计算机科学
COMPUTER SCIENCE

面向 SOA 的集成测试序列生成算法研究

张冰清，费琪，王轶辰，杨召

引用本文

张冰清，费琪，王轶辰，杨召. 面向 SOA 的集成测试序列生成算法研究[J]. 计算机科学, 2022, 49(11): 24-29.

ZHANG Bing-qing, FEI Qi, WANG Yi-chen, Yang Zhao. Study on Integration Test Order Generation Algorithm for SOA[J]. Computer Science, 2022, 49(11): 24-29.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.210400210

面向 SOA 的集成测试序列生成算法研究

张冰清[1]　费　琪[2]　王轶辰[1]　杨　召[2]
1 北京航空航天大学可靠性与系统工程学院　北京 100083
2 江苏自动化研究所　江苏 连云港　222006
(zhangbingqing@buaa.edu.cn)

摘　要　集成测试序列生成是软件集成测试研究中的一个重要课题,合理的测试序列可以在提高集成测试效率的同时有效降低测试代价。面向服务的架构 SOA(Service-Oriented Architecture)是近年来在企业中被广泛应用的一类分布式架构,目前针对 SOA 架构中集成测试序列生成的相关研究较少。由于 SOA 架构中服务间组合具有多态性,单纯使用传统的自顶向下和自底向上等集成测试策略,无法得到 SOA 架构中服务软件之间的集成测试序列,而目前以面向对象系统中类簇为对象的集成测试序列生成研究又很难适应 SOA 架构中服务之间复杂的耦合关系。基于此,提出了一种基于遗传算法的集成测试序列生成方法,用于解决 SOA 架构中服务软件之间的集成测试问题。该方法提出了利用服务特征组的概念表征集成测试影响因素和利用集成测试优先度的概念来表征服务软件的集成测试重要度的基本思想,同时构建了测试依赖图,用于描述 SOA 架构中服务软件之间的复杂耦合关系,在此基础上提出了面向测试依赖图的测试优先度算法,并以降低测试代价为最优化目标设计了遗传算法,用于生成集成测试序列。最后通过实例验证了所提方法的可行性和正确性,结果表明,所提方法能够生成测试优先度相对较高的节点优先集成和测试代价较低的服务软件集成测试序列。

关键词:SOA;Web 服务;集成测试序列;测试依赖图;遗传算法
中图法分类号　TP311.5

Study on Integration Test Order Generation Algorithm for SOA

ZHANG Bing-qing[1],FEI Qi[2],WANG Yi-chen[1] and Yang Zhao[2]
1 School of Reliability and System Engineering,Beihang University,Beijing 100083,China
2 Jiangsu Automation Research Institute,Lianyungang,Jiangsu 222006,China

Abstract　Integration test order generation is an important problem in software integration testing research. Reasonable test order can improve the efficiency of integration test and reduce the cost of test. Service oriented architecture(SOA) is a kind of distributed architecture widely used in enterprises in recent years. At present,there are few researches on integration test order generation in SOA architecture. Due to the polymorphism of service composition in SOA architecture,it is impossible to get the integration test sequence between service software in SOA architecture by using the traditional top-down and bottom-up integration test strategies. However,the current research on the generation of integration test sequence based on class cluster in object-oriented system is difficult to apply to the complex coupling relationship between services in SOA architecture. Therefore,an integration test order generation method based on genetic algorithm is proposed to solve the integration test problem between service software in SOA architecture. In this method,the concept of service feature group is used to represent the influence factors of integration testing,and the concept of integration testing priority is used to represent the importance of integration testing of service software. At the same time,the test dependency graph is constructed to describe the complex coupling relationship between service software in SOA Architecture. In order to reduce the cost of test,a genetic algorithm is designed to generate integrated test sequence. Finally,an example is given to verify the feasibility and correctness of the method. The example results show that the proposed method can generate service software integration test orders with relatively high test priority and low test cost.

Keywords　SOA,Web service,Integration test order,Test dependency graph,Genetic algorithm

1　引言

随着用于开发分布式的交互操作的应用程序的普及,SOA 被认为是下一代 Web 服务的基础框架,成为了计算机信息领域的一个新的发展方向,越来越多的企业开始使用 SOA 架构[1]。面向服务的所有设计都有一个共同目标,即让服务支持增加潜在的可组合性[2]。因此,SOA 中各服务间存在着大量的调用关系,在抽象化后会得到存在大量环路的网络图。传统的自顶向下集成测试策略和自底向上集成测试策略等面向的都是层次结构明显的软件体系架构[3],因此传统

到稿日期:2021-04-20　返修日期:2021-06-29
通信作者:王轶辰(wangyichen@buaa.edu.cn)

策略在面对层次化结构不明显且具有较多环路的 SOA 体系架构时，无法较好地解决服务软件集成测试序列生成问题，可能会增加额外的测试代价，测试效率大幅降低，因此需要有针对性地提出一种新的测试策略。

目前集成测试序列的研究对象主要集中在面向对象编程的类簇中，而针对 SOA 架构中服务软件的集成测试研究相对较少。类间测试序列的方法主要是通过删除类图中的某些依赖关系，破除环路，进行拓扑排序来得到最终序列[4]。文献[4-12]通过研究发现，目前类集成测试序列的确定主要依据两个目标，即减少集成测试的成本和提高集成测试的揭错效率。

SOA 架构中的服务软件与面向对象中的类簇均存在大量的调用关系，但是服务软件间的调用关系是动态的且具有多态性[13]。相比类，服务软件层次级别更高，软件自身更加复杂，需要进一步对服务软件进行分析。

针对类簇与服务软件之间的区别，本文提出了一种面向 SOA 的集成测试序列生成方法，用于确定服务软件的集成测试序列，将服务之间的动态调用关系统一抽象为依赖关系，以构建测试依赖图，利用服务软件自身复杂性来构建特征组，使用遗传算法获得最优测试序列。

2 相关研究

2.1 测试序列生成相关方法

目前关于集成测试序列的研究对象主要集中在面向对象的类簇中，已有的关于类集成测试序列生成的方法主要有 3 种：基于图论的方法、基于搜索的方法和基于切片的方法[4]。

(1)基于图论的方法

最早研究类集成测试序列的是国外学者 Kung 等[5]，他们提出了基于图论的方法。该类方法的主要思想是[4-6]：将类与类之间的关系抽象化，通过识别图中的强连通件，或者是对各节点边进行赋值操作，在基于测试代价最小的限制条件下进行删边操作，破除环路，根据得到的无环图进行拓扑排序生成测试序列。国内学者 Yu 等[7]结合了复杂网络的思想，将程序抽象为网络，考虑了网络中节点自身的重要性以及网络中各节点的错误传播率，在删边操作时保证了重要节点被优先测试。

(2)基于搜索的方法

由于基于图论的方法在针对大型的面向对象的程序时效率并不令人满意，因此 Briand 等[8]提出了利用智能优化算法来解决测试序列生成问题。基于搜索的类集成测试序列确定方法的总体思想是[4,8-11]：首先设计初始种群，然后在一定条件下，通过进化操作种群进行处理，进而打破环路，最后得到最优的类集成测试顺序。

(3)基于切片的方法

国外学者 Jaroenpiboonki 等将面向对象的切片技术用于类集成测试序列。该方法在方法层面按照依赖关系对类进行切分，由于不在类级别，因此不需要构建测试桩就可以进行删边操作，从而得到无环图进行拓扑排序，最终得到类测试序列[12]。

2.2 SOA 架构中服务软件测试相关研究

服务分析公司的资深分析家布鲁伯格将服务的测试技术的研究分为如下 3 个阶段[13-15]。

(1)关注服务内部的测试。主要测试单个服务的基本功能，类似于传统软件中的单元测试。

(2)测试面向服务的体系结构。主要测试服务的发布、绑定和查找这 3 个重要角色的功能，以及服务间的异步通信能力和服务质量问题。

(3)关注服务的动态特性和服务集成的总体测试。

目前，对 SOA 架构的服务软件测试的主要关注点在前两个阶段，即对单个服务的测试以及服务间的通信能力的测试[14]，较少涉及针对服务软件的集成测试研究。但实际上服务之间的互相调用可能引发消息和通信等错误，因此对服务软件进行集成测试十分必要。

在对 SOA 中的服务进行集成测试时，SOA 的独特特性会引发严重的问题。SOA 架构中的动态绑定允许我们将以服务为中心的系统定义为一个调用抽象接口的工作流，这些抽象接口在工作流执行之前甚至执行期间会绑定到具体的服务。因此，经典的集成测试方法在需要精确识别系统组件及其相互关系时很可能会失败。由于动态绑定，我们必须测试一个组合所有可能端点的服务伙伴链接，其中服务之间端点组合的多态性类似于测试面向对象系统[15]。但在针对 SOA 中的服务进行测试时，问题变得更加复杂，其原因在于服务之间的调用组合更加复杂后，测试所有可能的调用关系的测试成本更加高昂，因此确定服务软件之间的集成测试序列变得十分重要。

3 SOA 软件测试序列生成算法

本文将服务软件集成测试序列的研究问题定位在服务测试技术研究的第三阶段，根据服务软件自身的特殊性以及同类簇的相同点，对类集成测试方法中的基于图论和基于搜索的方法进行改进。

3.1 服务集成测试序列生成方法原理

服务软件集成测试序列的确定同样依据两个目标：降低测试成本和提高测试效率。本文方法的原理如图 1 所示，主要有 3 个部分，首先基于系统各层次的需求规范，构建服务的特征组，计算服务自身的初始重要度；然后定义、分析抽象系统各服务间的调用关系，通过构建测试依赖图来确定每个服务的集成测试优先度；最后利用遗传算法生成最终的集成测试序列。

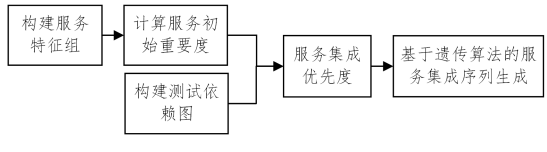

图 1 服务集成测试序列生成方法原理图

Fig. 1 Method schematic diagram of service integration test sequence generation

3.2 服务初始重要度计算

服务的重要度由多种因素决定。研究表明，软件的规模、结构复杂度和耦合性是影响软件自身复杂度的最重要的 3 个

方面[7]。本文通过构建服务特征组来计算服务的重要度度量 C。

定义 1 服务特征组是影响服务重要度的因素集合 F_i。结合服务自身复杂性以及对集成测试的影响，选取五元因素集来构建服务特征组，具体表示为：

$$F_i=\langle LOC_i, CBO_i, SMC_i, SAT_i, HER_i\rangle \tag{1}$$

其中，LOC 表示代码行数，依据服务的规模选取服务的代码行数，服务代码行数越多，则认为服务规模越大；CBO 表示耦合性，服务间的耦合性仅考虑与该服务之间存在调用关系的服务的数量，服务间调用数越多，服务间的耦合性越强，服务就越复杂；SMC 表示结构复杂度，可利用服务软件的代码圈复杂度计算表示，圈复杂度越高代表该服务软件的结构复杂度越高；SAT 表示服务所处层次，在 SOA 中，服务间存在着明显的层次结构特点，高层次的服务会调用低层次服务的各项服务，从减少集成测试中测试桩数量的角度考虑，需要优先集成低层次的服务；HER 表示历史出错率，在软件测试中，通常认为历史出错率高的服务在再次进行测试时出现错误的概率更大。服务的历史出错率选择在集成测试之前进行其他测试，如单元测试时的错误率。

定义 2 服务初始重要度 C 指服务受自身因素影响在集成测试中重要度的度量，由服务特征组中的因素归一化之后的加权之和计算得到。其计算式如下：

$$C_i=\alpha\frac{LOC_i}{\sum_{i=1}^{k}LOC}+\beta\frac{CBO_i}{\sum_{i=1}^{k}CBO}+\gamma\frac{SMC_i}{\sum_{i=1}^{k}SMC}+\theta\frac{SAT_i}{\sum_{i=1}^{k}SAT}+\lambda\frac{HER_i}{\sum_{i=1}^{k}HER} \tag{2}$$

其中，$\alpha,\beta,\gamma,\theta,\lambda$ 加权值取平均值 0.2。

3.3 服务测试优先度计算

本文把服务间的动态服务调用关系统一定义为依赖关系，以构建有向图——测试依赖图。网络图中的重要测试节点有两个特性，即节点的复杂性和节点的错误传播影响性，两者相互影响、相互作用。因此，本文提出集成测试优先度度量 P_i，将重要测试节点的两个特性与节点的初始重要度相结合，优先度高的服务需要被提前集成测试。

定义 3 集成测试优先度 P_i 指在集成测试中，各服务在集成测试中优先顺序的相对度量，即面向测试依赖图将节点自身的初始重要度与其邻接节点的初始重要度结合起来迭代得到的度量。

3.3.1 测试依赖图模型

本文将所有的服务抽象为测试依赖图中的节点，将两个服务之间所有的动态调用关系抽象为节点之间的一条边。其形式化定义如下。

定义 4 测试依赖图表示为一个二元组 $G=(V,E)$，其中，$V=\{v_1,v_2,\cdots,v_n\}$ 表示非空的有限服务节点集；$E=\{V_{ij}\mid v_i,v_j\subset V\}$ 表示各服务之间的动态调用关系，即当 v_i 调用 v_j 时，节点 v_i 指向 v_j。节点 v_i 和 v_j 的测试依赖图如图 2 所示。

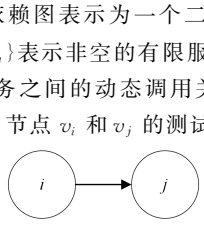

图 2 节点 v_i，v_j 的测试依赖图

Fig. 2 Test dependency graph of nodes v_i and v_j

3.3.2 基于测试依赖图的节点优先度计算

软件系统具有复杂网络的特性，在测试依赖图中，每个节点对整个软件系统动态服务调用的影响并不仅仅局限于其邻居节点，而是一种连锁式的影响[7]。若一个节点被很多节点指向，即多个节点都需要调用该节点的服务时，则该节点的错误传播影响力非常大，该节点发生错误会对系统造成更大的破坏。如果该节点被许多复杂的节点依赖，则认为该节点的影响力更大，因为当该节点出错时，有可能会造成“涟漪”效应[16]。节点的复杂性在考虑自身复杂度的同时，在测试依赖图中还需要考虑该节点所依赖的其他节点的错误传播影响力。当一个节点依赖于复杂且错误传播影响力大的节点时，该节点由于其他节点的错误传递而自身出错的可能性也更大。

因此，在测试依赖图中，节点自身的复杂性和影响力与其他节点的影响力和复杂性密切相关。本文提出了面向测试依赖图的软件测试优先度算法，通过“相互增强”的想法不断进行迭代计算，直到趋于稳定，得到最终的节点测试优先度 W。算法的具体流程如算法 1 所示。

算法 1 面向测试依赖图的测试优先度算法

输入：服务间的有向测试依赖矩阵 $\mathbf{A}=(a_{ij})$（当节点 i 有一条边指向节点 j 时，$a_{ij}=1$，否则为 0）。节点影响力和复杂度初始值 x 和 y（设定节点的影响力和复杂度的初始值均为节点自身初始重要度 C_i）。

输出：节点集成测试优先度向量 $\mathbf{W}_i$

1. 输入测试依赖矩阵 $\mathbf{A}$，以及节点影响力和复杂度初始值 x(0) 和 y(0)。设定节点的影响力和复杂度的初始值均为节点自身的初始重要度 C_i。

2. 校正节点影响力和复杂性。节点的影响力和复杂性均为服务自身重要度初始值 x，y。重复步骤 2.1－步骤 2.2，进行 k 次迭代，直到两组向量趋于稳定，迭代停止的条件为：

$$\|x_i(k)-x_i(k-1)\|+\|y_i(k)-y_i(k-1)\|<\varepsilon$$

2.1. 各节点的影响力校正为指向该节点的节点的复杂度之和，即测试依赖矩阵的转置乘以节点的复杂性向量：

$$x_i'(k)=\mathbf{A}^{\mathrm{T}}\times y(k-1)$$

2.2. 各节点复杂性校正为该节点指向的各节点的影响力之和，即测试依赖矩阵乘以节点的影响力向量：

$$y'(k)=\mathbf{A}\times x'(k)$$

2.3. 归一化：

$$x_i(k)=\frac{x_i'(k)}{\|x_i'(k)\|}\qquad y_i(k)=\frac{y_i'(k)}{\|y_i'(k)\|}$$

3. 输出各节点集成测试优先度：

$$P=0.5x(k)+0.5y(k)$$

3.4 面向 SOA 软件的测试序列生成算法

根据调研[4,7,17-19]，本文通过降低测试桩复杂度以及优先集成重要节点来实现测试序列关注的两个因素，利用遗传算法来生成 SOA 中服务的测试序列。针对遗传算法中随机初始化种群具有一定的盲目性这一问题，本文利用基于优先级的方法初始化种群。

3.4.1 基于优先级的方法初始化种群

在基于遗传算法生成类簇测试序列的方法中，大部分都是随机生成初始种群，具有一定的盲目性和随机性。因此本文提出利用基于优先级的方法来初始化种群，以期初始测试

序列根据节点的邻接关系生成。具体步骤如下：

步骤 1　根据测试依赖图生成图中节点的邻接节点集合 S；

步骤 2　随机生成初始种群，生成的染色体编号表示各节点的优先级；

步骤 3　选取优先级最高的第一个节点进行集成，然后选取该节点的邻接节点中优先级最高的节点进行集成，接着从刚集成的节点中选取优先级最高且还未被集成的节点进行集成；

步骤 4　若该节点的邻接节点已经全部被集成，则选择染色体中优先级最高且还未被集成的节点；

步骤 5　循环步骤 3 和步骤 4，直到该染色体中的所有编号全部测试完成，输出集成测试序列 L。

3.4.2　构建适应度函数

在利用遗传算法寻找最优集成测试序列时，确定适应度函数是非常重要的环节，恰当的适应度函数能够客观准确地评价基因的优劣，指导种群朝正确的方向进化[20]。本文在构建遗传算法的适应度函数时考虑了提出的集成测试策略中的两个因素，即测试桩复杂度最小化和测试优先度高的节点优先集成。

测试桩是服务的一种概括或是某种特殊服务的实现。假设组件 1 依赖组件 2，当组件 1 集成但组件 2 尚未集成时，需要构建测试桩来模拟服务 2 中的某些服务[4]。模拟一个对象构建测试桩的过程十分复杂，成本也较高。因此，在集成测试时需要通过减少构建测试桩来降低测试成本。对于基于搜索的算法，适应度函数的选取更偏向于选取测试桩的总体复杂度，这是因为选取测试桩数目为问题的优化目标会导致精确度较低。

本文测试桩的复杂度计算式如下：

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

其中，$\overline{S(i,j)}$表示服务 i 调用服务 j 时，模拟服务 j 需要构架的测试桩的复杂度，$S(i,j)$表示服务 i 调用服务 j 的接口的数量，$S_{\max}$表示服务 j 被其他服务调用的接口最多的数量，$S_{\min}$表示服务 j 被其他服务调用的接口最少的数量。

利用算法 1 得到的各节点的测试优先度为测试依赖图中的各依赖边赋值 $C(i,j)$，其表达式如下：

$$C(i,j)=\frac{P_i+P_j}{2} \tag{4}$$

其中，$C(i,j)$表示边(i,j)的测试优先值，即两节点测试优先度的平均值。

适应度函数的计算式如下：

$$Fitness=\frac{\sum\limits_{i,j\in l}C(i,j)}{\sum\limits_{i,j\in T}S(i,j)} \tag{5}$$

其中，T 表示需要构建的测试桩集合，L 表示被保留下来的测试依赖边集合。

4　实例研究分析

实验选取基于 SOA 构建的某银行管理系统，应用本文提出的集成测试序列生成方法，通过分析得到的序列的合理性来验证本文算法的可行性。该银行管理系统一共有 11 个服务。分析该企业的服务流程，根据服务层次划分标准[2]来分析系统服务的特点和功能并对其进行层次划分。该系统的服务层次主要分为企业服务层、业务功能层以及数据层，如表 1 所列。

表 1　系统服务层次

Table 1　System service hierarchy

企业服务层	1. 综合业务服务　2. 客户管理服务　3. 信用贷款服务
业务功能层	4. 账户管理　5. 账户转账　6. 费用支付　7. 贷款业务配置 8. 贷款业务查询　9. 信贷审批流程　10. 系统管理
数据层	11. 数据信息库

4.1　实例服务初始重要度计算

结合服务特征组的定义，通过对该银行系统的需求规格说明书进行分析，以及审查历史测试文档，获得计算该软件服务的服务初始重要度所需的相应数值，如表 2 所列。

表 2　实例服务特征组

Table 2　Feature group of example service

Software	*LOC*	*CBO*	*SMC*	*SAT*	*HER*
1	40	1	2	0.2	1
2	30	1	2	0.2	2
3	30	1	2	0.2	1
4	70	4	5	0.3	3
5	60	3	5	0.3	2
6	70	5	4	0.3	3
7	60	1	3	0.3	1
8	60	1	4	0.3	2
9	50	1	3	0.3	3
10	40	3	4	0.3	2
11	40	2	4	0.5	3

根据式(2)，计算得到上述 11 个服务的初始重要度如表 3 所列。

表 3　实例服务的初始重要度

Table 3　Initial importance of example service

Software	Importance Classification
1	0.05490
2	0.06000
3	0.04788
4	0.13130
5	0.10350
6	0.12430
7	0.07029
8	0.80770
9	0.07721
10	0.09081
11	0.09992

在上述 11 个服务中，服务 4 的初始重要度最高为 0.1313，由表 2 可知该服务的软件规模和结构复杂度也最高；其次是服务 5 和服务 6；服务初始重要度较低的服务为服务 1、服务 2 和服务 3。通过分析特征值可知，上述 3 个服务的规模较小，同时封装的功能数较少，在历史软件测试的出错率也较低。

4.2　软件测试优先度计算

根据 3.3.1 节对测试依赖图模型的定义，结合银行管理系统结构中的服务调用情况，得到服务间的测试依赖图，如图 3 所示。图 3 中，服务 4 和服务 10 的调用关系最多，节点 4 的入度为 4，节点 3 的入度为 3，因此，节点 4 和节点 10 的影响力较大，系统中一旦发生错误，受其影响的节点数就较多。

节点 1 和节点 3 的出度均为 3，且初始重要度相近，但由于节点 3 依赖于节点 5 和节点 6 两个初始重要度较高的节点，因此节点 1 的复杂性高于节点 3。

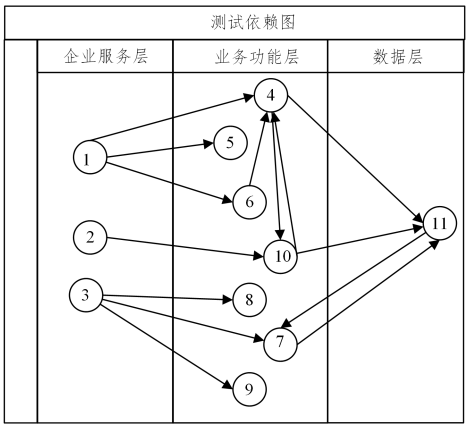

图 3 实例服务软件测试依赖图

Fig. 3 Example service software test dependency diagram

以上述服务依赖图为依据，利用算法 1 得到 11 个服务的集成测试优先度 P。该算法输入的邻接矩阵 $\boldsymbol{A}$、初始节点影响性 x 和初始节点复杂性 y 如图 4 所示，经过计算后得到的最终的测试优先度向量 $\boldsymbol{P}$ 如图 5 所示。

$$\boldsymbol{A}=\begin{pmatrix}0&0&0&1&1&1&0&0&0&0&0\\0&0&0&0&0&0&0&0&0&0&1&0\\0&0&0&0&0&0&1&1&1&0&0\\0&0&0&0&0&0&0&0&0&0&1&0\\0&0&0&0&0&0&0&0&0&0&0&0\\0&0&0&1&0&0&0&0&0&1&0\\0&0&0&0&0&0&0&0&0&0&0&1\\0&0&0&0&0&0&0&0&0&0&1&0\\0&0&0&0&0&0&0&0&0&0&0&0\\0&0&0&1&0&0&0&0&0&1&1\\0&0&0&1&0&0&1&0&0&0&0\end{pmatrix}\quad x=y=\begin{pmatrix}0.05490\\0.00600\\0.04788\\0.13130\\0.10347\\0.12433\\0.07029\\0.08077\\0.07021\\0.09081\\0.09992\end{pmatrix}$$

图 4 算法 1 的输入

Fig. 4 Input of algorithm 1

$$\boldsymbol{P}=\begin{pmatrix}0.29115\\0.00047497\\0.10889\\0.41665\\0.12662\\0.30763\\0.21064\\0.047877\\0.047402\\0.23718\\0.37529\end{pmatrix}$$

图 5 实例的测试优先度向量

Fig. 5 Test priority vector of an example

节点 4 的测试优先度最高，为 0.41665，其次是节点 11、节点 6 和节点 1，测试优先度得到的结果与在测试依赖图中进行的分析相符。上述 4 个节点的错误影响力和传播力较大，在集成测试中需要重点测试。

4.3 服务测试序列生成

本文的实例系统规模较小，只有 11 个服务。根据文献[8]中对遗传算法种群规模和迭代次数的研究，本实验设置种群规模为 22，迭代次数为 100。由于算法中采用精英保留选择算子，为了避免陷入局部最优，设置交叉率为 0.8，变异率为 0.3。利用本文提出的基于优先级初始种群的遗传算法进行实例研究的结果如图 6 和表 4 所示。图 6 给出了 100 次迭代后最优个体的目标函数值和种群个体的平均目标函数值，横坐标为迭代次数，纵坐标为目标函数值。表 4 列出了本次算法得到的最佳测试序列及其测试桩的总体复杂度。

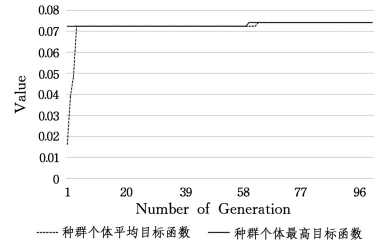

图 6 种群个体最优/平均目标函数值

Fig. 6 Optimal/average objective function value of individual population

表 4 最佳测试序列及其测试桩的总体复杂度

Table 4 Optimal test sequence and total complexity of test pile

Best test order	Test piles complexity
10-4-11-7-3-9-8-6-1-5-2	0.281

本文实验得到的最佳测试序列中前 3 的节点的测试优先度次序分别为 5，1，2，说明当测试完系统中 30% 的节点时，已经有效地测试到了测试优先度排在前 50% 的相对重要的节点。本文最佳序列需要构建的测试桩的复杂度占总体测试桩复杂度的 28%，说明本文的方案能够在一定程度上通过减小构建测试桩的复杂度来降低测试成本。结果表明本文算法得到的 SOA 架构中服务的集成测试顺序保障了重要节点优先测试，降低了测试桩总体复杂度，验证了本文算法的有效性。

结束语 通过分析 SOA 架构和 SOA 中服务软件自身的特点，本文提出了服务特征组及服务初始重要度的概念，将服务之间所有的动态服务调用关系抽象后得到测试依赖图，结合在依赖图中节点重要性互相依赖和影响的特点，提出了面向测试依赖图的软件测试优先度计算方法，将服务的自身重要度和各邻接服务的自身重要度相结合，迭代得到服务的测试优先度。选取降低测试桩服务复杂度和测试优先度高的服务优先集成两个因素作为测试序列生成的目标。利用基于优先级初始化种群的遗传算法，避免初始化种群的盲目性，生成最终的测试序列。通过实例分析，验证了本文方法的可行性，其能够被应用于实际工程。本文研究的实例对象中服务软件间的交互关系较为简单且服务软件数量较少，当服务软件数量过多时，测试依赖图和特征组的构建可能会存在一定的困难。由于对 SOA 架构的集成测试序列研究较少，因此无法设计实验来验证本文方法的有效性，在后续工作中将进行进一步调研并进行实验设计。本文仅考虑了 SOA 架构中服务软件间的调用关系，还需要在以后的工作中进行进一步的研究。

参 考 文 献

[1] NN A, WI A, IG B, et al. Understanding Service-Oriented Archi-

tecture(SOA):A systematic literature review and directions for further investigation [J]. Information Systems, 2020, 91: 101491.

[2] ERL T,GEE C,NORMANN H,et al. Next generation SOA:A concise introduction to service technology & service-orientation [M]. Pearson Education,2014.

[3] ZHANG N P,CHEN X Q. Software testing technology [J]. Microcomputer Development,2005(7):69-72.

[4] ZHANG Y M,JIANG S J,ZHANG M,et al. Review of class test sequence generation techniques in integration testing [J]. Acta Computa Sinica,2018,41(3):670-694.

[5] KUNG D C,GAO J,PEI H,et al. Class Firewall,Test Order, and Regression Testing of Object-Oriented Programs[J]. JOOP-Journal of Object-Oriented Programming,1995,8(2):51-65.

[6] ZHOU Y,SONG J H. Research on object oriented integration test sequence [J]. Computer Measurement and Control,2010, 18(9):2014-2015,2018.

[7] WANG Y,YU H,ZHU Z L. Integration test sequence generation method based on software node importance [J]. Computer Research and Development,2016,53(3):517-530.

[8] BRIAND L C,FENG J,LABICHE Y. Experimenting with genetic algorithms to devise optimal integration test orders[C]// Software Engineering with Computational Intelligence. Springer,Boston,MA,2003:204-234.

[9] BRIAND L C,JIE F,LABICHE Y. Experimenting with Genetic Algorithms and Coupling Measures to Devise Optimal Integration Test Orders ABSTRACT[C]// International Conference on Software Engineering & Knowledge Engineering. DBLP,2011.

[10] ZHANG Y M,JIANG S J,CHEN R Y,et al. Class integration-test sequence determination method based on particle swarm optimization [J]. Journal of Computer Science,2018,41(4):931-945.

[11] ASSUNÇÃO W,COLANZI T E,VERGILIO S R,et al. A multi-objective optimization approach for the integration and test order problem[J]. Information Sciences,2014,267:119-139.

[12] ZHANG Y N. Research on class integration test sequence generation method based on evolutionary optimization [D]. Xuzhou: China University of mining and technology,2019.

[13] YANG L L,LI B X. Survey of web service testing [J]. Computer Science,2008,35(9):258-265.

[14] HUANG N,YU Y,ZHANG D Y. Research and implementation of web service software testing technology [J]. Computer Engineering and Application,2004(35):147-149.

[15] CANFORA G,DI PENTA M. Testing services and service-centric systems:challenges and opportunities[J]. It Professional, 2006,8(2):10-17.

[16] ZHANG M,KEUNG J W,CHEN T Y,et al. Validating class integration test order generation systems with Metamorphic Testing[J]. Information and Software Technology,2021,132: 106507.

[17] XUAN G N,CHENG R W. Genetic algorithm and engineering optimization [M]. Beijing:Tsinghua University Press,2004.

[18] QIN H B,LI D L,GUO L,et al. Software architecture complexity measurement method based on complex network [J]. Microelectronics and Computer,2013,30(2):5-8.

[19] CZIBULA G,CZIBULA I G,MARIAN Z. An effective approach for determining the class integration test order using reinforcement learning[J]. Applied Soft Computing,2018,65:517-530.

[20] GUIZZO G,BAZARGANI M,PAIXAO M,et al. A hyper-heuristic for multi-objective integration and test ordering in google guava[C]// International Symposium on Search Based Software Engineering. Cham:Springer,2017:168-174.

ZHANG Bing-qing, born in 1998, postgraduate. Her main research interests include software testing and so on.

WANG Yi-chen, born in 1977, senior engineer, associate professer. His main research interests include model-based software testing, software quality evaluation, etc.

(责任编辑:李亚辉)