13,14,15}中各边的源类创建的测试桩复杂度的度量结果,即采用耦合度量的方法得到的测试桩复杂度.其中,第 1 列为构成 *SCC* 的各个类的编号;第 2 列为通过 TOGOS 工具统计的 *SCC* 中各边;第 3 列为统计的 *SCC* 中各边涉及的环路个数;第 4、5 两列分别为采用 Briand 等人[8]、Abdurazik 等人[1]构造的测试桩复杂度;最后一列是采用本文方法,使用表 2 中的值,通过式(3)~(7)计算的删除 *SCC* 各边时为各边的源类构造的测试桩复杂度.其中,式(3)的 W_V、W_M、W_R 和 W_P 的值由式(8)~(11)进行计算.

接下来是环路消除模块所完成的工作:把测试桩的复杂度作为边的权值,根据算法 2 对表 4 中各边进行环路-权重比的简单计算,边 8→10 的环路-权重比最高,首先删除该边,可以打破 20 个环路,剩余 10 个环路.然后,再重新计算表 4 中剩余各边的环路-权重比,经过计算,11→10 环路-权重比最高,删除,可以打破 5 个环路,剩余 5 个环路.同理,以下依次打破的边为 8→9,10→12,10→13,10→14 和 10→15,至此,所有的环路都被打破,类图成为无环图.

表 4　*SCC*{8,9,10,11,12,13,14,15}中各边的权值

序号	边	涉及的环路个数	V&M（Briand 等人的方法）[8]	V&M&R&P（Abdurazik 等人的方法）[1]	V&M&R&P（本文方法）
1	8→9	1	0.71	1.00	0.14
2	8→10	20	0.53	1.22	0.32
3	9→8	11	1.00	1.17	0.89
4	10→8	1	1.00	1.66	0.81
5	10→9	1	0.74	1.13	0.23
6	10→11	3	∞	5.00	0.72
7	10→12	6	0.23	1.13	0.23
8	10→13	6	0.23	1.13	0.23
9	10→14	6	0.26	1.13	0.23
10	10→15	6	0.21	1.13	0.23
11	11→8	5	0.74	1.17	0.29
12	11→9	5	0.71	1.00	0.14
13	11→10	5	0.53	1.13	0.23
14	12→8	1	0.81	1.60	0.96
15	12→9	1	0.81	2.59	0.96
16	12→10	1	0.53	2.01	0.28
17	12→11	3	∞	5.00	∞
18	13→8	1	0.81	1.50	0.59
19	13→9	1	0.81	2.62	1.28
20	13→10	1	0.53	2.01	0.28
21	13→11	3	∞	5.00	∞
22	14→8	1	0.77	1.39	0.53
23	14→9	1	0.77	2.58	1.08
24	14→10	1	0.53	2.01	0.28
25	14→11	3	∞	5.00	∞
26	15→8	1	0.74	1.29	0.57
27	15→9	1	0.77	2.58	0.86
28	15→10	1	0.53	2.01	0.28
29	15→11	3	∞	5.00	∞

最后将本文方法与文献[1,8]中已有的两种方法进行了比较. ATM 系统度量结果的比较如表 5 所示. 其中，第 1 列为 3 种不同的方法；第 2 列为分别采用这 3 种不同的方法删除的边的个数；第 3 列为被删除的各条边；第 4 列为被删除的各条边的复杂度；第 5 列为打破环路需要构造的测试桩的总体复杂度 $OCplx$；第 6 列为 3 种不同的算法需要执行的次数. 其中，采用 Briand 遗传算法中删除边所在的列为趋于稳定状态时所删除的 7 条边.

表 5　ATM 系统的度量结果比较

方法	删除边个数	删除边	各边复杂度 $SCplx$	总体复杂度 $OCplx$	算法执行次数
Briand 等人的方法[8]	7	8→9	0.71	2.70	多次
		8→10	0.53		
		11→10	0.53		
		10→12	0.23		
		10→13	0.23		
		10→14	0.26		
		10→15	0.21		
Abdurazik 等人的方法[1]	7	8→9	0.71	2.70	1 次
		8→10	0.53		
		11→10	0.53		
		10→12	0.23		
		10→13	0.23		
		10→14	0.26		
		10→15	0.21		
本文方法	7	8→10	0.32	1.61	1 次
		11→10	0.23		
		8→9	0.14		
		10→12	0.23		
		10→13	0.23		
		10→14	0.23		
		10→15	0.23		

4.1.2　ANT 系统

对于 ANT 系统，我们采用同样的方法进行实验. 该系统的类图可以参见文献[8]，其中所包含的类如表 6 所示. 首先，以式(1)的形式所表示的 ANT 系统中强连通分量 *SCC*{2,4,10,16,17,18,19,20,21,22,23,24}的各条边所对应的 CM 值如表 7 所示.

表 6　ANT 系统

类编号	类	类编号	类
1	AntClassLoader	14	NoBannerLogger
2	BuildEvent	15	PathTokenizer
3	BuildException	16	Project
4	BuildListener	17	ProjectComponent
5	BuildLogger	18	ProjectHelper
6	DefaultLogger	19	RuntimeConfigurable
7	DemuxOutputStream	20	Target
8	DirectoryScanner	21	Task
9	FileScanner	22	TaskAdapter
10	IntrospectionHelper	23	TaskContainer
11	Launcher	24	UnknownElement
12	Location	25	XmlLogger
13	Main		

表 7　*SCC*{2,4,10,16,17,18,19,20,21,22,23,24}中各边的 *CM* 值

类编号	2	4	10	16	17	18	19	20	21	22	23	24
2				0.1.0.0				0.1.1.0	0.2.1.0			
4	0.3.0.6											
10				0.3.2.0					0.0.0.1		0.0.0.1	
16	0.5.0.6	0.7.0.7			0.1.0.1				0.5.0.4	0.1.0.1		
17				0.2.0.2								
18			0.6.1.12	0.14.1.3			0.4.0.3	0.5.3.0	0.6.0.0	0.1.1.0	0.1.0.0	0.2.0.0
19				0.1.0.2		0.3.0.10						
20				0.5.2.7		0.1.1.3	0.2.0.1					0.1.1.2
21				0.4.0.6			0.2.0.1	0.0.2.0				
22				0.1.0.0								
23									0.0.2.5			
24			0.2.2.4	0.3.1.3			0.3.2.2			0.1.1.0	0.1.1.0	

表 8 给出的是采用本文方法对 *SCC*{8,9,10,11,12,13,14,15}中各边的源类创建的测试桩复杂度的耦合度量结果,即各边权重.

表 8 *SCC*{2,4,10,16,17,18,19,20,21,22,23,24}中各边的权值

序号	边	V&M&R&P(本文方法)	序号	边	V&M&R&P(本文方法)	序号	边	V&M&R&P(本文方法)	序号	边	V&M&R&P(本文方法)
1	2→16	0.07	11	16→21	0.35	21	18→24	0.14	31	22→16	0.07
2	2→20	0.24	12	16→22	0.08	22	19→16	0.14	32	23→21	0.50
3	2→21	0.22	13	17→16	0.16	23	19→18	0.74	33	24→10	0.41
4	4→2	0.43	14	18→10	0.83	24	20→16	0.53	34	24→16	0.25
5	10→16	0.45	15	18→16	0.89	25	20→18	0.25	35	24→19	0.39
6	10→21	0.08	16	18→19	0.27	26	20→19	0.13	36	24→22	0.24
7	10→23	0.08	17	18→20	0.67	27	20→24	0.21	37	24→23	0.24
8	16→2	0.44	18	18→21	0.43	28	21→16	0.43			
9	16→4	0.54	19	18→22	0.24	29	21→19	0.13			
10	16→17	0.08	20	18→23	0.07	30	21→20	0.67			

使用 TOGOS 工具打破环路的过程如表 9 所示.其中,第 1 列和第 2 列分别表示被处理的 *SCCi* 的序号以及 *SCCi* 本身;第 3 列为 *SCCi* 中包含的环路的个数;第 4 列为处理 *SCCi* 时所删除的边.最后,由于 Abdurazik[1] 没有对 ANT 进行分析,这里仅对采用本文方法与文献[8]中的方法的度量结果进行了比较.ANT 系统度量结果的比较如表 10 所示.

表 9 打破环路的过程

序号	处理的 *SCCi*	包含的环路个数	处理 *SCCi* 时所删除的边
1	*SCC*{4,10,18,19,20,2,16,17,21,22,24,23}	654	19→18
2	*SCC*{4,10,18,20,2,16,17,21,22,24,23,19}	306	20→18
3	*SCC*{4,10,24,20,2,16,17,21,22,23,19}	135	20→24
4	*SCC*{4,16,2,17,21,20,22,23,19}	50	19→16
5	*SCC*{4,16,2,17,21,20,22,23}	33	17→16
6	*SCC*{4,16,2,20,21,22,23}	22	21→16
7	*SCC*{4,16,2,20,21,22,23}	12	20→16
8	*SCC*{4,16,2,22}&&*SCC*{20,21,23}	3&&2	2→16&&21→20
9	*SCC*{16,22}	1	16→22
10	End	0	——

表 10 ANT 系统的度量结果

方法	被删除的边的个数	被删除的边	总体复杂度 *OCplx*	算法执行次数
Briand 等人的方法[8]	12	17→16, 21→16, 21→19, 21→20, 20→16, 20→18, 20→19, 20→24, 4→2, 16→2, 16→22, 19→18	3.59	多次
本文方法	10	19→18, 20→18, 20→24, 19→16, 17→16, 21→16, 20→16, 2→16, 21→20, 16→22	3.28	1 次

4.1.3 DNS 系统

对于 DNS 系统,所采用的方法与前两个实例的方法相同,这里我们不对过程进行重复描述.并且对采用本文方法与文献[8]中的方法的度量结果进行了比较.如表 11 所示.

表 11 DNS 系统的度量结果

方法	被删除的边的个数	被删除的边	总体复杂度 *OCplx*	算法执行次数
Briand 等人的方法[8]	6	21→8,21→11,32→48,32→58,38→33,52→33	1.47	多次
本文方法	6	33→52,21→11,33→38,32→48,21→8,32→58	1.36	1 次

4.2 实验结果和分析

由表 5 可以发现,本文方法与采用 Briand 等人[8]的遗传算法和 Abdurazik 等人[1]的基于图的算法产生不同的结果:他们的方法中删除了 7 条边[1,8],总体测试桩复杂度为 2.70,而本文同样也删除了 7 条边,但总体测试桩复杂度为 1.61,那么构造的总体测试桩的复杂度降低了(2.70−1.61)/2.70=40.4%.最后,通过 TOGOS 工具自动生成 ATM 系统的类测试顺序,如表 12 所示.其中,第 1 列为主测试顺序号,第 2 列为被测试的类,第 3 列为被测类所依赖的类,即在被测类之前进行测试的类.对于主测试顺序号相同的类,它们之间没有依赖关系,其测试顺序可以是任意的.因此,表 12 中的测试顺序并不是唯一的测试顺序.

表 12　ATM 系统类测试顺序

主测试顺序号	被测类	被测类所依赖的类
1	6,16,20,21	——
2	1	16,20
	2	16
	7	20
	17	16
3	3	2,16,20
	4	17,16
	5	17,16,20
4	8	1,16,20,2,3,4,17,5,6,7
5	9	8,1,16,20,2,3,4,17,5,6,7,21
6	11	8,1,16,20,2,3,4,17,5,6,7,9,21,11
	18	8,1,16,20,2,3,4,17,5,6,7,9,21
	19	8,1,16,20,2,3,4,17,5,6,7,9,21
7	10	8,1,16,20,2,3,4,17,5,6,7,9,21,11
8	12	8,1,16,20,2,3,4,17,5,6,7,9,21,10,11
	13	8,1,16,20,2,3,4,17,5,6,7,9,21,10,11
	14	8,1,16,20,2,3,4,17,5,6,7,9,21,10,11
	15	8,1,16,20,2,3,4,17,5,6,7,9,21,10,11

Abdurazik 等人在文献[1]中仅对 ATM 系统进行了分析，而对于 ANT 系统未作任何分析. 因此，对于 ANT 系统，本文方法仅与 Briand 等人[8]的遗传算法进行比较. 同理，由表 10 可以发现，本文方法与 Briand 等人[8]的遗传算法产生不同的结果：他们的方法中删除了 12 条边[8]，总体测试桩复杂度为 3.59，而本文删除了 10 条边，总体测试桩复杂度为 3.28，那么构造的总体测试桩的复杂度降低了 (3.59−3.28)/3.59＝8.6%. 通过 TOGOS 工具自动生成 ANT 系统的类测试顺序：(9, 12, 15, 17), 3,(8,19),21,23,20,2,4,(5,16),(1,6,7,10,22, 25),(11,14,24),18,13. 即共 13 个主测试序. 其中，括号中的类表示它们的主测试顺序号相同.

对于 DNS 系统，由表 11 可以发现，本文方法与 Briand 等人[8]的遗传算法产生不同的结果：他们的方法中删除了 6 条边，总体测试桩复杂度为 1.47，而本文也删除了 6 条边，但总体测试桩复杂度为 1.36，那么构造的总体测试桩的复杂度降低了 (1.47−1.36)/1.47＝7.5%. TOGOS 工具自动生成 DNS 系统的类测试顺序如表 13 所示.

表 13　DNS 系统类测试顺序

主测试顺序号	被测类	被测类所依赖的类
1	9,10,13,15,17,27, 28,31,36,44,46,49	——
2	5	28
	47	46
	53	15,27,31,36
3	21	5,28,49
4	8	21,5,28,49
	14	21,5,28,49,14
	22	21,5,28,49,46,47
5	32	8,21,5,28,49,10,36,44,46
6	1	8,21,5,28,49,17,32,10,36,44,46
	2	8,21,5,28,49,17,32,10,36,44,46
	4	8,21,5,28,49, 32,10,36,44,46
	6	8,21,5,28,49, 32,10,36,44,46
	16	8,21,5,28,49, 32,10,36,44,46
	18	8,21,5,28,49, 13,32,10,36,44,46
	19	8,21,5,28,49, 32,10,36,44,46
	23	8,21,5,28,49, 32,10,36,44,46
	25	8,21,5,28,49, 32,10,36,44,46
	26	8,21,5,28,49, 32,10,36,44,46
	29	8,21,5,28,49, 32,10,36,44,46
	34	8,21,5,28,49, 32,10,36,44,46
	39	8,21,5,28,49, 32,10,36,44,46
	40	8,21,5,28,49, 32,10,36,44,46
	43	8,21,5,28,49, 31,32,10,36,44,46
	45	8,21,5,28,49, 32,10,36,44,46
	48	8,21,5,28,49, 32,10,36,44,46
	56	8,21,5,28,49, 32,10,36,44,46
	58	8,21,5,28,49, 32,10,36,44,46
7	7	8,21,5,28,49,25,32,10,36,44,46
	11	8,21,5,28,49,25,32,10,36,44,46
	20	10,21,5,28,49,25,8,32,36,44,46
	24	8,21,5,28,49,25,32,10,36,44,46
	30	8,21,5,28,49,25,32,10,36,44,46
	35	13,21,5,28,49,25,8,32,10,36,44,46
	41	8,21,5,28,49,31,32,10,36,44,46,58
	54	8,21,5,28,49,46,56,32,10,36,44
	55	8,21,5,28,49,10,15,25,32,36,44,46,27,29,43,31,53
	57	8,21,5,28,49,46,56,32,10,36,44

（续 表）

主测试顺序号	被测类	被测类所依赖的类
8	37	7,8,21,5,28,49,25,32,10,36,44,46, 11,35,13
	42	10,21,5,28,49,25,8,32,36,44,46,31,43,55,15,27,29,53
	59	35,13,21,5,28,49,25,8,32,10,36,44,46
	61	55,8,21,5,28,49,10,15,25,32,36,44,46,27,29,43,31,53
9	51	7,8,21,5,28,49,25,32,10,36,44,46,9,11,13,15,20,22,47,31,33,35,37,39,55,27,29,43,53,59
	60	21,5,28,49,55,8,10,15,25,32,36,44,46,27,29,43,31,53,61
10	12	7,8,21,5,28,49,25,32,10,36,44,46,9,11,14,31,35,13,37,51,15,20,22,47,39,55,27,29,43,53,59,60,61
	33	55,8,21,5,28,49,10,15,25,32,36,44,46,27,29,43,31,53,60,61
	50	7,8,21,5,28,49,25,32,10,36,44,46,9,11,20,22,47,35,13,37,39,55,15,27,29,43,31,53,60,61
11	3	4,8,21,5,28,49,32,10,36,44,46,12,7,25,9,11,14,31,35,13,
		37,51,15,20,22,47,39,55,27,29,43,53,59,60,61,30
	38	14,21,5,28,49,15,25,8,32,1036,44,46,29,31,33,55,27,43,53,60,61,39,42,48
12	52	14,21,5,28,49,31,33,55,8,10,15,25,32,36,44,46,27,29,43,53,60,61,38,39,42,48

接下来需要对生成的类测试顺序的有效性进行验证.本文以 ATM 系统为例,其测试序的正确性验证如下.

证明. 根据测试依赖性定理 1 可知,类 6,16,20,21 不依赖于任何其它类,则它们最先进行测试;类 11 是 12,13,14,15 的父类,则类 11 先于类 12,13,14,15 的测试,而类 12,13,14,15 之间没有依赖关系,则它们的测试顺序是任意的;而类 11 依赖于类 8,9,10,20,21,而 11→10 依赖在打破环路时已被删除,则 8,9,20,21 先于类 11 的测试;类 8 是类 1～7 的聚集类,则类 1～7 先于类 8 的测试;同理,类 7 和 9 是类 20 的聚集类,类 10 是类 11 的聚集类,类 11 是类 20 的聚集类,类 18 和 19 是类 8 和 9 的聚集类,则类 20 先于类 7,9 和 11 测试,类 11 先于类 10 的测试,类 8 和 9 先于类 18 和 19 的测试;其它剩余的那些弱联系关系的类测试顺序的正确性验证方法同上. 证毕.

本文的耦合度量方法以及启发式算法是有效的;同时,从实验结果可以看出,与采用 Briand 等人[8]和 Abdurazik 等人[1]的方法相比,求解质量有所提高.原因如下:首先,与 Abdurazik 等人[1]的方法相同,本文实验中计算 W_V、W_M、W_R 和 W_P 的值时,一个类 C_i 直接使用的类 C_j 的公有变量的个数均为 0,体现了类的封装性.然而,计算时所使用的式(1)中的 M、R、P 的值则与 Abdurazik 等人[1]的不同,我们的方法是直接使用类 C_i 调用的类 C_j 中方法的个数,在被调方法中出现的不同返回类型个数以及在被调方法中出现的不同参数个数,因为它们可以直接反映两个相互作用的类之间的耦合度.而 Abdurazik 等人[1]首先在软件实现过程中将其分为 9 种不同的耦合类型,然后对于每一种耦合类型,定义相应的耦合度量来衡量服务类和客户类之间的依赖关系,此外,他们同时对边的权重和节点权重进行定量的耦合度量,增加了算法的复杂度.计算式(4)中 $\overline{V}(i,j)$ 的值时,我们所使用的 V、$V_{\max}$ 和 $V_{\min}$ 的值可以在文献[8]中查找.其次,在 Briand 等人[8]的方法中,耦合度量方法中的属性耦合和方法耦合分配的是相同的权值,考虑的是一种特殊的情况,此外,他们忽略了返回值和传递的参数对耦合度量的影响,具有一定的局限性.我们在克服现有方法存在的问题并结合他们优势的基础上给出了本文方法.对测试桩的复杂度进行度量时,我们给出了一种新的方法,在打破环路时,弥补了断开继承、组合和聚集等强联系关系导致的测试桩复杂度提高的缺陷,采用基于图论的启发式算法,只需运行一次,便能够确定类间测试顺序,并满足花费总体测试桩复杂度最小.同时,类间测试序列自动生成工具 TOGOS 的开发,实现了自动化测试,减少了测试的工作量,提高了测试效率.

5 相关研究

在面向对象程序的测试中,Kung 等人[10]最早提出了解决类测试顺序问题的方法,并证明如果对象关系图中没有环,则可以通过逆向拓扑排序得到类间测试顺序.如果类图中有环,则首先识别其中的强联通分量,然后删除部分关联边使之成为无环图,这一过程意味着需要开发测试桩,而开发测试桩是一项成本很高的工作[11].减少开发测试桩成本的方法主要有两类:一类是最小化所需测试桩的数目,例如:Tai 等人[5]提出的测试顺序分配策略导致构造大量多余的测试桩;Le Traon 等人[6]采用了一种基于测试依赖图模型的方法进行集成测试,测试依赖图是由类和方法之间的测试依赖关系构成的,他们的策略最优化了桩的数目,但可能打破继承和聚集关系;Briand 等人[2]在不打破继承、聚集等强联系关

系的前提下，给出基于图论的测试顺序策略，通过最小化测试桩的数目找到一个最佳测试顺序，减少了所需测试桩数目. 另一类是最小化总体测试桩的复杂度，例如：Briand 等人[8] 和 Abdurazik[1] 等人以及文献[7]分别使用了不同的耦合度量方法估算测试桩的复杂度，其中，文献[7]使用的是耦合度量和随机交互算法解决类测试顺序问题，他们允许断开继承和聚集等强关联关系；Briand 等人[8] 的方法虽然避免断开继承和聚集等强关联关系，但是在计算测试桩的复杂度时，对于耦合度量方法中的属性耦合和方法耦合分配的是相同的权值，因此，考虑的是一种特殊的情况；而 Abdurazik 等人[1] 首先对边的权重和节点权重进行定量的耦合度量，进一步精确了度量结果，然后采用基于图论的方法，根据边的权重、节点权重以及环路的个数这 3 个因素来打破环路，但是增加了算法的复杂度，提高了测试成本.

因为不同的测试桩各自复杂度不同，因此，测试桩的数量越少并不能表示一个测试序列需要花费的总体代价越低. 因此，我们采用的是最小化总体测试桩的复杂度的方法.

总体来看，解决类集成测试顺序问题时，对于测试桩的复杂度的度量方法，现有的方法一部分是采用断开继承和聚集关系的方法所导致测试桩复杂度的增加[6-7]，而另一部分由于考虑因素过多使度量方法过于繁琐[1]；对于打破环路的方法，采用 GA 方法必须执行很多次，使过程复杂化，基于图论的方法一部分是采用断开继承和聚集关系[6-7] 的方法，一部分是由于在测试桩复杂度的度量阶段的不精确性导致最终打破环路所需构造测试桩的总体复杂度的提高. 因此，我们对测试桩的复杂度进行度量时，给出一种简单普遍的方法，在打破环路时，弥补了断开继承、组合和聚集等强联系关系导致的测试桩复杂度提高的缺陷，采用基于图论的启发式算法，只需运行一次，便能够确定类间测试顺序，并满足测试桩的总体复杂度最小.

6　结束语

类间测试序的研究是类簇级测试的一个难点. 本文通过实验证明，当把测试桩的复杂度作为权值，我们的耦合度量方法以及启发式算法与 Briand 的和 Abdurazik 的耦合度量方法相比，更加精确地度量测试桩复杂性，性能较优，降低了测试桩的总体复杂度，遗传算法需要多次运行，而我们的方法只需运行一次，从而降低了测试代价. 在此基础上开发的 TOGOS 工具，能够自动生成一个面向对象系统中的类集成测试序列，提高了测试顺序生成的速度，减少了测试的工作量.

可以发现，本文中我们没有考虑抽象类的特点，实际上，抽象类的特性，将会影响类间的依赖性，进而将影响类间测试顺序. 这是我们下一步要解决的问题之一. 此外，现有的类间分析只限于静态依赖分析，而类间的动态依赖关系比较普遍，增加打破动态依赖边的复杂度，但是这样会带来新的问题：动态依赖关系是由于静态依赖关系而在执行阶段存在的关系，当打破静态依赖关系时，一些动态依赖关系会随之消失，对于没有消失的动态依赖关系，在何种情况下删除这些动态依赖关系来打破环路，如何为动态依赖关系构造测试桩以及动态依赖边权重的计算方法是我们下一步要解决的主要问题.

致　谢　在此，我们向对本文给予建议的同行表示感谢. 同时，对审稿人提出的有益建议表示感谢！

参　考　文　献

[1] 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

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

[3] Hanh V L, Akif K, Traon Y L, Jezequel 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, LNCS 2072. Springer-Verlag, 2001: 381-401

[4] 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

[5] Tai K C, Daniels F. Test order for inter-class integration testing of object-oriented software//Proceedings of the 21st International Computer Software and Applications Conference. Washington, DC, USA, 1997: 602-607

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

[7] Wang Z, Li B. 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

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

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

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

[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] Labiche Y. Incremental class testing from a class test order. Carleton University, Technical Report SCE-05-06, 2005

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

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

LI Hai-Yang, born in 1987, M. S. candidate. His research interests include software analysis and testing.

WANG Qing-Tan, born in 1989, M. S. candidate. His research interests include software analysis and testing.

Background

Class integration testing is an important part in object-oriented software testing, and the determination of class order is a key and difficult problem of class integration testing. For integration testing problem, testers need to find test sequences of classes in order to execute interactions. One major problem of test order is the presence of cyclic dependency calls. Many researchers have proposed techniques to solve this problem by removing relationships to break cycles and then create test stubs. If the orders of tested classes are different, the corresponding costs of stubbing are also different. An appropriate test order for software testing can reduce test cost. The overall complexity of stubbing is determined by the accurate measurement of the complexity for each stub.

However, the current solutions lack an effective coupling measure technique to estimate test stub complexity and an effective algorithm to break cycles. Overall, in the existing coupling measure method of stubbing complexity, some allow removing inheritance and aggregation to break cycles, which lead to the increase of stubbing complexity; others consider measure factors too much, which make the measurement method too complicated. For cycles breaking problem, there are mainly two solutions. One is to use genetic algorithm, which greatly complicate the process with many times run; the other is graph-based method, which only need run once.

The goal of this study is to design an optimal test order with the minimum overall complexity of stubbing. This paper presents an approach for inter-class integration test order determination based on coupling measures. The technique combines inter-class coupling measurement and graph-based heuristic algorithm. It is proved that the method makes the overall stubbing complexity reduced obviously and improves the test efficiency obviously. The main contribution of this work is that implements the accurate measurement of the complexity of stubs; in addition, the approach can find the accurate test order of classes with the minimum overall complexity of stubbing.

This work was supported in part by awards from the National Natural Science Foundation of China under grant No. 60970032, the Key Project of Chinese Ministry of Education under grant No. 108063, Natural Science Foundation of Jiangsu Province under grant No. BK2008124, Qing Lan Project, and Graduate Training Innovative Projects Foundation of Jiangsu, China under grant CX10B_157Z.