文章编号：2096-1472(2022)-12-59-04
DOI:10.19644/j.cnki.issn2096-1472.2022.012.012

融合智能优化的软件测试用例生成方法

党向盈，李金凤

(徐州工程学院信息工程学院，江苏 徐州 221018)
✉dangpaper@163.com; 41407770@qq.com

摘　要：针对复杂软件中测试用例难生成问题，提出一种融入聚类和进化算法的软件缺陷测试方法，开发一套智能软件测试系统。首先，对被测程序采用变异测试方法注入缺陷；基于不同策略对缺陷聚类。然后，针对多个缺陷簇，建立测试用例生成问题的优化模型，并采用进化算法生成能检测缺陷的测试用例。最后，基于不同评价指标，完成软件测试报告。测试结果表明，基于测试用例检测缺陷的成功率、缺陷率，以及消耗时间和迭代次数等指标，验证了所提方法提高了检测缺陷检测率、降低了测试的时间及提高了测试用例生成的效率。由此可见，人工智能融于软件测试技术，不仅提升了软件测试效率，而且丰富人工智能应用领域。

关键词：软件测试；测试用例生成；聚类；进化算法
中图分类号：TP311　**文献标识码**：A

Generation Method of Software Test Case based on Intelligent Optimization

DANG Xiangying, LI Jinfeng

(*School of Information Engineering, Xuzhou University of Technology*, *Xuzhou* 221018, *China*)
✉dangpaper@163.com; 41407770@qq.com

Abstract: Aiming at the problem that test cases are difficult to generate in complex software, this paper proposes a software fault testing method incorporating clustering and evolutionary algorithms, and an intelligent software testing system is developed. First, mutation test is used to inject faults into the program under test, and the faults are clustered based on different strategies. Then, for multiple fault clusters, an optimization model of test case generation problem is established, and an evolutionary algorithm is used to generate test cases that can detect faults. Finally, the software test report is completed based on different evaluation indicators. The test results show that the proposed method improves the fault detection rate, reduces the test time and improves the efficiency of test case generation based on the success rate, fault rate, time consumption and iteration number of test case detection. It can be seen that the integration of AI (Artificial Intelligence) into software testing technology not only improves the efficiency of software testing, but also enriches the application field of AI.

Keywords: software testing; test case generation; clustering; evolutionary algorithm

1 引言(Introduction)

软件测试是软件开发生命周期中的一个重要环节，其目的是通过检测尽可能多的缺陷以保证软件质量[1]。然而，传统的软件测试方法很难检测大量潜在的缺陷。随着人工智能技术的发展，软件测试方法智能化程度越来越高。在进行软件测试时，将程序的输入作为测试用例[2]。对于程序中的缺陷，如何高效地获得测试用例检测它们，一直是很多学者的研究热点。

本文借鉴人工智能中的进化算法和聚类方法，构建软件测试平台，生成具有检测缺陷能力的测试用例。近年来，进化算法一直被广泛地应用于测试用例生成[3]。进化算法的进化策略包括选择、交叉、变异、种群竞争或合作等[3]。此外，聚

基金项目：江苏省高等学校自然科学研究重大项目(21KJA520006)；徐州市科技计划项目(KC21007).

类已经成为实施数据挖掘和分析的重要方式之一[4]，它是一种无监督机器学习的分类方法，将一些相似的对象形成一个个集合(簇)，根据数据在簇中是否有重叠数据，将聚类分为硬聚类和模糊聚类[5]。

考虑到程序中的缺陷隐蔽性较强，很多学者[5]提出变异测试概念，它是通过主动出击的方式，模拟程序中真实的缺陷，也就是对程序中的原语句进行变异，被改动后的语句为变异语句，变异之后的程序为变异体[6]。

鉴于以上分析，本文借鉴变异测试技术[7]，以“主动出击”的方式，在程序中植入变异语句作为潜在的缺陷；再采用进化算法和聚类方法，生成具有检测缺陷能力的测试用例，并开发一套智能软件测试系统。

2 智能软件测试系统总体框架 (Framework of intelligent software test system)

图1为本文所提智能软件测试系统总体框架，测试主要分为五步，下面结合可视化“智能软件测试平台”(图2)，阐述实施过程。

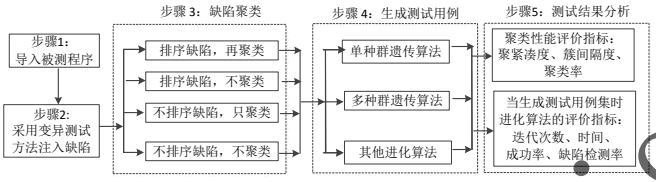

图1 智能软件测试流程

Fig.1 Process of intelligent software testing

首先，登录测试平台，点击“导入被测程序”的“缺陷注入”，可以实现测试对象的预处理。“缺陷注入”功能是通过自动化变异测试工具实现的，对被测语句实施变异，注入一些缺陷(生成变异体)。

其次，对于注入的众多缺陷，点击“缺陷聚类”，启动编程环境。如图1所示，“缺陷聚类”可以通过四种不同的聚类策略实现。同时，可以采用经典硬聚类和模糊聚类方法聚类缺陷。首先设置进化算法的参数，包括种群数目、种群规模和最大迭代次数等，单击“生成测试用例”，启动编程环境。算法1给出经典模糊聚类方法实现缺陷聚类的步骤。然后采用进化算法生成测试用例。先设置进化算法的参数，单击“生成测试用例”，启动编程环境。输出生成测试用例的时间、迭代次数和测试用例，以及成功率和缺陷检测率。

最后，分析测试结果，主要从两个方面评价各种方法的性能。当采用不同聚类方法和策略实现众多缺陷的聚类时，通过三种指标评价聚类性能。当采用进化算法生成测试用例时，通过四种评价指标考察算法的性能。对比测试结果和性能指标，给出最优方法并进行分析，完成软件测试报告。

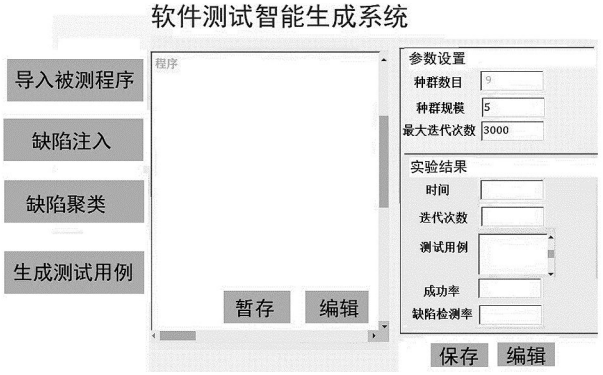

图2 智能软件测试平台

Fig.2 System for intelligent software testing

3 聚类缺陷(Faults clustering)

研究表明，缺陷之间是有关联的，也就是检测到一个缺陷的测试用例，同时可能检测到其他缺陷[8]。而且缺陷的重要度不一样，因此为了保证聚类缺陷的性能，需要考虑缺陷重要度和缺陷之间的关联度。为了评价聚类缺陷的性能，采用四种策略聚类缺陷，分别如下。

(1)RC：首先基于缺陷重要度排序缺陷，以最重要的缺陷为聚类中心，然后基于缺陷之间的关联，对缺陷聚类；最后采用进化算法生成测试用例。

(2)R￢C：缺陷只进行排序，不进行聚类。排序好的缺陷序列中，任意选一个缺陷采用进化算法生成测试用例；判断该测试用例是否能够检测其他缺陷；反复执行此过程，直到满足终止准则。

(3)￢RC：缺陷没有进行排序，直接进行聚类。在缺陷集合中，首先随机选择某一缺陷作为簇中心，然后聚类缺陷，最后生成测试用例。

(4)￢R￢C：缺陷没有排序和没有聚类。从缺陷集合中，一个接一个地随机选择缺陷，作为优化目标，生成检测它们的测试用例；判断该测试用例是否能够检测其他缺陷；反复执行此过程，直到满足终止准则。

对于以上四种策略中的聚类方法，可以采用硬聚类和模糊聚类方法对缺陷分类。此外，为了进一步考察聚类缺陷的性能，即评价不同方法的性能，选择一些评价指标。

记缺陷集合为$F=\{F_1,F_2,\cdots,F_n\}$，基于它们的检测难度降序排列，得到有序缺陷集合，记为$F'=\{F_1',F_2',\cdots,F_n'\}$，其中$F_1'$是最难杀死变异体。基于缺陷之间的关联度，构建关联矩阵，记为Λ。

算法1为模糊聚类缺陷算法，输入为缺陷集合$F=\{F_1,F_2,\cdots,F_n\}$和缺陷之间关联矩阵Λ，输出为若干缺陷组成的簇集合$C_1, C_2, \cdots, C_m$。

首先，对缺陷集合的缺陷按照它的覆盖难度排序，得到

有序集合$F^{'}=\{F_1^{'},F_2^{'},\cdots,F_n^{'}\}$；再从中选出首元素$F_1^i$作为聚类中心，更新簇$C_i=\{F_1^i\}$，同时从$F_1^i$中删除$F_1^i$。

然后，记F_1^i与F_j的关联度值$\alpha_{i,j}$，其中$F_1^i,F_j\in F$，$j=1,2,\cdots,n,F_j\neq F_1^i$；考察$\alpha_{i,j}$中对应的行$F_1^i$，如果$\alpha_{i,j}\geqslant T$(阈值)，可以把$F_j$放入$C_i$；以此类推，得到以$F_1^i$为中心的簇$C_i=\{F_1^i,F_2^i,\cdots,F_{|C_i|}^i\}$，其中$|C_i|$为簇中元素的个数，$F_k^i$为第$i$个簇中第$k$个元素，同时将从$F_1^i,F_2^i,\cdots,F_{|C_i|}^i$从$F^{'}$中删除。以上操作重复，再从$F^{'}$选首元素，直到$F^{'}$为空集。

最后，输出所有的簇，记为$C_1,C_2,\cdots,C_m$，其中$F_1^1,F_1^2,\cdots,F_1^m$为每个簇的聚类中心，m为簇的个数。

4 基于进化算法生成测试用例(Test case generation based on evolutionary algorithm)

4.1 优化模型构建

为了采用进化算法生成测试用例，需要构建测试用例生成问题的优化模型。

(1)目标函数。针对某一个缺陷F_i，生成能检测F_i的测试用例，作为进化算法优化的目标。因此，对于F_i，设目标函数为$f(X)$，其中X为决策变量，即程序的输入，也是实验中需要找到的测试用例。当X能检测到F_i，$f(X)=0$；否则$f(X)=1$。因此，当且仅当$f(X)=1$取最小值0时，X检测到F_i。通过这种方式，生成检测到F_i测试用例的问题，转化为求$f(X)$最小值问题，可以表示为$\min f(X)$。

然而，$f(X)$的取值只有0和1两种，显而易见，如此目标函数很难引导种群的进化。为了提供更多的信息指导种群进化，需要为优化模型增加一个约束。

(2)约束函数。研究表明，在检测一个变异体之前，测试用例首先可达缺陷位置，也就是说，在检测缺陷之前，测试用例必须先覆盖被测语句。因此，基于分支覆盖的约束函数，可以表示如下：

$$g(X)=Appr(\theta,X)+(1-1.001^{-dist(\theta,X)}) \tag{1}$$

其中，$Appr(\theta,X)$是X对于θ的层接近度，$dist(\theta,X)$为分支距离。由式(1)可知，当且仅当$g(X)=0$时，X能覆盖θ。

(3)优化模型。根据X、$f(X)$和$g(X)$，建立生成能检测缺陷F_i的测试用例的优化模型如下：

$$\begin{aligned}&\min\ f(X)\\&\text{s.t.}\begin{cases}g(X)=0\\X\in D\end{cases}\end{aligned} \tag{2}$$

其中，D为程序的输入域。

4.2 生成测试用例

对于上面的优化模型，采用进化算法生成测试用例。遗传算法(GAs)已经被广泛地应用于测试用例生成。一般而言，单种群遗传算法执行一次，只能针对一个优化目标，也就是只能为一个缺陷生成测试用例。显然，对于多个簇的缺陷，单种群遗传算法的效率比较低下。考虑到缺陷已经被划分为多个簇，可以采用多种群遗传算法生成测试用例。

考虑到优化模型中包括一个目标函数和一个约束函数，适应值函数可以表示如下：

$$fit(X)=f(X)\times(g(X)+\varphi) \tag{3}$$

其中，φ是很小的正整数，它的作用是确保括号里的值大于0。由式(3)可知，当且仅当$fit(X)=0$时，X能检测到缺陷F_i。

在多种群遗传算法中，一个种群包含m个子种群，分别处理m个簇$C_1,C_2,\cdots,C_m$中缺陷的测试用例生成，所有的子种群并行进化，第i个子种群的进化个体为$X_1^i,X_2^i,\cdots,X_{Size}^i(i=1,2,\cdots,m)$，其中$Size$为子种群中进化个体个数。算法的输出为生成的测试用例集。终止条件有两个，一个是对于m个簇的缺陷，期望的测试用例全部找到；另一个是种群进化到最大进化代数g。

算法2为基于多种群遗传算法生成测试用例，输入为种群(包括m个子种群)和m个簇；输出为测试用例集T。

首先，初始化m个子种群和算法中的各种参数，并设变量$count=1$；对于簇C_i，每个进化个体$X_1^i,X_2^i,\cdots,X_{Size}^i(i=1,2,\cdots,m)$，都执行中心缺陷$F_1^i$，判断终止条件是否满足，如果没有满足，再用$C_i$对应进化个体$X_1^i,X_2^i,\cdots,X_{Size}^i$检测其他簇中缺陷是否能检测；如果能，终止第$i$个子种群的进化，保存测试用例；如果不满足，进化个体进行遗传操作，进行选择、交叉、变异操作。其次，所有的簇采用同样的策略，每个簇以并行方式完成对应簇内缺陷的测试用例生成。最后，对所有簇输出测试用例集。

测试时，实现随机方法、单种群遗传算法和多种群遗传算法生成测试用例。单种群遗传算法与多种群遗传算法生成测试用例的不同之处在于，单种群生成测试用例时，每个簇以串行的方式，采用遗传算法生成能检测簇中心的测试用例。采用随机法生成测试用例时，一个一个簇依次采用串行方式生成测试用例。

考虑到多种群遗传算法的最大迭代次数为3,000，那么采用随机方法时，随机生成测试用例为3,000次，然后执行缺陷。单种群遗传算法的参数设置与多种群遗传算法相同，它们的区别在于，单种群遗传算法只有一个子种群，一次只能优化一个目标；多种群遗传算法有多个子种群，一个子种群优化一个簇中所有的缺陷。多种群遗传算法的优势在于，在每一次迭代中，多种群遗传算法执行一次，几个子种群能够以并行方式检测不同簇中的多个缺陷。相比之下，基于随机方法和单种群遗传算法，测试用例是一个接着一个依次生成的。

为了比较三种方法生成测试用例的性能，选择时间消耗、迭代次数、缺陷检测率、成功率这几种评价指标[4]。一般而言，如果生成测试用例的时间消耗越少，则对应的方法性能越好；迭代次数越少，则对应方法的效率越高。

缺陷检测率可以表示如下：

$$FS=\frac{\text{被检测到的缺陷}}{\text{所有缺陷}} \tag{4}$$

此外，假设某种算法运行了W次，其中V次成功地找到检测缺陷的期望数据，那么，成功率可以定义如下：

$$SR=\frac{V}{W} \tag{5}$$

由式(5)可知，成功率越高，对应的算法的搜索性能越好。

5 软件测试报告(Software testing reports)

本文所提软件测试方法，结合了人工智能算法中的聚类、遗传算法和多种群遗传算法，对于隐蔽性强的缺陷，采用变异测试技术，模拟真实缺陷，生成能检测缺陷的测试用例。开发的测试平台可以应用于一般的基准程序和工业程序，测试范围领域广泛且数据类型、逻辑结构、功能和规模多种多样。

实验中，被测程序G1—G8分别在智能软件测试平台(图2)上执行，获得的测试结果，然后比较分析各个方法的性能，并撰写软件测试报告。对八个被测程序，采用四种不同聚类策略获得的缺陷检测率的结果如图3所示。由图3可知，先基于缺陷重要度排序缺陷，以最重要的缺陷为聚类中心的RC方法获得缺陷检测率最高；基于R⁀C和⁀RC方法获得的缺陷检测率相差不多；没有排序和没有聚类的⁀R⁀C方法获得的缺陷检测率最低。这说明，对缺陷进行聚类和排序，有助于缺陷检测率的提高。

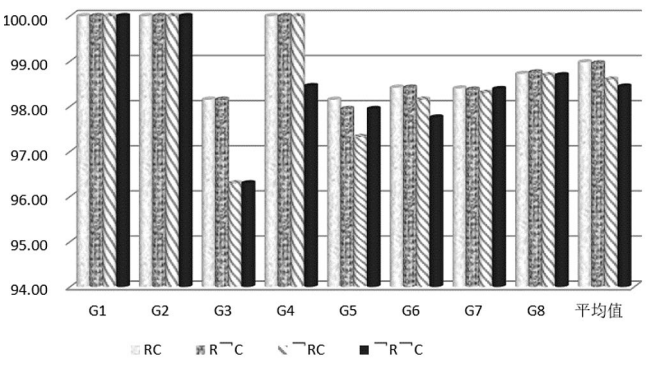

图3 基于四种聚类策略缺陷检测率

Fig.3 Fault detection rate based on four clustering strategies

图4和图5为基于不同算法生成测试用例的时间消耗和迭代次数。从测试结果可以看出，进化算法中的单种群和多种群遗传算法采用全局搜索方式，大大提升了测试用例生成的效率，而且多种群遗传算法利用个体之间信息共享方式，能够提高测试数据找到的速度和迭代次数。

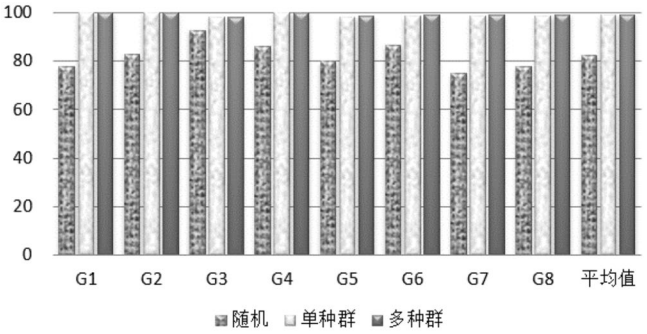

图4 采用三种方法生成测试用例时的时间消耗(s)

Fig.4 Time consumption of generating test case by the three methods (s)

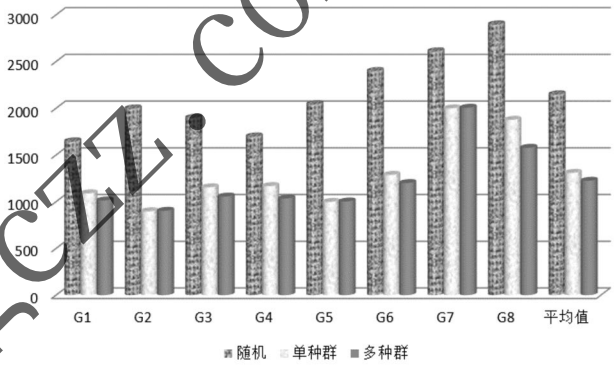

图5 采用三种方法生成测试用例时的迭代次数

Fig.5 The number of iterations of generating test case by the three methods

6 结论(Conclusion)

智能软件测试平台融合了机器学习中的聚类方法和进化算法中的遗传算法，应用于八个经典的程序和工业软件，从实验数据可以看出，所提方法和开发的测试系统，通过多种群遗传算法能够获得更好的缺陷检测率。此外，通过时间消耗和迭代次数等指标验证了所提方法能高效、快捷地找到检测缺陷的测试用例。由此可见，本文方法是人工智能、自动化、计算数学及计算机多学科交叉，实现提高软件测试效率的目的。该平台具有较好的扩展性，可以与其他综合实验灵活地融合使用，具有一定的推广价值和应用前景。

参考文献(References)

[1] DURELLI V H, DURELLI R S, BORGES S S, et al. Machine learning applied to software testing: a systematic mapping study[J]. IEEE Transactions on Reliability, 2019, 68(3): 1189-1212.

(下转第49页)