基于粒子群优化算法的类集成测试序列确定方法

张艳梅[1),2)] 姜淑娟[1),3)] 陈若玉[1)] 王兴亚[1)] 张妙[1)]

[1)](中国矿业大学 计算机科学与技术学院, 江苏 徐州 221116)

[2)](计算机软件新技术国家重点实验室(南京大学), 江苏 南京 210093)

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

摘 要 类测试序列的确定是类集成测试中一个难以解决的关键问题。合理的类集成测试序列可以降低构造测试桩的总体复杂度，降低测试代价。提出一种基于粒子群优化算法的类集成测试序列确定方法。首先，对所有类进行排列组合生成所有可能的类测试序列，并将每个类测试序列看成一个粒子并映射到一维空间，用空间中的每一个位置代表一个类集成测试序列；然后，根据适应度函数计算每个粒子的速度和位置，再通过粒子群优化算法选择粒子的最优位置和最优适应度，得到最优粒子；最后，根据映射关系，将选择的最优粒子映射为其对应的类测试序列，则该测试序列即为所求得的最优类测试序列。实验结果表明，采用本文方法求得的类测试序列花费更小的测试代价，本文方法更有效。

关键词 测试序列；面向对象；集成测试；粒子群优化算法；一维空间

中图法分类号 TP311

Class Integration Testing Order Determination Method Based on Particle Swarm Optimization Algorithm

ZHANG Yan-Mei[1), 2)] JIANG Shu-Juan[1),3)] CHEN Ruo-Yu[1)] WANG Xing-Ya[1)] ZHANG Miao[1)]

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

[2)](State Key Laboratory for Novel Software Technology (Nanjing University), Nanjing 210093, China)

[3)](Guangxi Key Laboratory of Trusted Software, Guilin, 541004, China)

Abstract Class integration testing is an important part in object-oriented software testing, and it is a key and difficult problem to determine the class integration test order of class cluster in integration testing. Reasonable class integration test order can reduce the overall complexity of test stub, and reduce test cost. A class integration test order determination method based on particle swarm optimization algorithm is proposed. First, all possible classes test orders are generated through permutation and combination, and each class test order is taken as a particle and is mapped to one dimensional space, and then each position in dimensional space represents a integration test order; Then, we calculate the velocity and position of each particle according to fitness function, and then choose the optimal position and the optimal fitness of the particles by particle swarm optimization algorithm, and obtain the optimal particle; Finally, according to the mapping relationship, we get the test order

本课题得到国家自然科学基金（61502497）、南京大学计算机软件新技术国家重点实验室开放课题(KFKT2014B19)、广西可信软件重点实验室研究课题 (kx201530)、中国博士后科学基金项目(2015M581887)、徐州市科技计划项目（KC15SM051）资助. 张艳梅，女，1982年生，博士，讲师，主要研究领域为软件分析与测试，E-mail: ymzhang@cumt.edu.cn. 姜淑娟（**通讯作者**），女，1966年生，博士，教授，博士生导师，CCF高级会员（E200015801M），主要研究领域为编译技术、软件工程等，E-mail: shjjiang@cumt.edu.cn. 陈若玉，女，1990年生，硕士，主要研究领域为软件分析与测试，E-mail: ruoyuchen2008@163.com . 王兴亚，男，1990年生，博士生，主要研究领域为软件分析与测试，软件错误定位等，E-mail: yizitianxianjian@163.com . 张妙，女，1992年生，硕士生，主要研究领域为软件分析与测试等，E-mail: miaozhang@cumt.edu.cn .

that the optimal particle is corresponding to, which is the optimal test order. The optimal test order makes the minimum overall complexity of test stub and the minimum test cost. The experimental results show that the proposed approach takes a lower test stub cost for solving the class test order problem, which is more effective.

Key words test order; object-oriented; integration testing; particle swarm optimization algorithm; one-dimensional space

1 引言

软件测试是保证软件质量的重要手段。面向对象程序的测试通常由方法级测试、类级测试、类簇级测试和系统级测试四部分组成。经过方法级和类级的测试后，每个模块都能正常运行，但是要保证模块与模块可以相互合作，需要进行类簇级测试，也称集成测试。集成测试中一个重要的工作是确定类测试序列。不同的类测试序列，需要不同的测试桩代价。这是因为，模块与模块之间存在相互关联关系，当对一个模块进行测试时，需模拟其调用其它模块的功能，模拟的功能模块称为测试桩。合理的类测试序列可以减少测试桩的总体复杂度，进而减少测试的代价。

类测试序列的确定问题，需要考虑影响测试桩的各种因素，因此，可以将求解类测试序列的问题转化为优化问题。Briand 等人[1]在一定规模的初始测试序列下，利用遗传算法选择最优的测试序列。但是由于该方法初始测试序列的局限性导致解空间不够大，对测试序列的交叉和变异影响测试序列的选择的速度和精确度。粒子群优化算法（Particle Swarm Optimization，PSO）和遗传算法都属于人工智能算法。其中，PSO 是一种启发式算法，简单高效，在软件测试领域已经显出较大的优势，已成功应用于解决许多组合优化问题，如旅行商问题，顺序排序问题，集合覆盖问题等。

鉴于此，本文提出一种基于粒子群优化算法的类集成测试序列的确定方法，将粒子群优化算法应用到类集成测试序列的确定问题中。首先，对所有类进行排列生成所有可能的类测试序列；然后，将每个类测试序列看成一个粒子，利用粒子群优化算法选择最优的粒子；最后，将生成的最优粒子映射为类测试序列。该方法在保证足够大的解空间时，利用粒子群优化算法的设置参数少、收敛速度快、精确度高等优点得到满足条件的类集成测试序列。实验结果表明粒子群优化算法求得类测试序列具有更小的测试桩代价，更快的选择速度。

本文的主要贡献如下：

(1) 提出将类测试序列映射到一维空间的方法，以及将选择的最优粒子映射为类测试序列的方法。

(2) 提出适应度函数构造方法，用于评价粒子（类测试序列）的优劣。

(3) 提出一种基于粒子群优化算法的类集成测试序列确定方法。实验结果表明，该方法求得的类测试序列花费更小的测试桩代价。

2 基本定义

测试桩复杂度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用来衡量一个测试序列需构造的所有测试桩的难易，也用来衡量生成一个测试序列所花费的测试代价的高低[1]。

定义 1. 测试桩[2] 给定两个类 i 和 j，i 依赖于 j。在进行增量集成测试的过程中，当对 i 进行测试时，如果 j 没有被测试，则需要模拟 j 的功能来保证 i 的测试可以正常进行。此时，该模拟组件就为测试桩。

测试桩并不是真正的对象，仅为待测的对象提供数据或者状态，以使待测对象可以正常进行测试。

定义 2. 属性依赖[1] 如果目标类 i 声明的属性引用（指向）源类 j 的属性，则称类 i 和 j 存在属性依赖。其中，类 i 和类 j 属性依赖值称为属性复杂度，记为 $A(i, j)$，通常包括目标类调用（1）源类方法参数（Parameter Dependence，简称 PD）；（2）源类方法返回值（Return Dependence，简称 RD）；（3）源类成员变量（Member Dependence，简称 MD）。

定义 3. 方法依赖[1] 若一个测试序列 O 中打破循环时源类 i 调用目标类 j 的方法，则称类 i 和 j 之间存在方法依赖。其中，类 i 和类 j 方法依赖值称为方法复杂度，记为 $M(i, j)$，是源类调用目标类的方法的数量（包括构造方法），记为 ID。

定义 4. PSO 算法[3] 在 PSO 算法中，设种群含有的粒子数目为 N，搜索空间维度为 D，用 $X_i = (x_{i1}, x_{i2}, \cdots, x_{iD})$表示第 i 个粒子，该粒子经历的最好位置（个体极值）记为 $P_i = (p_{i1}, p_{i2}, \cdots, p_{iD})$，也称为

pbest。种群中所有粒子经历的最好位置（全局最优解）记为 gbest。粒子 i 的速度用 $V_i = (v_{i1}, v_{i2}, \cdots, v_{iD})$ 表示。粒子速度和位置的更新公式如下：

$$v_i^{t+1} = wv_i^t + c_1r_1(pbest - x_i^t) + c_2r_2(gbest - x_i^t) \quad (1)$$

$$x_i^{t+1} = x_i^t + v_i^{t+1} \quad (2)$$

其中，c_1 和 c_2 均为常数，r_1 和 r_2 为(0, 1)之间的随机数，w 是取值范围在(0.4, 0.9)间的一个惯性权重，且将 w 的取值根据迭代次数依次递减。

3 基于粒子群优化的类间集成测试序列生成

3.1 方法的总体框架

基于粒子群优化算法的类集成测试序列生成方法流程如图 1 所示，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，将类测试序列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序列，则该测试序列就是所求得的最优测试序列。最优测试序列使得构建测试桩的总体复杂度最小，测试花费的代价最小。

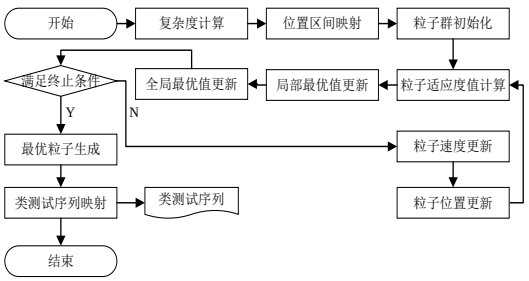

图 1 方法流程图

3.2 测试序列的映射

利用 PSO 算法求解问题时，需要定义算法的解空间作为粒子的位置区间，解空间中的每一个位置代表问题的一个解。具体地说，首先将类测试序列映射到一维空间，用空间中的每一个位置代表一个类集成测试序列。在进化过程中，不断进化寻找更好的位置，从而获得最优的粒子。当求得最优粒子时，由于仍需要知道该最优粒子所对应的确切的类序列，因此需要将生成的最优粒子映射为类测试序列，即所求的最优的类集成测试序列，以便指导开发测试人员工作。

（1）将类集成测试序列映射为粒子位置

对 Java 源程序进行静态分析，在获得类图的基础上，把类图中的类映射到一维空间，使得空间中的每一个位置代表一个类集成测试序列。把类映射到一维空间算法如算法 1 所示。

算法 1. 类集成测试序列映射为粒子位置.

```
输入：list_citos 类集成测试序列
list_classes 应用类链表，链表中类按照依赖数降序排列
输出：position 粒子位置
1. BEGIN
2. position = 0;
3. n = set_classes.size();
4. FOR (int i = 0; i < n; i ++)
// n 是待测试类的数目，i 为第 i 个需要被测试的类
5.     class = list_citos.get(i);
6.     index = list_classes.getIndex(class);
7.     position = position + index * (n-i-1)!;
8. ENDFOR
9. END
```

算法 1 描述类集成测试映射为粒子位置算法的主要步骤。首先，算法第 2 行对粒子位置进行初始化和第 3 行获得数据结构中类的个数。其次，算法第 4-8 行是对 n 个类进行粒子位置的确定。算法第 5、6 行获的第 i 个类在 list_classes 中的位置，进而获得第 i 个类的 index，通过算法第 7 行求得一个类集成测试序列中第 i 个位置的类信息。以上步骤得到一个类集成测试序列在一维空间粒子的位置。

（2）将粒子的位置映射为类集成测试序列

通过粒子群优化算法获得最优粒子，此粒子的位置是所求测试桩总体复杂度最小的类集成测试序列。通过粒子位置确定类集成测试序列的算法如算法 2 所示：

算法 2. 粒子位置映射为类集成测试序列.

```
输入：position 粒子位置
list_classes 应用类链表，链表中类按照依赖数降序排列
输出：list_citos 类集成测试序列
1. BEGIN
2. size = set_classes.size();
3. FOR(int i = 1; i < n; i ++)
// n 是待测试类的数目，i 为第 i 个需要被测试的类
4.     index = position / (n-i)!;
5.     class = list_classes.get(index);
```

```
6.    add class to list_cito;
7.    remove class from list_classes;
8.    position = position % (n-i)!;
9. ENDFOR
10. class = list_classes.get(0);
11. add class to list_citos;
12. END
```

算法 2 描述粒子位置映射为类集成测试序列的主要步骤。首先，通过算法第 2 行产生类图的优先级表 list_classes，优先级表根据类的属性个数和方法个数之和从大到小进行排序；然后，通过算法第 3-9 行计算粒子对应的类集成测试序列，算法第 4 行是通过对应位置信息除以该位置的类排列数，获得该类在应用链表中的位置并从链表中移除（第 5、7 行）该类，算法第 6 行将该类添加到链表 cito 中，通过算法第 8 行获得剩余类的位置信息，循环求得剩余类的位置信息。当链表 list_classes 只剩最后一个类时，算法第 10-11 行将最后一个类加入链表 list_cito 中。

3.3 适应度函数设计

3.3.1 耦合度量

Briand 等人[1]打破环路时不允许删除强依赖关系（继承关系、组合关系和聚集关系），减小了构建测试桩复杂度的代价；同时，为了与该方法进行比较，本文在粒子进化过程中也不允许删除强依赖关系。本文采用耦合度量的方式衡量粒子进化过程中打破环路所花费的代价，并采用 PSO 算法确定一个类集成测试序列，使它满足构造测试桩所花费的总体复杂度最小。耦合是指两个类之间依赖性的一个度量，根据面向对象程序的类间关系，耦合类型可以分为：继承耦合、组合耦合、聚集耦合、关联耦合和使用耦合。

对于每一种耦合类型，类间耦合信息包含以下两部分[4]：（1）属性依赖；（2）方法依赖。我们用耦合度量的方式来计算测试序列的代价，如 3.3.2 节所示。

3.3.2 估算测试桩复杂度

对于一个复杂度 *Cplx()*，其标准化方法记为 $\overline{Cplx()}$，其标准化形式为式(3)[4]：

$$\overline{Cplx()}=\frac{Cplx(i,j)}{Cplx_{\max}-Cplx_{\min}} \tag{3}$$

其中，*Cplx*(*i, j*) 是一个矩阵，表示复杂度，行和列都是类，类 *i* 依赖于类 *j*，需要计算 $Cplx_{\min}=Min\{Cplx(i,j),i,j=1,2,...\}$ 和 $Cplx_{\max}=Max\{Cplx(i,j),i,j=1,2,...\}$。矩阵 $Cplx(i,j)$ 中最小的复杂度的值等于 0。公式（3）可以写为[4]：

$$\overline{Cplx()}=\frac{Cplx(i,j)}{Cplx_{\max}} \tag{4}$$

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*)可以通过标准化的加权几何平均计算[4]，如下：

$$SCplx(i,j)=(W_A\times\overline{A(i,j)^2}+W_M\times\overline{M(i,j)^2})^{1/2} \tag{5}$$

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，且 $W_A+W_M=1$。

对于一个给定的测试序列 *O*，则测试序列的整体复杂度为[4]：

$$Ocplx(o)=\sum_{i=1,j=1}^{n}SCplx(i,j) \tag{6}$$

公式(6)中，*n* 代表类集成测试序列中类的个数。对于类集成测试序列，通过对 $Ocplx(o)$ 计算来求得类集成测试序列测试桩的总体代价。

采用 PSO 算法在求解类集成测试序列的问题时，通过适应度函数来评价粒子的优劣，其中粒子代表类集成测试序列，而测试序列通过测试桩总体复杂度来衡量，所以在对粒子衡量时选择类集成测试序的测试桩总体复杂度。适应度函数表示方法如公式(7)所示[4]。其中 *fitness* 表示适应度函数。等式右边代表测试桩总体复杂度。

$$fitness=Ocplx(o)=\sum_{i=1,j=1}^{n}SCplx(i,j) \tag{7}$$

本节在耦合度量的基础上，设计 PSO 算法中适应度函数的表示方法，通过适应度函数来衡量粒子的优劣。

3.4 示例分析

本节通过一个示例程序详细说明类集成测试序列与粒子位置之间的映射关系，图 2 所示为示例程序 SimpleHospital。它包含 Hospital、Doctor、Patient 等 3 个类，此时根据排列数公式有 A_3^3，共 6 种类集成测试序列。根据第 2 节的定义，类间的属性依赖关系和方法依赖关系如表 1 所示，其中类 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。

```
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 2 示例程序 SimpleHospital

表 1 示例程序属性依赖关系和方法依赖关系矩阵

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

将类集成测试序列映射为粒子位置时，首先按照类依赖个数进行降序排列，生成应用类链表 list_classes={Hospital, Doctor, Patient}。给定一个类集成测试序列 list_citos={Doctor, Hospital, Patient}，根据算法 1，该测试序列对应位置为：

$$position(list\_citos)=\sum_{i=0}^{2} index[list\_citos(i)]\times(3-i-1)=2$$

将 SimpleHospital 所有可能的类集成测试序列映射为粒子位置后，需要计算寻找最优粒子。随机选取一组粒子作为初始种群，进化计算每个粒子的适应度值，从而记录每个粒子的最优位置和种群中粒子的最优位置。在计算粒子的适应度时，首先需要通过算法 2 找到该粒子对应的类集成测试序列，然后通过公式(7)计算得到该测试序的测试桩复杂度作为粒子的适应度。以粒子位置为 2 时为例，首先根据算法 2 可以有如下计算：

$$list\_citos(0)=list\_classes(2/(3-1)!)=Doctor$$

$$list\_citos(1)=list\_classes(0/(3-2)!)=Hospital$$

$$list\_citos(2)=Patient$$

此时该粒子对应类集成测试序列 *list_citos* 为：Doctor、Hospital、Patient。通过公式(7)计算该测试序的测试桩复杂度。与 Briand 等人[1]的研究相同，本文为属性复杂度和方法复杂度赋予相同的权重，因此 $W_A=W_M=0.5$。此时，该测试序列的测试桩复杂度为：

$$\begin{aligned}
&Ocplx(list_citos)\\
&=\sum_{i=1,j=1}^{3} SCplx(i,j)\\
&=\sum_{i=1,j=1}^{3}\left(\frac{1}{2}\times\overline{A(i,j)^2}+\frac{1}{2}\times\overline{M(i,j)^2}\right)\\
&=\left(\frac{1}{2}\times\overline{A(1,1)^2}+\frac{1}{2}\times\overline{M(1,1)^2}\right)+\left(\frac{1}{2}\times\overline{A(1,2)^2}+\frac{1}{2}\times\overline{M(1,2)^2}\right)+\left(\frac{1}{2}\times\overline{A(1,3)^2}+\frac{1}{2}\times\overline{M(1,3)^2}\right)+\\
&\left(\frac{1}{2}\times\overline{A(2,1)^2}+\frac{1}{2}\times\overline{M(2,1)^2}\right)+\left(\frac{1}{2}\times\overline{A(2,2)^2}+\frac{1}{2}\times\overline{M(2,2)^2}\right)+\left(\frac{1}{2}\times\overline{A(2,3)^2}+\frac{1}{2}\times\overline{M(2,3)^2}\right)+\\
&\left(\frac{1}{2}\times\overline{A(3,1)^2}+\frac{1}{2}\times\overline{M(3,1)^2}\right)+\left(\frac{1}{2}\times\overline{A(3,2)^2}+\frac{1}{2}\times\overline{M(3,2)^2}\right)+\left(\frac{1}{2}\times\overline{A(3,3)^2}+\frac{1}{2}\times\overline{M(3,3)^2}\right)\\
&=\left(\frac{1}{2}\times 0+\frac{1}{2}\times 0\right)+\left(\frac{1}{2}\times 2^2+\frac{1}{2}\times 0\right)+\left(\frac{1}{2}\times 1^2+\frac{1}{2}\times 0\right)+\\
&\left(\frac{1}{2}\times 0+\frac{1}{2}\times 0\right)+\left(\frac{1}{2}\times 0+\frac{1}{2}\times 0\right)+\left(\frac{1}{2}\times 1^2+\frac{1}{2}\times 2^2\right)+\\
&\left(\frac{1}{2}\times 0+\frac{1}{2}\times 0\right)+\left(\frac{1}{2}\times 0+\frac{1}{2}\times 0\right)+\left(\frac{1}{2}\times 0+\frac{1}{2}\times 0\right)\\
&=5
\end{aligned}$$

表 2 示例程序进化信息

粒子位置	类集成测试序列	测试桩复杂度
0	Hospital、Doctor、Patient	7
1	Hospital、Patient、Doctor	6.5
2	Doctor、Hospital、Patient	5
3	Doctor、Patient、Hospital	2.5

4	Patient、Hospital、Doctor	4
5	Patient、Doctor、Hospital	2

同理，通过上述步骤，我们可以计算得到粒子经过其它所有可能位置时对应的类测试序列信息及测试桩复杂度信息，如表 2 所示。当粒子进化满足终止条件时，寻优过程结束，此时将最优粒子对应的类集成测试序列作为计算结果。在本示例中，当进化经历位置 5 时，方法寻到粒子的最优位置，测试桩复杂度为 2，即类集成测试序列的测试桩复杂度最小的位置。此时，该粒子对应的类集成测试序列 *list_citos* 为：Patient、Doctor、Hospital。

3.5 类集成测试序列生成工具

根据上述类集成测试序列的算法设计并实现一个工具——GenCITO（Generation of Class Integration and Test Order Based on Particle Swarm Optimization Algorithm），其系统结构如图 3 所示。

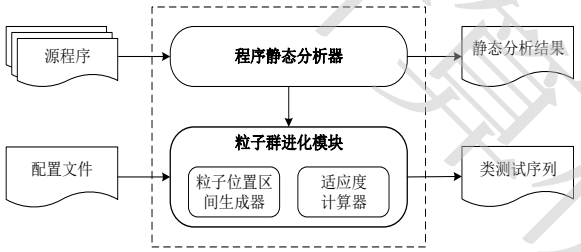

图 3 系统结构图

该工具的输入信息是源程序和配置信息（包括迭代次数，初始种群等）。

（1）程序静态分析模块：首先，利用开源工具 soot—对源程序进行分析，获取类间依赖关系，主要包括包名、类名、成员变量信息、成员方法信息等；其次，通过 Soot 统计获得类间属性复杂度的值和方法复杂度的值。

（2）粒子群进化模块：在程序静态分析的基础上，对类排列，将类测试序列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，通过适应度函数对粒子进行评价，向最优的粒子方向进化。

（3）测试序列生成模块：采用粒子群优化算法生成最优的粒子，并将该最优粒子映射为对应的类集成测试序列。

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

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

4 实验

实验部分以开源程序为实验对象，提出三个研究问题，收集实验结果并加以分析，以验证本文方法的有效性。

4.1 实验对象

选取 ATM、ANT、SPM、BCEL、DNS、JBoss、JHotDraw 和 MyBatis 等八个系统开展实验。

ATM[1]是一个自动取款机模拟系统。ANT[1]是基于 Java 平台的开源代码，是 Jakarta 程序中的一部分。SPM[1]是一个保安巡逻监控系统。BCEL[1]是一个方便用户对 Java 类文件分析、创建和操纵的系统等。DNS[1]是提供网络域名服务的系统。Briand 等人[1]的文献可以提供这五个系统的类图和类间关系等详细信息。表 3 给出了第一组实验对象的详细信息，其中列 2 到列 8 分别给出了实验程序的类数、使用关系数、关联数和聚合数、组合数、继承数、环路数、代码行数等信息。其中，如 3.3.1 节所述，在粒子进化过程中的限制条件是不允许删除组合和继承等强依赖关系，其它类型的关系可以被删除。ATM，ANT，SPM 和 BCEL 系统有合适的类（类个数在 19 和 45 之间），有不同数目的环路（ATM 的 30 个环路到 BCEL 的 416091 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。以上表明每个应用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。对于 ATM、ANT、SPM、BCEL 和 DNS 这五个系统，我们以类间关系作为输入（不需要将源程序作为输入信息），通过 GenCITO 工具自动生成类测试序列。

JBoss、JHotDraw 和 MyBatis 系统的类个数在 133 和 428 之间，详细信息如表 3 所示。与前五个实验系统不同的是，JBoss、JHotDraw 和 MyBatis 这三个系统，我们没有再现基于图论（D）[5]的方法，而是以各个系统的源程序作为输入，然后通过 GenCITO 工具自动生成类测试序列，不需要知道环路的个数，因此对于这三个系统我们没有统计环路个数，用“-”表示。

表 3 八个系统详细信息

系统	类数	使用关系数	关联和聚合数	组合数	继承数	环路数	代码行数
ATM	21	39	9	15	4	30	1390
ANT	25	54	16	2	11	654	4093

SPM	19	24	34	10	4	1178	1198
BCEL	45	18	226	4	46	416091	3033
DNS	61	211	23	12	30	16	6710
JBoss[a]	133	431	216	41	164	-	13710
JHotDraw[b]	411	3236	3974	457	287	-	117157
MyBatis[c]	428	1917	1755	338	183	-	25687

a http://www.jboss.org/jbossas/downloads.html –subsystem used: management of JBoss (version 6.0.0M5): is a Java application server

b http://sourceforge.net/projects/jhotdraw/ – package used: org.jhotdraw.draw of JHotDraw (version 7.5.1): framework for the creation of drawing editors

c http://code.google.com/p/mybatis/downloads/list: data mapper framework that makes easier the use a relational database with OO application (version 3.0.2)

表 4 表示前五个系统属性复杂度和方法复杂度的详细信息[1]。其中，第 2、3 列表示属性复杂度取值范围和平均值，第 4、5 列表示方法复杂度的范围和平均值。

表 4 耦合分布信息

系统	属性复杂度		方法复杂度	
	范围	平均值	范围	平均值
ATM	[1,13]	6.15	[0,7]	1.79
ANT	[0,31]	8.36	[0,14]	2.52
SPM	[1,16]	7.53	[0,8]	2.33
BCEL	[0,20]	1.89	[0,7]	1.52
DNS	[0,12]	3.36	[0,10]	1.46

4.2 研究问题

为了检验本文提出的基于粒子群优化的类测试序列生成方法（PSO 方法）的有效性，与基于图论（D 方法）[5]、基于遗传算法（包括基于属性复杂度和方法复杂度具有相同权重的遗传算法的方法（GA 方法）[1]、基于仅有属性复杂度的遗传算法的方法（A 方法）[1]、基于仅有方法复杂度的遗传算法的方法（M 方法）[4]）和基于随机迭代算法的类集成测试序列生成方法（RIA 方法）[6]进行比较，并设计以下三个问题：

（1）与 D 方法、GA 方法，A 方法，M 方法和 RIA 方法相比，本文方法是否可以找到测试代价更小的类集成测试序列？

（2）与 GA 方法，A 方法，M 方法和 RIA 方法相比，在相同条件下（如种群数量，迭代次数），本文方法是否可以更快找到相同或更优的类集成测试序列？

（3）与 D 方法、GA 方法，A 方法，M 方法和 RIA 方法相比，本文方法的方法复杂度和属性复杂度是否更低？

4.3 实验参数设置

粒子群优化算法需要考虑对初始种群的规模、迭代次数、速度的取值范围、W_A、W_M 等这些参数进行设置。其中，初始种群的规模影响粒子群取得最优解的快慢和是否能够避免陷入局部最优解。若初始种群数量过大，会增加对适应度函数评估的时间。Briand 等人[1]指出初始种群的规模在 25 到 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，若粒子群运动速度过慢，容易陷入局部最优解。因此，本文粒子群速度的取值范围为[0, n!/n]，其中，n 为待测系统中类的个数。

采用粒子群优化算法解决类集成测试序列问题时，实验参数设置是十分必要的。结合 Briand 等人[1]提出的基于遗传算法的方法在求解类集成测试序列时的参数设置情况，我们针对所使用的实验对象规模的不同，设置不同的参数。对于前五个系统，初始种群设置为 100，对于 JBoss 系统，初始种群设置为 300，对于 JHotDraw 和 MyBatis 系统，初始种群设置为 1000。具体的参数设置如表 5 所示。

表 5 参数设置

初始种群	迭代次数	W_A	W_M	速度取值范围
100	500	0.5	0.5	[0, n!/n]
300	500	0.5	0.5	[0, n!/n]
1000	500	0.5	0.5	[0, n!/n]

4.4 实验结果及分析

由于遗传算法、随机迭代算法和粒子群优化算法求解的随机性，实验中我们重复执行算法 30 次。

（1）测试桩复杂度

首先开展 4.2 节描述的第一组实验。该组实验比较了本文方法和其它方法生成的类集成测试序列的测试桩复杂度高低。实验结果如表 6 和图 4 所示。其中，表 6 的列 2-列 6 分别给出 D 方法、GA 方法、A 方法、M 方法，RIA 方法以及 PSO 方法在 ATM、ANT、SPM、BCEL、DNS、JBoss、JHotDraw 和 MyBatis 这八个系统中的类集成测试序列的测试桩复杂度，其中，每个系统的第一行是执行算法 30 次所得的复杂度的最小值到最大值的范围，第二行是 30 次的测试桩复杂度平均值。图 4 的横轴表示

所用实验对象，即 ATM、ANT、SPM、BCEL、DNS、JBoss、JHotDraw 和 MyBatis 这八个系统，纵轴表示 D 方法、GA 方法、A 方法、M 方法，RIA 方法以及 PSO 方法分别执行算法 30 次所得的测试桩复杂度平均值。

与前五个系统不同的是，JBoss、JHotDraw 和 MyBatis 系统没有与 D 方法进行比较。原因有以下两点：首先，D 方法要想充分考虑并估算需要构建测试桩的具体成本，如类中属性的数量、调用的数量以及各种约束等所有这些因素是比较困难的[7]，由此 D 方法得到的解往往是次优的；其次，对于大规模系统，模拟图论的方法具有相当的复杂性，难以实现。

表 6 测试桩复杂度

Systems	D	GA(W_A=W_M=0.5)	A(W_A=1)	M(W_M=1)	RIA	PSO
ATM	[2.68-4.18]	[2.50-6.30]	[2.46-5.92]	[1.71-3.71]	[2.17-11.74]	[2.17-3.60]
	3.44	3.71	3.81	2.49	6.03	2.49
ANT	[3.97-6.42]	[2.43-5.00]	[1.65-4.19]	[2.00-3.86]	[2.44-13.23]	[1.78-2.34]
	4.63	3.46	2.74	2.70	6.75	2.13
SPM	[5.82-9.02]	[3.45-6.59]	[2.57-6.71]	[3.00-6.13]	[3.63-13.16]	[2.45-3.34]
	6.42	5.03	4. 72	4.36	8.07	2.90
BCEL	[8.59-10.28]	[10.31-14.84]	[2.00-5.90]	[12.00-16.43]	[10.10-26.22]	[4.88-10.22]
	9.22	12.33	3.75	14.70	17.82	7.30
DNS	[1.47-1.99]	[6.44-13.32]	[4.08-13.92]	[4.10-9.10]	[5.15-40.31]	[6.27-8.72]
	1.73	9.94	9.16	6.72	21.56	7.11
Jboss	-	[42.91-45.88]	[24.04-24.36]	[39.95-44.30]	[46.32-60.41]	[42.48-43.63]
	-	43.99	26.4	41.53	56.02	42.92
JHotDraw	-	[80.34-92.73]	[85.71-93.24]	[93.64-97.57]	[62.94-125.22]	[81.73-84.36]
	-	85.63	90.48	96.68	98.36	82.74
MyBatis	-	[79.18-86.36]	[83.15-85.17]	[85.56-88.83]	[70.12-104.72]	[71.02-72.25]
	-	81.76	86.49	88.57	90.23	71.91

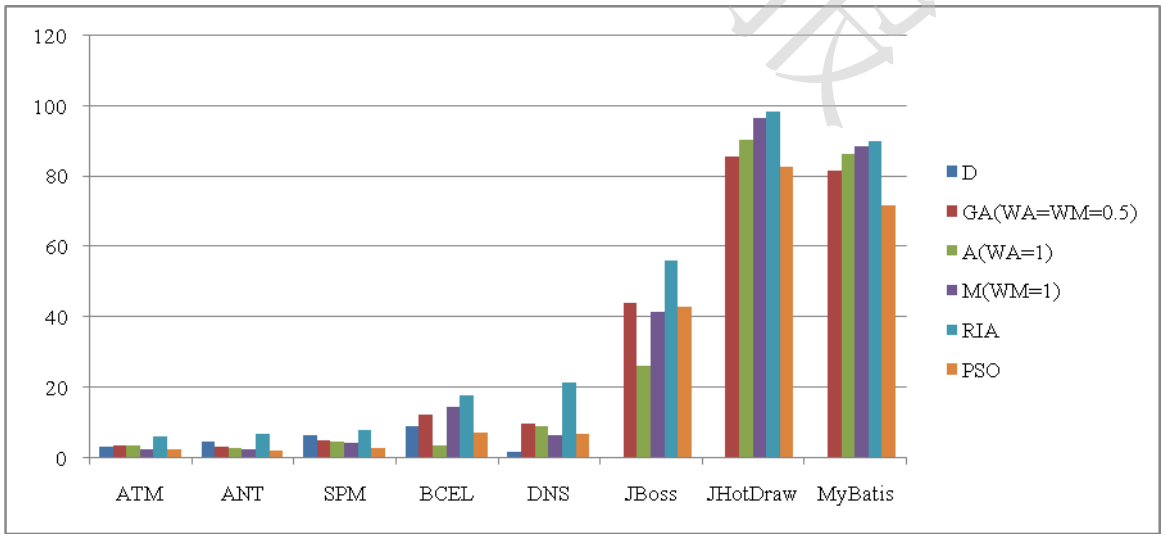

图 4 测试桩复杂度对比图

由表 6 和图 4 可以看出，对于前五个系统，只有 DNS 系统采用 D 方法生成的类集成测试序列所

花费的测试桩复杂度低于 PSO 方法所生成的类集成测试序列所花费的测试桩复杂度，这是由于 DNS 系统有 61 个类，数目较多，但是仅有 16 个环路，数目少，在这情况下，D 方法比 PSO 方法更加精准。环路少，容易进行 SCCs 划分，便于直接从环路入手，适应于 D 方法。这种情况下，基于搜索的方法如 PSO 方法具有盲目性。而对于 ATM、ANT 和 SPM 三个系统，与 D 方法和 GA 方法、A 方法、M 方法相比，PSO 方法所生成的类集成测试序列所花费的测试桩复杂度最低，RIA 方法生成的类集成测试序列所花费的测试桩复杂度最高。对于 BCEL 系统，A 方法所生成的类集成测试序列所花费的测试桩复杂度最低，PSO 方法生成的类集成测试序列所花费的测试桩复杂度次低，而 RIA 方法生成的类集成测试序列所花费的测试桩复杂度最高。

由表 6 和图 4 还可以看出，对于后三个系统中的 JBoss 系统来说，A 方法生成的类集成测试序列所花费的测试桩复杂度最低，M 方法生成的类集成测试方法所花费的测试桩复杂度次低，PSO 方法生成的类集成测试序列所花费的测试桩复杂度低于 GA 方法生成的类集成测试序列所花费的测试桩复杂度，RIA 方法生成的类集成测试方法所花费的测试桩复杂度最高。对于 JHotDraw 和 MyBatis 系统，PSO 方法生成的类集成测试序列所花费的测试桩复杂度最低，GA 方法、A 方法、M 方法生成的类集成测试序列所花费的测试桩复杂度次低，而 RIA 方法生成的类集成测试序列所花费的测试桩复杂度最高。

根据上述分析总体来看，PSO 方法生成的类集成测试序列所花费的测试桩复杂度较低，测试时花费的代价较小。

（2）属性复杂度

本部分实验比较 PSO 方法和其它方法生成类集成测试序列所花费的属性复杂度高低。实验结果如表 7 所示。其中表 7 的列 2-列 6 分别给出 D 方法、GA 方法、A 方法、M 方法、RIA 方法以及 PSO 方法在 ATM、ANT、SPM、BCEL 和 DNS、JBoss、JHotDraw 和 MyBatis 这八个系统的类集成测试序列的属性复杂度。

由表 7 可以看出，对于 ATM、ANT、SPM 和 BCEL 系统，相对于其它的类集成测试序列生成方法，PSO 方法生成的类集成测试序列的属性复杂度最低，而由于随机算法的随机性，RIA 方法生成的类集成测试序列的属性复杂度值的范围最大，GA 方法、A 方法和 M 方法生成的类集成测试序列的属性复杂度均低于 D 方法生成的类集成测试序列的属性复杂度。

但是对于 DNS 系统，D 方法生成的类集成测试序列的属性复杂度最低，这同样是由于 DNS 系统所包含的类的个数较多，而环路个数较少，适应于 D 方法。PSO 方法生成的类集成测试序列的属性复杂度次低，而 RIA 方法生成的类集成测试序列所花费的属性复杂度最高。即 PSO 方法生成的类集成测试序列的属性复杂度低于 GA 方法、A 方法、M 方法和 RIA 方法生成的类集成测试序列的属性复杂度。

由表 7 还可以发现，对于 JBoss、JHotDraw 和 MyBatis 这三个系统，与其它方法相比，PSO 方法所生成的类集成测试序列所花费的属性复杂度最低，而 GA 方法、A 方法、M 方法优于 RIA 方法。

总体来看，PSO 方法生成的类集成测试序列有较低的属性复杂度，测试时花费的代价较小。

（3）方法复杂度

本部分比较本文方法和其它方法生成类集成测试序列的方法复杂度高低。实验结果如表 8 所示。其中表 8 的列 2-列 6 分别给出 D 方法、GA 方法、A 方法、M 方法，RIA 方法以及 PSO 方法在 ATM、ANT、SPM、BCEL 和 DNS、JBoss、JHotDraw 和 MyBatis 这八个系统中的类集成测试序列的方法复杂度。

由表 8 可以看出，对于 ATM、SPM 和 BCEL 系统，与其它的方法相比，PSO 方法生成的类集成测试序列的方法复杂度最低。D 方法生成的类集成测试序列的方法复杂度次低。GA 方法、A 方法、M 方法生成的类集成测试序列的方法复杂度略低于 RIA 方法生成的类集成测试序列的方法复杂度。

由表 8 还可以看出，对于 JBoss、JHotDraw 和 MyBatis 三个系统，PSO 方法生成的类集成测试序列的方法复杂度最低，而 GA 方法、A 方法、M 方法生成的类集成测试序列的方法复杂度略低于 RIA 方法生成的类集成测试序列的方法复杂度。

因此，总体来看，PSO 方法生成的类集成测试序列有较低的方法复杂度，测试时花费的代价较小。

表 7　属性复杂度

Systems	D	GA(W_A=W_M=0.5)	A(W_A=1)	M(W_M=1)	RIA	PSO
ATM	[39-67]	[32-90]	[32-77]	[34-90]	[30-188]	[30-51]
ANT	[152-274]	[57-127]	[51-130]	[75-293]	[53-460]	[36-57]
SPM	[146-232]	[60-136]	[54-141]	[59-205]	[75-310]	[45-71]
BCEL	[101-143]	[64-168]	[40-118]	[58-231]	[54-330]	[51-119]
DNS	[19-28]	[64-169]	[49-167]	[64-304]	[62-556]	[53-103]
JBoss	-	[1140-1206]	[1130-1145]	[1140-1199]	[1130-1213]	[1130-1138]
JHotDraw	-	[7606-8142]	[7448-7825]	[7936-8217]	[5295-10260]	[6675-7238]
MyBatis	-	[4589-4713]	[4437-4690]	[4908-5102]	[3993-5922]	[4125-4199]

表 8 方法复杂度

Systems	D	GA(W_A=W_M=0.5)	A(W_A=1)	M(W_M=1)	RIA	PSO
ATM	[13-19]	[14-31]	[13-31]	[12-26]	[11-46]	[11-18]
ANT	[19-32]	[33-80]	[30-76]	[28-54]	[23-116]	[26-37]
SPM	[27-47]	[23-52]	[25-59]	[24-49]	[25-80]	[18-25]
BCEL	[70-87]	[93-130]	[91-167]	[84-115]	[96-221]	[30-83]
DNS	11	[48-110]	[59-117]	[41-91]	[35-234]	[43-71]
JBoss	-	[800-876]	[884-1117]	[810-881]	[768-1104]	[791-823]
JHotDraw	-	[7809-8476]	[8337-8976]	[7015-8373]	[4727-10682]	[6321-6782]
MyBatis	-	[2380-2876]	[2593-3176]	[1726-2476]	[1913-3087]	[1578-1830]

（4）实际运行时间

为了回答问题（2），针对不同的方法，统计它们各执行一次平均所花费的实际运行时间。为了获取较为准确的时间，在 GenCITO 系统中相应位置添加 System.currentTimeMillis()方法获取程序运行时间。生成类测试序列所需要运行的时间如表 9 和图 5 所示。其中，图 5 的横轴表示实验对象，即 ATM、ANT、SPM、BCEL、DNS、JBoss、JHotDraw 和 MyBatis 这八个系统，纵轴表示 GA 方法、A 方法、M 方法，RIA 方法以及 PSO 方法各执行一次平均所花费的实际运行时间。

实验操作系统为 Windows 8，CPU 为酷睿 i5 双核（2.6GHz），物理内存为 4G。

表 9 运行时间（ms）

Systems	Time(GA)	Time(A)	Time(M)	Time (RIA)	Time(PSO)
ATM	3003	2579	2826	6.93	5445
ANT	6968	7945	5486	8.27	7991
SPM	3182	2596	2385	2.87	6026
BCEL	82546	55318	57226	96.23	18700
DNS	118191	121923	117914	145.17	10642
JBoss	4749491	4968435	4857692	4580.9	61457
JHotDraw	714134094	794767982	684568791	252143	561936
MyBatis	510072671	485789893	469898214	317725.33	359067

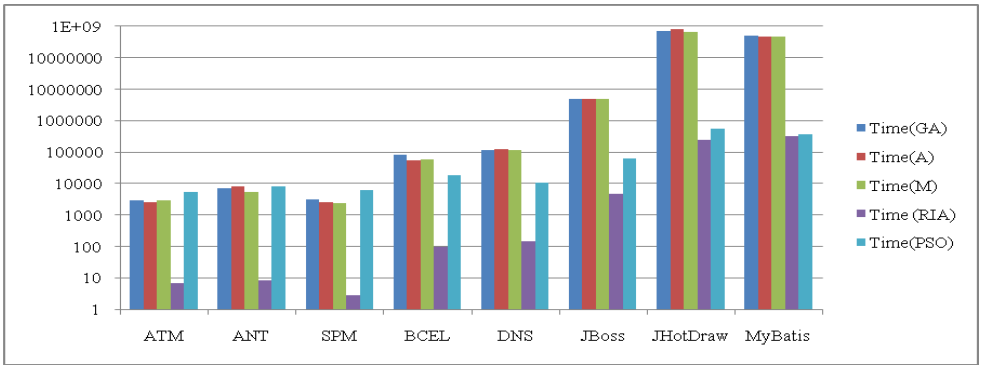

图 5 运行时间对比图

由统计数据可以看出，生成最优类测试序列的时间随着类规模的变化而变化。随着待测系统规模的不断增大，时间逐渐上升。这是由于类是生成最优类集成测试序列的对象。即，运行时间与类的数量有较大关系。总体可以看出，对于所有的待测系统，RIA 方法运行时间最短，表现出明显的时间优势。对于较小规模的系统，如 ATM、ANT 和 SPM 系统，PSO 方法的运行时间比 GA 方法、A 方法、M 方法的运行时间长。而对于较大规模的系统，如 BCEL、DNS、JBoss、JHotDraw 和 MyBatis 系统，PSO 方法的运行时间比 GA 方法、A 方法、M 方法的运行时间短。因此，PSO 方法对规模较大的软件系统能够显示运行时间方面的优势。

基于上面的实验数据及分析，针对 4.2 节提出的三个研究问题做出回答，具体如下：

回答问题（1）：与 D 方法、GA 方法，A 方法，M 方法和 RIA 方法相比，本文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。

回答问题（2）：在 D 方法、GA 方法，A 方法，M 方法和 RIA 方法这些所有方法中，RIA 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 GA 方法，A 方法和 M 方法相比，在相同条件下（如种群数量，迭代次数），本文方法可以更快找到更优的类集成测试序列。

回答问题（3）：与 D 方法、GA 方法，A 方法，M 方法和 RIA 方法相比，本文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序列。

5 相关工作

对于类集成测试序列的确定问题，其主要解决方法分为基于图论的方法和基于搜索的方法这两大类。

5.1 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用对象关系图（ORD）或测试依赖图（TDG）来表示。类集成测试序列确定问题就变成了对有向图中节点的排序问题。

5.1.1 最小化测试桩的数量

Kung 等人[8]最早提出类测试序列的确定方法。他们指出如果类图中不存在依赖环路，可以通过逆向拓扑排序得到类测试序列。他们认为类图中的每个环路至少包含一个关联关系。因此，为了消除环路并且使得构造尽可能简单的测试桩，应打破关联关系。

Tai 和 Daniels[9]假设继承关系和聚合关系不形成环，只有关联关系会形成环。该方法将类划分为高低两个级别：如果一个类存在继承关系或聚合关系，则将该类划分到高级别的类中；如果一个类只存在关联关系，则将该类划分到低级别的类中。强依赖关系只在高级别的类中进行识别，强依赖关系中的每一条边被赋予一个权值（有向边源节点的入度与目标节点出度的和），权值最大的边将首先被消除，因为权值大的边可能与多个环有关。他们的测试序列分配策略将对象关系图中的三种关系分为两个层次，继承关系和聚集关系位于一个层次，关联关系处于一个层次，当穿越主层的关联关系没有形成环路时，该方法导致构造不必要的测试桩。

Le Traon 等人[10]采用了一种基于 TDG 模型的方法进行集成测试，该测试依赖图由类和方法之间的测试依赖关系构成。该方法中，每个节点被赋予一个权值，然后删除具有最大权重的节点的入边，直至图中不存在环路为止。该方法首先识别强连通图，然后将强连通图中的边分为树边、前向边、叶边和交叉边，节点的权值是所有流入和流出该节点的叶边的数量和。TDG 中没有提供任何有关依赖关

系（继承、聚合和关联）的信息，他们的方法减少了测试桩的数目，但可能断开继承和聚集等强依赖关系，导致构造复杂的测试桩。

Briand 等人[5]将 Tai 和 Daniels[9]和 Le Traon 等人[10]的方法相结合，提出了一种更为精确的权值计算方法。该方法首先识别强连通组件，然后为强连通组件中的每条关联边赋予一个权值。有向边 v1→v2 的权值为节点 v1 的入度与节点 v2 的出度乘积。然后消除权值最大的关联边，直至有向图中不存在环为止。Briand 等人[5]不打破继承、聚集等强依赖关系，此外，与 Tai 和 Daniels[9]的方法相比，该策略对边权值的计算更加精确。

Tai 和 Daniels[9]的方法和 Le Traon 等人[10]的方法都是将所需构造的类桩数目作为打破环路的优化目标，而 Briand 等人[5]是以构造的特效桩数目作为优化目标。

Hewett 等人[7]将 UML 类图创建的 TDG 作为依赖模型，采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[11]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列，通过最小化测试桩的数目找到一个最佳测试序列，实验表明可以通过添加额外的启发式信息的方式改进他们已有的算法，改进后的方法不使用依赖类型信息，与 Le Traon 等人[10]的方法相比，所需测试桩数目均最少，而 Hewett[11]方法花费更少的时间，显示了时间性能方面的优势。

国内，江西财经大学毛澄映老师[12]对 Tai 和 Daniels [9]算法以及 Briand 等人[5]算法进行了改进。他们使用由 UML 类图创建的加权对象关系图（WORD）作为依赖模型，包括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、边的方向、关联强度）表示每个关联关系的权重。首先选择和删除关联边打破所有环偶对（组成环路的边个数为 2 的环路为一个环偶对），然后使用基于图论的方法打破其它类型的环路。该算法选择和删除权重最高的关联边来打破环路。

据我们所知，对于 ORD 中存在环路的类集成测试序列问题，目前只有我们的工作[13, 14]较全面地考虑了面向对象程序的特性，如封装、继承和多态、抽象类不可实例化等，考察了这些特性对打破环路以及测试序列的影响，提出了静态依赖关系和动态依赖关系构成的环路中边的删除规则，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原则的类集成测试序列问题。

本文与以上大多数基于图论的解决方法（Le Traon 等人的方法[10]除外）的相同点是在打破环路时，不允许删除继承和聚集等强依赖关系，而只允许删除关联等弱依赖关系。不同点是我们采用的不是基于图论的方法。本文与我们以前工作[13, 14]的相同点是在打破环路时，不允许删除继承和聚集等强依赖关系，而是只打破关联等弱依赖关系。不同点是，本文采用的不是基于图论的方法，同时没有考虑面向对象程序的多态、抽象类不可实例化等特性。

然而，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序列所花费的总体测试桩复杂度越低。相比于测试桩的数量，测试桩复杂度更能够准确地衡量创建一个测试桩难易程度。因此，本文并没有采用测试桩的数目作为衡量测试代价的标准，而是将最小化测试桩的总体复杂度作为衡量测试代价的标准。

5.1.2 最小化测试桩的总体复杂度

Abdurazik 等人[15]提出了一种基于启发式的全局优化技术。首先，根据定量的耦合度量来计算节点和边的复杂度，其中，耦合度量时使用的属性复杂度和方法复杂度分配相同的权重。其次，利用结点和边的复杂度模拟类以及类之间的关系。第三，结合边复杂度、节点复杂度以及环路的个数三个因素来打破环路。该方法需要构造更复杂的测试桩，但得到类测试序列所花费的总体测试桩的复杂度最低。

我们在以前工作中[16]提出了一种基于耦合度量的类间集成测试序列的确定方法。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。与 Abdurazik 等人[15]的方法不同，我们提出了度量边复杂度的新方法，对度量时使用的属性复杂度和方法复杂度重新分配了权重值。实验结果表明，采用我们的方法[16]生成一个类集成测试序列所花费的总体复杂度有所降低，节约了测试成本。

总体来看，目前基于图论的研究方法以构造的测试桩的个数为评价指标为主，以构造的测试桩总体复杂度为评价指标的方法相对较少。

5.2 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[4,17]指出基于图论的算法的缺点：基于图论的解决方案大多假设每个测试桩的开发成本相同。但是 Briand 等人[4,17]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，并且没有充分考虑并估计需要构建的测试桩的具体成本，如类中属性的数量、调用的数量以及各种约束等。事实上，在基于图论的解决方案中要想考虑到所有这些因素是比较困难的[7]。

为了克服这个限制，Briand 等人[4,17]将类集成测试序列的确定问题作为一个多目标优化问题[18]。

5.2.1 最小化测试桩的数目

Hanh 等人[19]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。即首先将一个类集成测试序列表示成一个个体，一组类集成测试序列作为初始种群，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，进而打破环路，产生满足条件的类测试序列。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。

Borner 等人[20]将 ORD 作为依赖模型。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，并使用模拟退火算法和遗传算法找到一个最优的类集成测试序列。

国内王正山老师等[21]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。他们提出了一种耦合度量技术，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（RIA）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，提高了有效性。

由以上分析可以发现，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，来评价生成一个类集成测试序列需要花费的总体代价，进而评价类集成测试序列生成方法的优劣。

5.2.2 最小化测试桩的总体复杂度

Briand 等人[4,17]提出了基于遗传算法的解决方案。他们的解决方案中，使用类之间的耦合度和遗传算法来计算复杂的测试桩模块开发成本。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[4,17]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，导致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。因此不适合复杂的软件系统。为了更好地克服这一问题，Cabral 等人[22]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。他们使用基于蚁群优化算法的多目标优化算法[23,24]来生成 Pareto 最优解集合，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。

之后，Vergilio 等人[18]又提出了两种算法：非占优排序多目标遗传算法（Non-dominated Sorting GA[24]，NSGA-II[25,26]）和禁忌搜索多目标优化（MTabu [27,28]）。其中，NSGA-II 是最著名的算法之一，使用的是多目标优化算法。另一方面，MTabu 不是进化算法，而是基于禁忌搜索的算法，是运筹学领域最适用的算法[27]。因此，Vergilio 等人[18]选择了 NSGA-II 和 MTabu 算法。使用与 Briand 等人[17]相同的基准程序执行这些算法，根据帕累托优势概念[29]评估多目标优化算法返回的解，耦合度量指标：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。然而，只有两个耦合度量指标，即基于创建的测试桩的属性和方法的数量。

Assunção 等人[30]又对 NSGA-II 和改进的强度帕累托进化算法（SPEA2[31]）这两种多目标优化算法进行了性能比较：以被访问的属性的个数，被调用的不同方法的个数（包括构造函数），不同的返回类型的个数，传递的不同的参数类型的个数作为四个适应度函数时，SPEA2 比 NSGA-II 略有更好的收敛性。后来，Assunção 等人[32]又进一步对 NSGA-II、SPEA2 和 PAES 这三种算法进行了比较，

结果证明对于比较复杂的系统，PAES 算法效果更优，但是对于大多数实验对象来说，NSGA-II 效果最好。

近年来随着智能计算领域的发展，出现了新型智能算法——超启发式算法[33]。超启发式算法提供了某种高层策略，通过操纵或管理一组低层启发式算法，以获得新启发式算法。Vergilio 等人[34]提出用超启发式算法来解决类测试序列问题。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。

国内，王正山老师[35]提出了一种针对面向对象软件集成测试中确定类测试序列的混合遗传算法。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[4,17]的基于遗传算法的方法。之后，东南大学李必信教授和王正山老师[21]又使用耦合度量的方法计算测试桩的复杂度。他们将耦合度量和 RIA 相结合解决类测试序列确定问题，允许断开继承和聚集等强依赖关系。

基于搜索的方法中，解决类集成测试序列问题的主要途径之一是基于遗传算法的方法（包括改进的遗传算法，如 NSGA-II）。对于遗传算法，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。

本文属于第二类方法，即基于搜索的方法。本文是将测试桩的总体复杂度作为衡量测试代价高低的指标。

相比于基于遗传算法的方法，本文方法具有以下优势：（1）PSO 算法的粒子具有记忆性，能够保存所有粒子最优解；（2）PSO 算法不需要编码，没有交叉和变异操作，粒子通过内部更新，原理更简单、设置参数少更易实现；（3）PSO 算法随最优解不断运动，收敛速度快。

6 结束语

本文提出一种基于粒子群优化算法的类集成测试序列确定方法。实验结果表明，该方法生成的类测试序列花费更小的测试桩代价。大型系统通常包含多个类，难以直接应用本文方法确定类集成测试序列。但是我们可以将系统按照系统结构或功能模块划分为多个类数量较少的子系统，利用本文方法确定子系统的类集成测试序列之后，逐层向上对较高层次进行分析，从而求解整个系统的类集成测试序列。

通过研究发现，粒子群优化算法对于类集成测试序列有不错的效果，但仍存在一些问题。首先，如类的数量过大时，形成的类排序序列成指数关系增长，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，需要采取一定的策略避免陷入局部最优解。其次，本文是在静态依赖关系的基础上进行的类集成测试序列确定问题的研究，而没有处理反射机制的类依赖关系，即没有考虑类间的动态依赖关系，导致结果不够精确，需要在以后的工作中进行研究。

致谢 感谢各位审稿专家提出的宝贵意见。

参 考 文 献

[1] Briand L C, Feng J, Labiche Y. Software Engineering with Computational Intelligence: Experimenting with genetic algorithms to devise optimal integration test order. New York, USA: Springer, 2003.

[2] Hanh V L, Akif K, Le Traon Y, and 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

[3] Windisch A, Wappler S, Wegener J. Applying particle swarm optimization to software testing// Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation. London, England, 2007: 1121-1128

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

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

[6] 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

[7] 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 Bay, USA, 2008: 703-708

[8] Kung D, Gao J, Hsia P, Toyoshima Y, Chen C. A test strategy for object-oriented programs//Proceedings of the 9th Annual International Computer Software and Applications Conference. Dallas, Texas, USA, 1995: 239-244

[9] 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

[10] 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

[11] 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

[12] 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

[13]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)

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

[15] 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

[16] 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)

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

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

[19] Hanh V L, Akif K, Le Traon Y, and 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

[20] 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

[21] 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

[22] 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

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

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

[25] 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

[26] 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

[27] Gendreau M, Potvin J Y. Tabu search. In: Gendreau, M., Potvin J.Y., Hillier F.S. (eds.) Handbook of metaheuristics, international series in operations research and management science. 2nd ed. New York,USA: Springer US, 2010

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

[29] Pareto V. Manuel d'economie politique. 2nd ed. France: AMS Press, 1927

[30] Assunção W K G, Colanzi T E, Pozo A T R, Vergilio S R. 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

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

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

[33] Burke E K, Hyde M, Kendall G, Ochoa G, Ozcan E, Qu R. A survey of hyper-heuristics. Nottingham: School of Computer Science and Information Technology, University of Nottingham, Technical report: NOTTCS-TR-SUB-0906241418-2747, 2009

[34] Guizzo G, Fritsche G M, Vergilio S R, Aurora T R, Poza A. 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

[35] 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)

Author1, ZHANG Yan-Mei, born in 1982, Ph. D, lecturer. Her research interests include software analysis and testing.

Author2, JIANG Shu-Juan, born in 1966, Ph. D, professor, Ph. D. supervisor. Her research interests include compilation techniques, software engineering.

Author3, CHEN Ruo-Yu, born in 1990, Master. Her research interests include software analysis and testing.

Author4, WANG Xing-Ya, born in 1990, Ph. D candidate. His research interests include software analysis and testing, fault localization.

Author5, ZHANG Miao, born in 1992, Master. Her research interests include software analysis and testing.

Background

Class integration testing is an important part in object-oriented software testing. For integration testing problem, testers need to find test sequences of classes in order to execute interactions, and it is a key and difficult problem to determine the class integration test order of class cluster in integration testing. Reasonable class integration test order can reduce the overall complexity of test stub, and reduce test cost.

One major problem of class integration test order is the presence of cyclic dependency calls. Many researchers have proposed techniques to solve this problem by removing relationships to break cycle s 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 algorithm to break cycles. Overall, in the existing method, some allow removing inheritance and aggregation to break cycles, which lead to the increase of stubbing complexity. For cycles breaking problem, there are mainly two solutions. One is graph- based method, which is difficult to achieve; the other is search-based algorithm, which is not very high for efficiency. 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 particle swarm optimization algorithm. The technique belongs to the search-based approach. It combines inter- class coupling measurement and particle swarm optimization algorithm. The experimental results show that the particle swarm optimization algorithm takes a lower test stub cost for solving the class test order problem.

The main contribution of this work is the followings: (1) a class integration test order determination method based on particle swarm optimization algorithm is proposed. (2) The approaches that each test order is mapped to one dimensional space, and the optimal particle is mapped to a class test order are proposed.

This work was supported in part by awards from the National Nature Science Foundation of China under grant No. 61502497; State Key Laboratory for Novel Software Technology at Nanjing University under grant No. KFKT2014B19; Guangxi Key Laboratory of Trusted Software No. kx201530; China Postdoctoral Science Foundation funded project under grant No. 2015M581887; Science and Technology Program of Xuzhou under grant No. KC15SM051.