doi:10.13582/j.cnki.1672-9102.2015.02.016

能提高错误检测能力的回归测试用例集约简

华丽[1],李晓月[2],王成勇[1],谷琼[1,3]

(1.湖北文理学院 数学与计算机科学学院,湖北 襄阳 441053;2.河南机电高等专科学校 计算机科学与技术系,河南 新乡 453002;
3.西南大学 逻辑与智能研究中心,重庆 400715)

摘 要:软件测试过程中,测试用例集的规模可能会随着软件的维护和修改而飞速地增长,使得回归测试费用大幅度增加.为降低回归测试成本,需对回归测试用例集进行约简.现有的测试用例集约简方法不仅缩小了测试用例集的规模,同时也可能削弱了错误检测能力.本文提出了一种新的约简方法,该方法在约简回归测试用例集的时候综合考虑测试用例的测试覆盖度、测试运行代价和错误检测能力3个因素.通过仿真实验表明该方法在有效约简回归测试用例集的同时能保证约简后的测试用例集的错误检测能力.

关键词:测试用例集约简;测试覆盖度;测试运行代价;错误检测能力

中图分类号:TP311 **文献标志码**:A **文章编号**:1672-9102(2015)02-0099-05

Regression test suite reduction on improving fault detection capability

Hua Li[1], Li Xiaoyue[2], Wang Chengyong[1], Gu Qiong[1,3]

(1. School of Mathematics and Computer Sciences, Hu Bei University of Arts and Science, Xiangyang 441053, China;
2. Department of computer science and technology, Henan Mechanical and Electrical Engineering College, Xinxiang 453002, China;
3. Institute of Logic and Intelligence, Southwest University, Chongqing 400715, China)

Abstract: During regression testing, when software is re-tested after some modifications, the size of test suite grow significantly. Thus the costs of regression testing increase dramatically. To reduce the cost of regression testing, regression test suite reduction must be carried out. Traditional methods reduce the faults detection capability either. The new method take into account the three factors of the test coverage, test execution cost and fault detection capability. The simulation results show that the method is effective in test suite reduction and ensuring fault detection capability after test suite reduction.

Keywords: test suite reduction; test coverage; test execution cost; fault detection capability

由于用户需求的变更将会导致软件系统进行更新升级.为了保证更新的模块对现有系统不会造成负面影响,必须对更新后的软件系统进行回归测试.回归测试用例集包括的测试用例主要由2部分组成[1]:原有测试用例集(用来对软件未改动部分进行测试)和新增加用例集(用来对软件更新模块进行测试).随着软件系统的不断更新,回归测试集中包含的测试用例将会越来越多.如果毫无策略地执行现有的所有用例集,回归测试所花费用也会随之大幅度增加[2].因此,对回归测试集进行约简,对于降低回归测试成本是十分必要的.

目前针对测试用例集的约简算法主要有贪心算法、HGS算法、遗传算法、蚁群算法等启发式算法[3-7].这些算法均能在覆盖所有需求的条件下有效地约简测试用例集,达到降低回归测试成本的目的.但是,上述这些方法在约简测试用例集时把每个测试用例的测试覆盖度和测试运行时间作为约简的主要

收稿日期:2014-08-07
基金项目:国家自然科学基金资助项目(71371066);湖北省科技支撑计划公益性科技研究类项目(2012BKB068)
通信作者:华丽(1975-),女,湖北襄阳人,硕士,讲师,主要从事软件工程、软件测试研究.E-mail:17359113@qq.com

参考指标,而没有考虑测试用例检测错误的能力. 因此,约简后的测试用例集极大地减少了测试用例的个数,但是错误检测能力也极大地减弱,这与在测试时要尽可能多地发现错误的目标是不一致的.

与普通测试相比,回归测试时执行的大部分用例都是已执行过的用例. 本文针对回归测试用例集的约简问题提出一种新的方法,该方法除了要考虑每个测试用例的测试覆盖度和测试运行时间外,还将执行过的用例发现错误的能力作为约简测试用例要考虑的重要影响因素. 实验结果表明该方法既能约简测试用例集的个数,有效地降低回归测试的成本,也能保证约简后的测试用例集的错误检测能力,使得测试后的软件质量得到有效的保证.

1 回归测试用例集约简的概念

定义 1 已知回归测试用例集 $T=\{t_1,t_2,\cdots,t_m\}$ 有 m 个测试用例,测试需求集 $R=\{r_1,r_2,\cdots,r_n\}$ 有 n 个测试需求. 测试用例集 T 与测试需求集 R 之间满足的关系用 m 行 n 列矩阵 $\boldsymbol{S}$ 来表示,矩阵元素由下式定义:

$$S(t_i,r_j)=\begin{cases}1, & t_i\in T \text{ 满足需求 } r_j\in R;\\ 0, & t_i \text{ 不满足需求 } r_j.\end{cases}$$

定义 2 对于测试用例 t,t 满足的所有测试需求集记为 $R(t)$.

定义 3 对于测试用例集 T_1,则 T_1 满足的测试需求集记为 $R(T_1)=\cup_{t\in T_1}R(t)$.

定义 4 对于测试需求 r,满足 r 的所有测试用例集记为 $T(r)$.

定义 5 对于测试需求集 R_1,则满足 R_1 的测试用例集记为 $T(R_1)=\cup_{r\in R_1}T(r)$.

定义 6 若测试用例集 $T'\subset T$,且 $R(T')=R(T)$,则称 T' 为测试用例集 T 的约简测试用例集. 若对 $\forall T''$ 为测试用例集 T 的约简测试用例集,其 $|T''|>|T'|$,则称 T' 为测试用例集 T 的最小代表用例集.

为进行回归测试用例集的约简,以下数据是实现的基础[8]:

测试用例库:记录了软件系统在各个阶段使用过的所有测试用例.

测试需求库:记录了每个测试用例与测试需求之间满足的关系.

测试运行代价信息:记录了每个测试用例在测试时的运行代价.

错误检测能力信息:记录了每个测试用例在测试时的错误检测能力.

2 回归测试用例集约简算法

2.1 问题的提出

设测试用例集 T 与测试需求集 R 之间满足的关系如表 1 所示:

表 1 测试用例集 T 与测试需求集 R 满足的关系表

测试用例	测试需求							
	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8
t_1	1	0	1	0	0	1	0	0
t_2	0	1	0	1	1	0	0	1
t_3	0	1	1	0	0	1	0	0
t_4	0	1	1	0	1	0	1	0
t_5	0	1	0	1	1	0	1	0

现使用 HGS 算法[9] 对上述测试用例集进行约简. 首先把测试需求根据 $|T(r_i)|$ 的大小分成 R_1,R_2,...,R_{max} 共 4 个集合即 $R_1\{r_1,r_8\}$,$R_2\{r_4,r_6,r_7\}$,$R_3\{r_3,r_5\}$,$R_4\{r_2\}$. 首先,因为 R_1 集合中所有的测试需求只能被唯一的测试用例满足,所以把测试用例 t_1 和 t_2 加入到最小代表集中,则这 2 个用例满足的测试需求 $r_1,r_2,r_3,r_4,r_5,r_6,r_8$ 标记为满足. 接着选择能满足 R_2 集合中最多测试需求的用例 t_5,则 t_5 满足的测试需求 r_7 标记为满足. 至此,所有测试需求均标记为满足,得到最小代表用例集为 $\{t_1,t_2,t_5\}$.

在上述例子中,使用的 HGS 算法约简测试用例集时,只考虑了每个测试用例的测试覆盖度,而没有考虑每个测试用例的运行代价和错误检测能力. 若测试用例 t_3 能够检测出被零除的严重错误,但却在 HGS 约简算法中当作冗余的测试用例被约简了. 这样导致的直接后果则是约简后测试用例集的错误检测能力也随之减弱.

2.2　本文算法要考虑的因素

为了保证约简后测试用例集的错误检测能力以及测试用例的执行时间尽可能短,在约简测试用例集的时候,除了要考虑每个测试用例的测试覆盖度外,还将考虑每个测试用例的测试运行代价和错误检测能力.

2.2.1　测试用例的测试覆盖度

测试用例的测试覆盖度计算公式定义如下:

$$\mathrm{cov}(t_i) = \sum_{j=1}^{n} \frac{1}{|T(r_j)|}. \tag{1}$$

其中,n 是用例 t_i 满足需求的个数,即 $|R(t_i)|$. $|T(r_j)|$ 表示满足需求 r_j 的测试用例的个数. 如上例中,$\mathrm{cov}(t_1) = 1 + 1/3 + 1/2$.

2.2.2　测试用例的测试运行代价

测试用例的测试运行代价即执行该用例所需要花费的时间[10]. 对于原有用例,测试用例库记录了每个测试用例执行所花的时间,即 $\mathrm{cos}(t_i)$ 为实际执行时所花费时间. 对于新增加的测试用例,其测试运行代价统一按照平均每个测试用例所要花费的时间进行计算,即

$$\mathrm{cos}(t_i) = \text{所有测试用例平均花费时间}. \tag{2}$$

2.2.3　测试用例的错误检测能力

每个测试用例的错误检测能力 $\mathrm{fau}(t_i)$ 由 2 个因素来决定. 其一,该用例在以前执行过程中发现错误的个数;其二,该用例在以前执行过程中发现错误的等级. 根据错误的严重性级别将错误等级分为 4 种:致命错误、严重错误、一般错误和轻微错误[11]. 错误等级定义如下:

致命错误:软件死机、软件崩溃、软件异常退出、数据丢失且很难恢复. 导致以上 4 种后果之一的软件错误视为致命错误.

严重错误:没有实现软件需求,对软件功能有较大影响;没有正确实现软件需求,导致软件不能正常使用;数据丢失但较容易恢复. 导致以上 3 种后果之一的软件错误视为严重错误.

一般错误:没有正确实现软件需求,对软件功能影响较小;没有正确实现软件需求,但对正确使用软件其它功能没有影响或影响较小;软件操作与软件使用说明不符. 导致以上 3 种后果之一的软件错误视为一般错误.

轻微错误:对于已经实现软件需求,对软件功能影响很小或不方便使用的小问题视为轻微错误.

设 fd 为每个错误等级对应的量化值,定义致命错误为 10,严重错误为 8,一般错误为 4,轻微错误为 2. 测试用例的错误检测能力 $\mathrm{fau}(t_i)$ 的计算公式定义如下:

$$\mathrm{fau}(t_i) = \sum_{i=1}^{j} fd_i. \tag{3}$$

其中 j 代表测试用例 t_i 共可以检测 j 个错误,fd_i 代表每个错误等级的值.

2.3　能提高错误检测能力的算法实现

在软件生命周期的整个过程中,测试人员要反复多次执行部分相同的测试用例集进行测试来保证新增加的或修改后的模块没有给整个软件系统带来不好的影响. 测试人员将每次执行测试用例的相关数据(如测试用例执行的时间、测试用例可以检测的错误等级等)记录下来,用于下次回归测试用例集的约简中. 这样约简后的测试用例除了能够有效约简用例的个数以降低回归测试成本外,同时还能保证约简后的测试用例集的错误检测能力,从而保证软件的质量.

2.3.1 测试用例的度量值公式

综合考虑每个测试用例 t 的测试覆盖度 $\mathrm{cov}(t)$、测试运行代价 $\cos(t)$和错误检测能力 $\mathrm{fau}(t)$3 个因素分别在测试过程中所占的重要性比例,现定义测试用例 t 的度量值公式为

$$\mathrm{val}(t) = \mathrm{cov}(t) * 0.4 + 1/\cos(t) * 0.2 + \mathrm{fau}(t) * 0.4. \quad (4)$$

其中,因数 0.4,0.2,0.4 分别代表测试覆盖度、测试运行代价和错误检测能力这 3 个因素的权重因子.

2.3.2 将本文提出的方法应用于 HGS 算法

在原 HGS 算法只考虑每个测试用例的测试覆盖度的基础上,增加考虑每个测试用例的测试运行代价和错误检测能力. 其算法框架描述如下:

步骤 1 根据测试用例集 T 和测试需求集 R 之间满足的关系矩阵 $\boldsymbol{S}$,将测试需求集划分为 $R_1, R_2, \cdots, R_{\max}$;

步骤 2 将满足 R_1 集合中测试需求的测试用例加入到约简后的测试用例集 T' 中,并将这些用例满足的测试需求标记为满足;

步骤 3 根据上式(1)、式(2)、式(3)计算满足 R_2 集合中测试需求的测试用例的测试覆盖度、测试运行代价和错误检测能力,并按式(4)计算每个用例的度量值,再按度量值的大小顺序选择用例至 T' 中,直到 R_2 集合中所有测试需求标记为满足;

步骤 4 重复步骤 3 的方法继续处理 $R_3, R_4, \cdots, R_{\max}$,直至测试需求集 R 中所有需求均标记为满足。

3 仿真实验分析

为了评价本算法的有效性,用 C + +开发了网上书店 web 程序,并使用白盒技术设计测试用例 135 个覆盖测试需求 118 个,且人为地植入 142 个错误,然后使用原 HGS 算法(Ori - HGS)、原蚁群算法(Ori - ACA)[12]及将本文提出的方法应用于 HGS 算法(New - HGS)和蚁群算法(New - ACA)进行比较以下 3 个方面[13-15]:

1)测试用例集约简规模% Size Red = $\dfrac{|T| - |T'|}{|T|} * 100\%$. 其中 $|T|$ 代表原始用例集的个数,$|T'|$ 约简后测试用例集的个数.

2)测试用例集运行代价约简规模% Size Cos = $\dfrac{|C| - |C'|}{|C|} * 100\%$. $|C|$ 代表原始用例集运行代价总和,$|C'|$ 代表约简后用例集的运行代价总和.

3)错误检测能力的丢失率% Fault Loss = $\dfrac{|F| - |F'|}{|F|} * 100\%$. $|F|$ 代表原始用例集能够检测出错误的个数,$|F'|$ 代表约简后用例集能检测出错误的个数.

如下表 2 是使用 Ori - HGS, Ori - ACA, New - HGS, New - ACA4 种算法对% Size red,% Size Cos,% Fault Loss3 个方面的对比. 从实验结果可以看出,使用了本文算法的 New - HGS 和 New - ACA 在丢失一小部分的测试用例集约简规模和运行代价约简规模的情况下,其错误检测能力的丢失率远远小于原 HGS 算法和原 ACA 算法.

表 2 原算法及新算法在 3 个性能方面的对比

算法	Ori - HGS	Ori - ACA	New - HGS	New - ACA
% Size Red	11.68	11.47	10.13	10.21
% Size Cos	10.54	11.23	9.77	9.83
% Fault Loss	7.91	8.05	3.13	2.87

4 结论

原有的一些约简算法在约简测试用例集时只考虑了每个测试用例的测试覆盖度,这样约简后的测试

用例集减弱了其错误检测能力. 本文提出一种方法综合考虑了每个测试用例的测试覆盖度、测试运行代价的错误检测能力,该方法在覆盖所有测试需求的条件下,除了能约简测试用例集的个数外,还能减少约简后测试用例集的测试运行代价,并能有效保证错误检测能力. 最后通过仿真实验结果表明了该算法在保证软件的质量前提下,大大节省了回归测试的成本.

参考文献:

[1] 游亮,卢炎生. 测试用例集启发式约简算法分析与评价[J]. 计算机科学,2011,38(12): 147 – 150.

[2] Harrold M J,Orso A. Retesting software during development and maintenance[C]//Frontiers of Software Maintenance,Beijing,China:IEEE,2008:99 – 108.

[3] Rajvir S,Mamta S. Test case minimization techniques: A review[J]. International Journal of Research & Technology,2013,2(12):1048 – 1056.

[4] Alireza K,Saeed P. Bi – criteria test suite reduction by cluster analysis of execution profiles[C]. International Federation for Information Processing,2012,243 – 256.

[5] 李金蓉. 基于改进的蚁群算法在测试用例集约简问题上的应用研究[M]. 广东:华南理工大学,2013.

[6] 刘艳. 基于程序切片算法的测试用例集约简方法[M]. 安徽:安徽大学,2013.

[7] 胡建军,裴东林. Pohlig – Hellman 算法的改进[J]. 湖南师范大学自然科学学报,2013,36(5):19 – 22.

[8] 马雪英,盛斌奎. 测试用例最小化研究[J]. 计算机应用研究, 2007,24(7):35 – 39.

[9] Harrold M J,Gupta R,Soffa M L. A methodology for controlling the size of a test suite[J]. ACM Transaction on Software Engineering an Methodology,1993,2(3):270 – 285.

[10] 华丽,王成勇,谷琼,程虹. 基于遗传蚁群算法的测试用例约简[J]. 工程数学学报,2012,29(4):486 – 492.

[11] 李华莹,胡兢玉. 回归测试用例优先级排序技术研究[J]. 计算机仿真,2013,30(10):298 – 301.

[12] Hua L, Ding X M. Test suite reduction based on ant colony algorithm with mutation index[C]. ICCSE'2008:74 – 76.

[13] Jeffrey D,Gupta N. Improving fault detection capability by selectively retaining test cases during suite reduction[J]. IEEE Transaction on Software Engineering,2007,33,108 – 123.

[14] 顾聪慧,李征,赵瑞莲. 基于 ACO 的测试用例预优化及参数影响分析[J]. 计算机科学与探索,2014(3):1463 – 1473.

[15] 乔德军,张延军,石磊娜. 基于遗传算法双目立体视觉传感器的优化设计[J]. 湖南师范大学自然科学学报,2014,37(6):53 – 56.