Computer Science and Application 计算机科学与应用, 2019, 9(9), 1716-1723
Published Online September 2019 in Hans. http://www.hanspub.org/journal/csa
https://doi.org/10.12677/csa.2019.99192

Hans汉斯

Software Testing Based on the Fault Propagation Path Coverage

Zhiqin Cao[1], Yichen Wang[1], Kun Wang[2]

[1]Science and Technology on Reliability and Environmental Engineering Laboratory, Beihang University, Beijing
[2]China Aerospace Science and Industry Corporation Third Research Institute, Beijing
Email: caozhiqin@buaa.edu.cn

Received: Aug. 28th, 2019; accepted: Sep. 12th, 2019; published: Sep. 19th, 2019

Abstract

The correlated defects are uncertain, and some existing software testing technique and strategies are difficult to find such defects. This paper proposes a software testing technique for fault propagation path coverage. Based on the analysis of empirical and historical data, this method speculates and generalizes the defect characteristics and types, thereby cultivating seed defects that can induce these defects. It is injected into the software code by designing the algorithm, to find test cases that can activate these seed defects, and then use the resulting test cases as input to activate the defects and cover their propagation paths. It was eventually discovered that potentially associated defects were found on the propagation path.

Keywords

Correlated Defects, Fault Propagation Paths, Test Case Generation, Improved Simulated Annealing Algorithm (ISAA)

基于故障传播路径覆盖的软件测试

曹志钦[1]，王轶辰[1]，王　坤[2]

[1]北京航空航天大学可靠性与环境工程国防重点实验室，北京
[2]中国航天科工集团第三研究院，北京
Email: caozhiqin@buaa.edu.cn

收稿日期：2019年8月28日；录用日期：2019年9月12日；发布日期：2019年9月19日

文章引用: 曹志钦, 王轶辰, 王坤. 基于故障传播路径覆盖的软件测试[J]. 计算机科学与应用, 2019, 9(9): 1716-1723.
DOI: 10.12677/csa.2019.99192

摘　要

关联缺陷具有不确定性，现有的一些软件测试方法与测试策略难以发现此类缺陷。本文提出了一种面向故障传播路径覆盖的软件测试方法，该方法在分析经验与历史数据的基础上，对缺陷特征和类型进行推测和归纳，从而培育出能够诱发这些缺陷的种子缺陷，并将其注入到软件代码中。通过设计算法，找到能够激活这些种子缺陷的测试用例，再以得到的测试用例作为输入，激活缺陷并覆盖其传播路径。最终在传播路径上发现潜在的关联缺陷。

关键词

关联缺陷，故障传播路径，测试用例生成，改进的模拟退火算法(ISAA)

Copyright © 2019 by author(s) and Hans Publishers Inc.
This work is licensed under the Creative Commons Attribution International License (CC BY).
http://creativecommons.org/licenses/by/4.0/
Open Access

1. 引言

如今，软件在计算机系统中起着日益重要的作用，而随着人们对软件的依赖性越来越强，软件的可靠性问题在软件工程领域的重要性显得愈发明显。因此，软件测试在软件的生命周期中所占的比重越来越高。在对软件测试的过程中有学者发现，软件缺陷之间存在某种关联关系。

Katerina 和 Trivedi [1]引入了失效关联的概念，认为在实际测试过程中软件失效不是相互独立的，并且提出了一种 Markov 更新模型来对有失效关联的软件可靠性进行建模。Chen [2]等人认为测试回合不是相互独立的，在此基础上提出了一种二进制 Markov 过程模型，用来预测随机测试策略发现的软件失效数。Bishop [3]等人指出可以用失效屏蔽效应来解释软件失效之间的关联关系，并可以通过适当的软件设计来杜绝失效关联。Mehmet [4]则根据失效关联/分支覆盖关联定义了一种 Bayesian 测试停止规则，以决定何时停止测试。

在软件缺陷层面，景涛[5]提出了关联缺陷的定义，并将关联缺陷应用于软件检测过程中，提出了一种缺陷放回的测试方法来剔除关联缺陷。但是没有对缺陷之间关联的形式进行建模，也没有提出一种针对关联缺陷的测试方法。刘新忠[6]进一步推进了关联缺陷的研究，他认为关联缺陷的存在很大程度上是由于缺陷的检测能力被其它缺陷所屏蔽，并且引入了关联缺陷的 P-NHPP 模型来对关联缺陷进行检测和评估，但是对这类缺陷的预测和最后的验证没有系统的算法和方法。

在测试方法层面，宋想[7]提出了一种基本路径覆盖的测试用例生成方法，有效解决了软件测试中部分路径不可达的问题，却没有考虑关联缺陷并不仅存在于某一条基本路径，因此基于基本路径覆盖所生成的测试用例会约束缺陷检测率的提高。

综上所述，针对这种缺陷的预测，我们将使用引入种子缺陷的方法，来诱发与之相关联的缺陷，使其成为可测的故障；而针对最后的验证我们将提出一种基于故障传播路径覆盖的测试方法，利用智能算法测试用例自动生成的功能不断尝试输入，尽可能多地覆盖故障传播路径，在这一过程中发现更多的缺陷。最后，设计对比实验，验证相比较全路径覆盖测试法和基本路径覆盖测试法，基于故障传播路径覆盖的缺陷测试法可以用最少的测试用例发现最多的缺陷数量。

2. 相关知识

2.1. 基于路径覆盖的软件测试方法

2.1.1. 基于路径测试方法

文献[8]程序的路径是程序中顺序执行的一个语句序列，是由控制流图中的一系列节点组成。在实际测试中，一个不太复杂的程序，其路径数都是一个庞大的数字。为了解决这一难题只得把路径数压缩到一定限度内，如程序中的循环体只执行一次。因此，基本路径集满足了实际测试过程中的这种需求。基本路径集具有以下特点：1) 每一条路径都是一条独立路径；2) 程序中所有的边都被访问；3) 程序中的所有的、不属于该基本路径集的路径都可以由这个基本路径集中的路径经过线性运算得到。

基本路径覆盖的测试方法可以描述为：将覆盖程序的基本路径集作为测试的目标，在程序的输入空间中寻找测试数据，使得该测试数据为输入，运行程序后执行基本路径的过程。

2.1.2. 故障传播路径测试法

故障传播路径，即对缺陷演化过程的一种描述方式。当软件系统中某个节点发生故障时，可能会逐步向其他节点扩散。在故障扩散的过程中故障会有限选择传播概率较大的路径进行传播(传播概率可以从故障历史数据中提取，也可以根据系统参数进行估计) [9]。我们把这种发生一系列故障的节点所在的路径就叫做故障传播路径。

故障传播路径测试法就是通过引入培育的种子缺陷，使其能够激发一系列的潜在缺陷，继而设计测试用例，运行程序后执行这一系列缺陷所在的故障传播路径的方法。该方法的优势在于引入种子缺陷之后能够引发直接运行程序不容易发现的潜在缺陷，因此相比较其他的路径测试法，可以发现更多的缺陷。

2.2. 故障传播路径的建模

故障传播模型的建立基于两个基本前提：1. 测试过程中在一个缺陷被激活变为故障继而演化为失效的过程中更容易激发其它的缺陷，即缺陷具有动态群聚性。2. 关联缺陷的发现需要首先激活一个缺陷，才能触发与之相关联的缺陷[5]。

3. 基于故障传播路径覆盖的缺陷测试方法原理

基于上一部分叙述的两个前提，我们提出了故障传播路径覆盖的软件测试方法原理。

(1) 为了发现新的缺陷，我们采用缺陷注入的方法，引入种子缺陷，并设计测试用例激活该种子缺陷使之成为故障，在覆盖该故障传播路径的过程中发现新的未知缺陷。

(2) 缺陷注入的关键在于培育典型缺陷，在这个过程中运用了测试用例设计中的猜错的方法。所谓典型缺陷就是通过传播诱发新缺陷的能力强，且类型与被测软件特点相吻合。这些特点的来源可以是以往对类似软件测试的经验或者从已有的缺陷类型总结中提取。

(3) 通过借鉴已有的软件建模形式(例如状态图、程序切片或者复杂网络)对被测软件进行建模，从而找到一种对传播路径进行准确便捷的描述方式。在本文中，由于我们是研究基于路径覆盖的测试方法，因此选用控制流图的形式对软件进行建模。

(4) 利用智能算法的自动生成测试用例的功能，不断尝试用例输入，激活缺陷，执行覆盖故障传播路径，在此过程中发现更多的潜在缺陷，从而比其他测试方法发现更多的缺陷。

4. 算法介绍

在测试过程中，需要不断尝试输入，激活注入的缺陷使之成为故障，并且需要判断每次的输入是否

覆盖了故障传播的路径，以及在该传播路径上是否发现了新的缺陷。这个过程需要大量的设计测试用例，而文献[10]中提到了一种改进的模拟退火智能算法(SAA)可以自动生成测试用例，已有的模拟退火算法在软件测试用例的生成的应用中，对劣解的处理只是单纯的依概率接受，这导致了接受劣解的过程只是出于形式化的考虑，或者是很盲目的去寻找局部最优解之外的全局最优解。然而在实例运用中，由于效率和时间的限制，很少会出现最优解之外的全局最优解。改进的退火算法通过缺陷覆盖率和语句覆盖率结合的双重校验的思想，也就是在原有的外层状态接收函数中嵌套了另外一层状态接受函数，并且在迭代的过程中通过设置的判断条件对输入进行优化使之向我们需要的方向不断进化，这样就满足了我们的测试需求。因此在寻找新的缺陷的过程中我们使用了模拟退火智能算法，大大减少了测试人员的工作量。退火算法的基本流程介绍如图 1 所示。

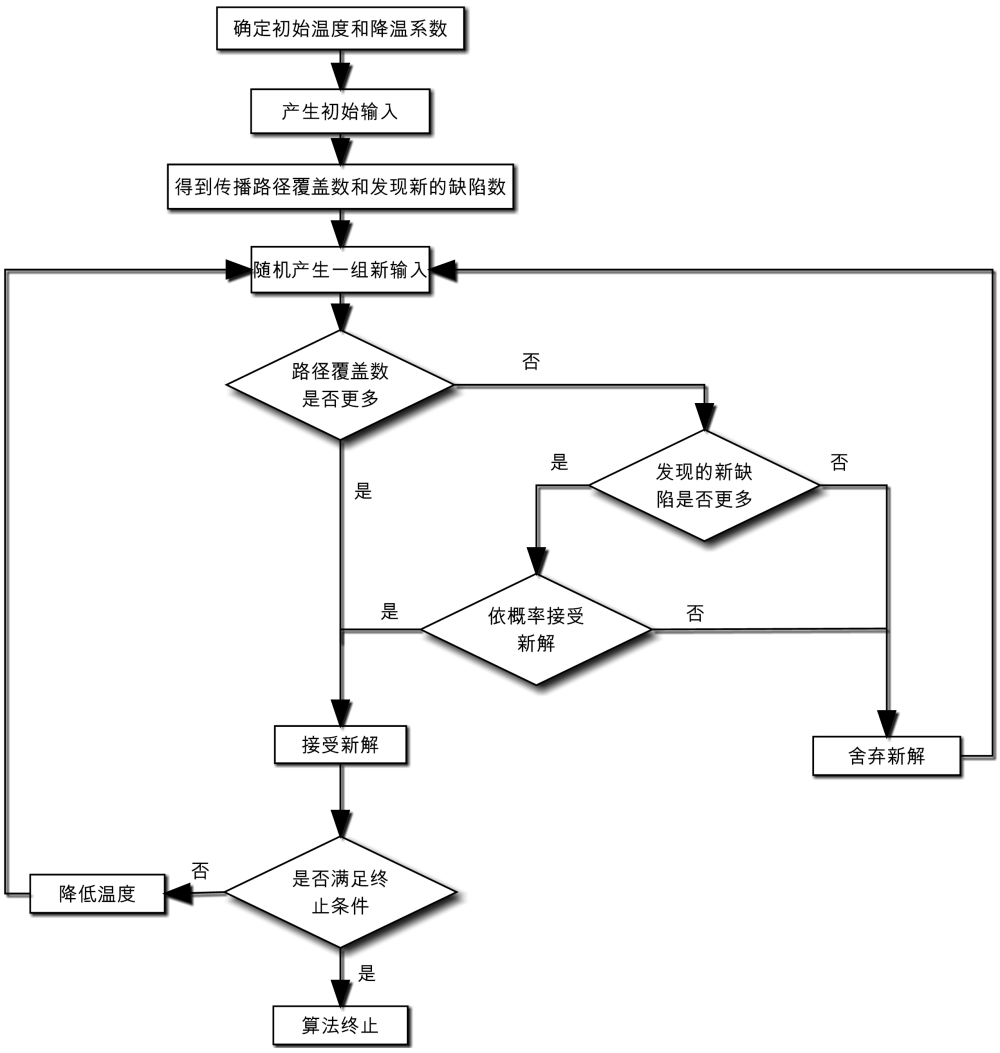

Figure 1. The flow of annealing algorithm
图 1. 退火算法流程

在该算法中，我们设定初始温度为 T = 100℃，降温系数为 k，每次降温幅度为 kT。因此，T2 = T1 − kT。随机产生初始测试用例集，运行程序之后得到结果，即覆盖的路径数和新发现的缺陷数。到此为止，第一次迭代结束。然后从输入域产生一组新的输入，运行程序。得到新的结果。判断是否覆盖了更多的

路径，如果是，则接受这一组新的输入，并且判断是否满足了算法的终止条件。如果满足，则算法终止。如果不满足，则降低温度，然后在当前输入的邻域内寻找新的测试用例集。再进行下一次迭代。如果新的输入没有覆盖到更多的路径，那么我们需要再次判断是否发现了更多的新的缺陷。如果是，则依照概率接受这一组新的输入。如果不是，则舍弃当前输入，在原输入的邻域内寻找新的测试用例集，开始下一次迭代。直到算法结束。

5. 对比实验的设计与结果分析

5.1. 被测软件介绍

本章实例验证选取的是 calendar 软件作为实验对象。由于该日历软件实现的功能较多，结构复杂，涉及过多操作类的测试用例，且本文对复杂软件系统测试方法研究深度有限，因此，只选取了其中一个 next date 模块进行测试。该模块的主要功能是输入一个日期序列，可以自动计算下一个日期并以序列形式输出。该模块的特点是程序结构清晰，路径数量适中，适合作为本章的实例进行该方法的验证。

5.2. 实验步骤

5.2.1. 分析软件结构

分析被测程序结构，导出控制流图，得到圈复杂度。控制流图如图 2 所示，其中，圈复杂度 = 独立路径条数 = 41。

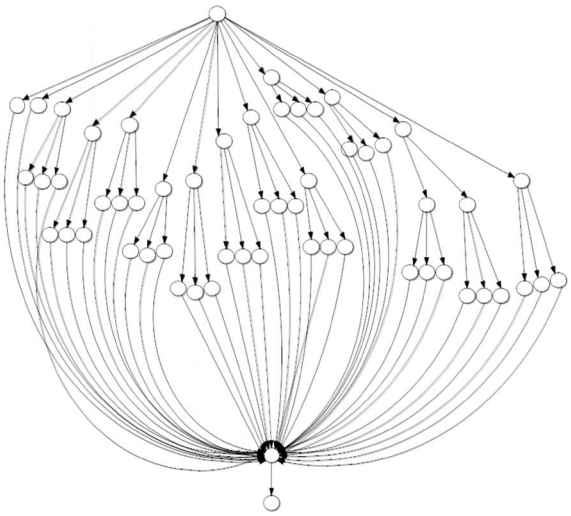

Figure 2. The flowchart of software under test
图 2. 被测软件控制流图

5.2.2. 基于软件需求的初始测试用例集设计

被测程序为 calendar 软件的一个 next date 功能模块，要实现的功能是随机输入一个 2000 年到 2999 年之间的日期，系统能够自动输出后一天的日期。那么在实验中我们给出的初始测试用例就是随机的在 2000 年到 2999 年之间取 41 个日期，作为三组对比实验的初始测试用例。

5.2.3. 算法对比实验

第一个实验：随机测试法(random)，其原理是用初始测试用例作为第一次实验输入，运行程序，得到路径覆盖率。将被覆盖路径所对应的测试用例集中的日期按照原有顺序保存，然后随机更新初始测试

用例集中其它的日期，作为第二次实验的输入，以此往复不断迭代生成新的测试用例集。直到路径覆盖率达到 100%，记录此时算法的迭代次数和最终得到的测试用例集。

第二个实验：模拟退火算法(SAA)，其原理是用初始测试用例作为第一次实验输入，运行程序，得到路径覆盖率。将被覆盖路径所对应的测试用例集中的日期按照原有顺序保存，然后给其余的日期一个微小的扰动，将年月日中的任意一个数字加一或者减一，得到一组新的测试用例集，作为第二次实验的输入。若第二次得到的覆盖率大于第一次，那么，以第二次的输入作为新的初始输入进行下一次迭代；如果第二次得到的覆盖率比第一次小，则依概率接受第二次初始输入，否则保留第一次输入。以此往复不断迭代生成新的测试用例集。直到路径覆盖率达到 100%，记录此时算法的迭代次数和最终得到的测试用例集。

第三个实验：改进的模拟退火算法(ISAA)，其原理是用初始测试用例作为第一次实验输入，运行程序，得到路径覆盖率。将被覆盖路径所对应的测试用例集中的日期按照原有顺序保存，然后给其余的日期一个微小的扰动，将年月日中的任意一个数字加一或者减一，得到一组新的测试用例集，作为第二次实验的输入。若第二次得到的覆盖率大于第一次，那么，以第二次的输入作为新的初始输入进行下一次迭代；如果第二次得到的覆盖率比第一次小，则先判断第二次的语句覆盖率大于第一次，如果是，则依概率接受第二次初始输入，否则保留第一次输入。以此往复不断迭代生成新的测试用例集。直到路径覆盖率达到 100%，记录此时算法的迭代次数和最终得到的测试用例集。

5.2.4. 路径覆盖对比实验

在软件测试中，圈复杂度可以用来衡量一个程序结构的复杂程度，数量上表现为独立路径的条数，即发现错误所需要测试的最少路径条数[11]。圈复杂度大说明程序代码测试和维护的难度越大。因此，该实验选取两个测试难度大小不同的程序，程序一为三角形判定程序，圈复杂度为 4，程序二是 next-date 程序，圈复杂度为 41。

两个被测程序中都存在着一定数量的已知缺陷，并且程序中分别注入了两个潜在缺陷。在用故障传播路径覆盖测试过程中引入了能够诱发其中两个潜在缺陷的种子缺陷。两种方法在每次迭代的过程中产生的测试用例数都等于圈复杂度。

5.3. 实验结果及分析

算法对比实验结果如图 3、图 4 所示，我们将第一组实验和第二组实验数据分别和第三组实验数据进行对比，其中横坐标为实验次数，纵坐标为迭代次数。

根据以上实验数据以及各组实验数据对比，我们可以得出以下几点结论：

1) 基于模型的测试技术与软件模型检验思想融合的集成测试框架将两者很好的结合起来，保证测试需求模型正确的同时，提高了测试用例的质量。

2) 第一组实验与第三组实验结果进行数据对比，我们可以看出，改进的模拟退火算法在覆盖同样多的路径时，所用的迭代次数比随机算法要少，即生成测试用例的效率要高。

3) 第二组实验与第三组实验结果对比可知，从方差上看，改进的模拟退火算法在测试用例生成方面的表现比传统的模拟退火要稳定。

路径覆盖对比实验结果如表 1，表 2 所示。

表 1 中的结果显示，基本路径覆盖测试法在软式过程中发现了 4 处缺陷，不包括注入的关联缺陷。而基于故障传播路径覆盖的测试法却能够在使用相同数量测试用例的情况下发现包括关联缺陷在内的所有已知缺陷。与表 1 中的结果相同，实验二同样证明了由于种子缺陷的引入和基于故障传播路径的覆盖，

本文提出的方法可以达到用最少的测试用例发现更多缺陷的目的。

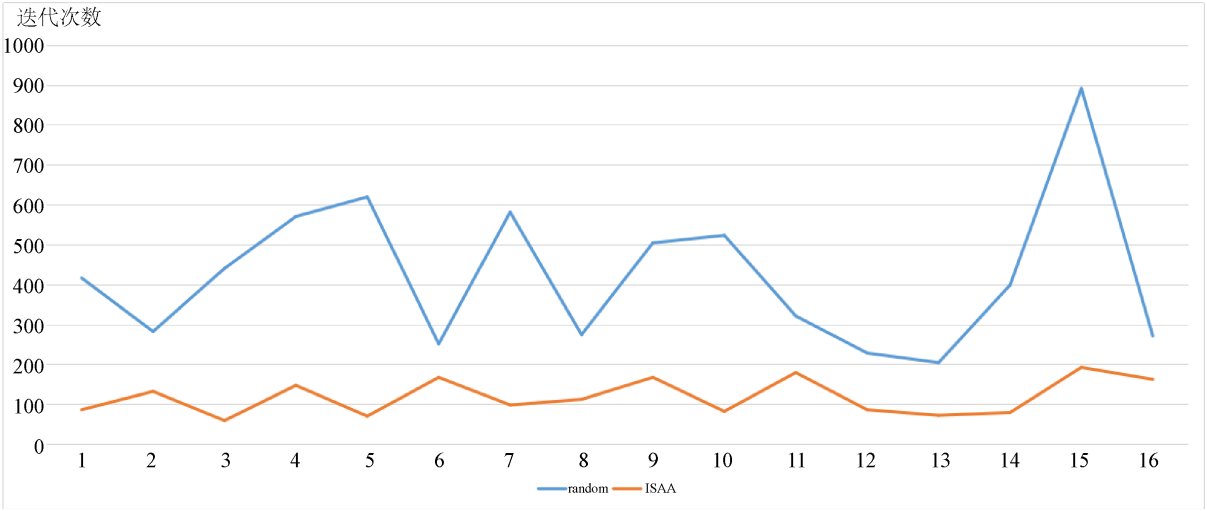

Figure 3. Comparison curve of the first and third sets of experimental results
图 3. 第一、三组实验结果对比曲线

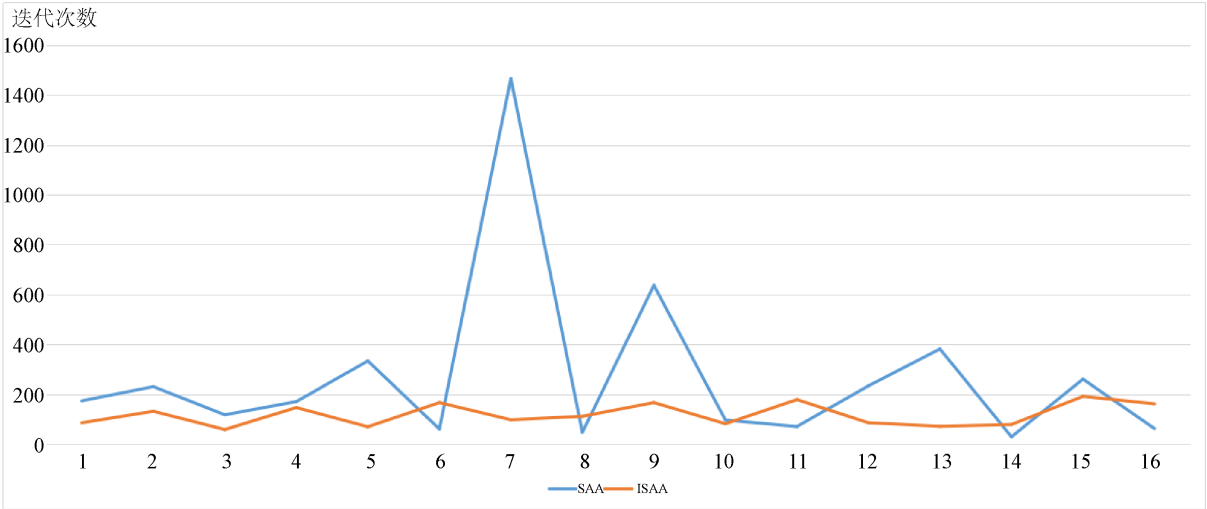

Figure 4. Comparison curve of the second and third sets of experimental results
图 4. 第二、三组实验结果对比曲线

Table 1. Number of defects found in program one (average)
表 1. 程序一中发现的缺陷数量(平均)

实验迭代次数	基本路径覆盖	故障传播路径覆盖
100	1	1
200	3	3
500	4	5
1000	4	6
1500	4	6

Table 2. Number of defects found in program two (average)
表 2. 程序二中的缺陷数量(平均)

实验迭代次数	基本路径覆盖	故障传播路径覆盖
100	4	4
200	4	6
500	4	6
800	5	8
1000	6	8
1500	8	9
1800	8	10
2000	8	10

6. 结论

从上述的实验结果分析中可以看出，由于基于故障传播路径覆盖的软件测试方法引入了故障传播路径的模型，并且在软件测试过程中根据对软件缺陷类型的预测，引入了培育好的种子缺陷，通过改进的退火算法生成测试用例集，激活了基本路径覆盖测试法等其他方法所不容易发现的、与种子缺陷相关联的潜在缺陷。因此，这种方法基本可以达到用最少的测试用例发现更多缺陷的目的，可以更有效地为软件的可靠性提供保障。

基金项目

本项工作得到航天科工联合基金项目——基于模型的实时控制软件需求验证及测试技术(K2370530)的支持。

参考文献

[1] Katerina, G.P. and Trivedi, K.S. (2000) Failure Correlation in Software Reliability Models. *IEEE Transactions on Reliability*, **49**, 37-48. https://doi.org/10.1109/24.855535

[2] Chen, S. and Mills, S. (1996) A Binary Markov Process Model for Random Testing. *IEEE Transactions on Software Engineering*, **22**, 218-223. https://doi.org/10.1109/32.489081

[3] Bishop, P.G. and Pullen, F.D. (1988) PODS Revisited—A Study of Software Failure Behavior. *International Symposium on Fault-Tolerant Computing*, Tokio, 1-8.

[4] Sahinoglu, M. (2003) An Empirical Bayesian Stopping Rule in Testing and Verification of Behavioral Models. *IEEE Transactions on Instrumentation and Measurement*, **52**, 1428-1443. https://doi.org/10.1109/TIM.2003.818548

[5] 景涛, 江昌海, 胡德斌, 等. 软件关联缺陷的一种检测方法[J]. 软件学报, 2005, 16(1): 17-28.

[6] 刘新忠. 关联缺陷及其应用研究[D]: [博士学位论文]. 长春: 吉林大学, 2010.

[7] 宋想, 宋晓秋. 基路径覆盖测试用例自动生成方法研究[J]. 计算机工程与设计, 2013, 34(8): 2759-2763.

[8] 李果, 高建民, 高智勇, 等. 基于小世界网络的复杂系统故障传播模型[J]. 西安交通大学学报, 2007, 41(3): 334-338.

[9] Wang, K., Wang, Y. and Zhang, L. (2015) Software Testing Method Based on Improved Simulated Annealing Algorithm. *International Conference on Reliability*, Guangzhou, 6-8 August 2014. https://doi.org/10.1109/ICRMS.2014.7107215

[10] 北京旋极信息技术有限公司软件测试培训中心. 软件复杂度概述[J]. 今日电子, 2001(12): 38-40.

[11] Lan, W., Zhou, K., Feng, J., *et al*. (2010) Research on Software Cascading Failures. *International Conference on Multimedia Information Networking & Security*, Nanjing, 4-6 November 2010. https://doi.org/10.1109/MINES.2010.214