回归测试中测试用例集缩减问题的研究

陈 翔[1,2] 顾 庆[2] 陈道蓄[2] 蒋峥峥[1]

(南通大学计算机科学与技术学院 南通 226019)[1] (南京大学软件新技术国家重点实验室 南京 210093)[2]

摘 要 测试用例集缩减(Test Suite Minimization,TSM)问题作为回归测试的研究热点和难点,在满足对指定测试需求的覆盖前提下,通过识别并移除冗余测试用例来降低回归测试成本。对国内外已有的TSM研究成果进行综述。首先分别从源代码和模型两个角度出发,总结已有的TSM方法:从源代码角度出发,重点分析与总结传统TSM方法和考虑缺陷检测能力的TSM方法;从模型角度出发,重点分析与总结基于扩展有限状态自动机的TSM方法。然后对实证研究中采用的评测程序、评测指标和实证结论进行总结。随后总结了TSM方法在特定测试领域的应用,包括GUI应用测试、Web应用测试和缺陷定位等。最后展望了未来的可能发展趋势。

关键词 回归测试,测试用例集缩减,线性规划,贪心法,元启发式搜索,多目标优化,实证研究

中图法分类号 TP311.5 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2014.09.037

Systematic Review of Test Suite Minimization for Regression Testing

CHEN Xiang[1,2] GU Qing[2] CHEN Dao-xu[2] JIANG Zheng-zheng[1]

(School of Computer Science and Technology, Nantong University, Nantong 226019, China)[1]

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

Abstract Test suite minimization (TSM) is a hot and difficult issue in regression testing search. It aims to identify and remove redundant test cases, and reducing the cost of regression testing. The reduced test suite can satisfy the same test requirements as the original test suite. In this paper, we reviewed the existing research work of TSM. We firstly classified existing TSM approaches into two categories: source code based and model based. In source code based approaches, we analyzed and summarized traditional TSM approaches and fault detection ability concerned TSM approaches. In model based approaches, we mainly analyzed and summarized EFSM based approaches. We secondly summarized empirical subjects, evaluation metrics, and empirical results of previous empirical studies. We thirdly summarized the successful applications of TSM in some specific testing domains, such as GUI application testing, Web Application testing, and fault localization. We finally gave some future work for this hot research topic.

Keywords Regression testing, Test suite minimization, Linear programming, Greedy algorithm, Meta-heuristic search, Multi-objective optimization, Empirical study

1 引言

开发人员在软件的开发和维护过程中,为移除软件内在缺陷、完善已有功能、重构已有代码或提高运行性能等,需要修改软件代码并触发软件演化。随着以统一过程和敏捷方法为代表的增量、迭代式开发过程的流行,软件演化频率迅速提高并亟需经济有效的测试方法来确保演化后软件产品的质量。回归测试作为一种有效方法,可有效保证代码修改的正确性并避免代码修改对被测程序其他模块产生的副作用。执行回归测试时的一个核心问题是测试用例的维护策略设定。在软件持续演化过程中,因新测试用例的不断添加,导致测试用例集规模持续增长并增加了测试用例的执行和维护开销。例如Rothermel等人在某一合作企业内,当测试一个包含约2万行代码的软件产品时,发现运行完所有测试用例后,所需时间高达7周[1]。为提高回归测试效率,国内外研究人员提出多种可降低回归测试开销的有效方法,包括:无效测试用例的识别和修复、测试用例选择、测试用例集缩减、测试用例优先排序和测试用例集扩充等[2]。本文对测试用例集缩减(Test Suite Minimization,TSM)(有的文献将该问题称为Test Suite Reduction)这一重要回归测试方法进行综述,该问题在可以满足指定测试需求的覆盖前提下,通过识别并移除冗余测试用例来缩减回归测试用例集规模,从而降低回归测试的开销。

国内外研究人员对TSM问题进行了深入分析并取得了丰硕的研究成果。为了对该研究问题进行系统分析、总结和

到稿日期:2013-04-08 返修日期:2013-06-30 本文受国家自然科学基金(61202006),江苏省高校自然科学研究项目(12KJB520014),南通市应用研究计划项目(BK2012023,BK2011011),南京大学计算机软件新技术国家重点实验室开放课题(KFKT2012B29)资助。

陈 翔(1980—),男,博士,讲师,主要研究方向为软件测试和程序分析,E-mail:xchencs@ntu.edu.cn;顾 庆(1972—),男,博士,教授,博士生导师,主要研究方向为软件质量保障;陈道蓄(1947—),男,教授,博士生导师,主要研究方向为并行计算和软件质量保障;蒋峥峥(1980—),女,硕士,讲师,主要研究方向为软件工程。

比较，我们首先在 IEEE、ACM、Spring、Wiley 和 Elsevier 等论文数据库中进行检索，选择时采用的关键词包括“Test Suite Minimization”和“Test Suite Reduction”等。然后对检索出的论文，通过查阅论文中的参考文献和相关研究人员发表的论文列表来进一步识别出遗漏论文。最终我们选择出与该主题直接相关的 40 篇论文(截止到 2013 年 2 月)。图 1 按论文发表的时间进行汇总，其中横坐标表示发表年份，纵坐标表示该年份发表的论文累积总数，并按会议和期刊进行了划分。例如从图 1 中可以得知：截止到 2013 年，总共发表会议论文 21 篇，发表期刊论文 19 篇。通过图 1，不难发现 TSM 问题一直是回归测试中的研究热点和难点，近 10 年来每年均会有一定数量论文出版。表 1 按论文出版源进行汇总，并按照论文发表总数降序排列，从汇总结果可以看出，大部分论文都发表在软件工程领域重要的会议和期刊上，例如 ICSE 会议有 5 篇、ICSM 会议有 6 篇、IST 期刊有 7 篇、TSE 期刊有 4 篇。

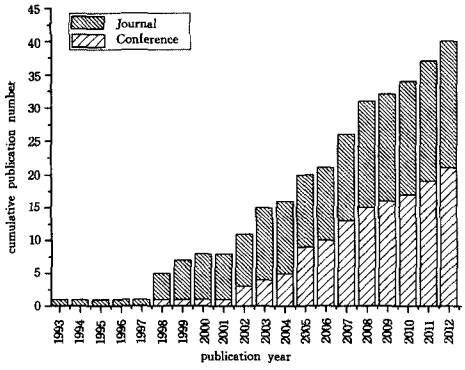

图 1　论文在不同时间的累积发表数量分布

表 1　论文在不同出版源上的分布

出版源简称	类型	发表论文总数
IST	期刊	7
ICSM	会议	6
ICSE	会议	5
TSE	期刊	4
SAC	会议	3
ISSRE	会议	2
ISSTA	会议	2
JSS	期刊	2
PASTE	会议	1
SEKE	会议	1
SOQUA	会议	1
TOSEM	期刊	1
STVR	期刊	1
JSMRP	期刊	1
SPE	期刊	1
计算机学报	期刊	1
软件学报	期刊	1

本文第 2 节介绍回归测试的研究背景和 TSM 问题的研究框架；第 3 节和第 4 节分别从源代码和模型两个角度出发对已有 TSM 方法进行分析和总结；第 5 节依次总结了实证研究中经常采用的评测程序、评测指标和实证结论；第 6 节总结了 TSM 方法在不同特定测试领域中取得的研究成果；最后对 TSM 问题的未来可能研究工作进行了展望。

2　测试用例集缩减问题的研究背景

2.1　回归测试

软件产品持续集成和交付时，回归测试是保障演化后软件产品质量的一种有效方法。但统计数据表明，回归测试所需开销一般占整个软件测试预算的 80%以上，占整个软件维护预算的 50%以上[3,4]。一种简单维护策略是重新执行已有所有测试用例，但该策略确存在诸多不足：(1)一些测试场景中，若测试用例数较多或单个测试用例执行开销较大时，执行所有测试用例所需的时间或成本开销将超过实际预算；(2)部分代码修改会影响到被测程序内在语义或外部接口，导致部分测试用例失效；(3)部分代码修改会生成新的测试需求，需额外设计新的测试用例。

针对上述问题，研究人员提出了多种有效回归测试技术。在 Harrold 等人对该领域的总结基础上[2]，本文给出图 2 所示的回归测试研究框架。具体来说，首先对修改前后程序进行建模和比对，识别出代码修改模块，通过修改影响分析识别出与代码修改存在依赖关系的相关测试需求。随后在已有测试用例集的基础上依次执行如下的测试用例维护技术：(1)无效测试用例的识别和修复技术。部分代码修改会造成相关模块的外部接口或内在语义发生变更，从而造成部分测试用例无效。若直接移除这类测试用例可能会降低原有测试用例集的质量，目前一种可行方法是对这类测试用例进行修复。(2)测试用例选择技术。通过分析代码修改，该技术可以从已有测试用例中选择出所有可检测代码修改的测试用例，并确保未被选择的测试用例在修改前后程序上的执行行为保持一致。(3)测试用例集扩充技术。在代码修改影响分析基础上，对已有测试用例集的充分性进行评估，若不充分则设计新的测试用例以确保对代码修改的充分测试。(4)测试用例集缩减技术。本文将对该研究问题进行系统综述。(5)测试用例优先排序技术。当测试预算不足以执行完所有测试用例时，可以基于特定排序准则，对测试用例进行排序以优化其执行次序，旨在最大化排序目标，例如最大化测试用例集的早期缺陷检测速率。

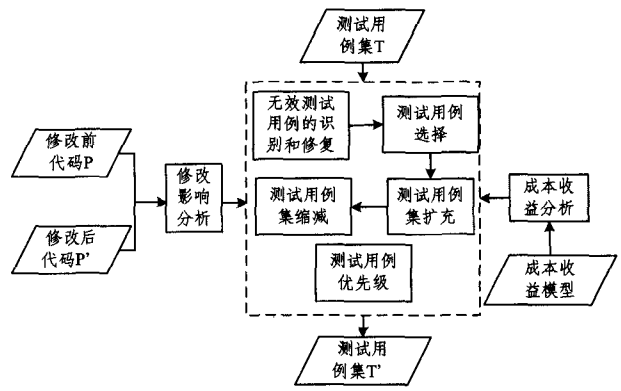

图 2　回归测试研究框架

将学术界取得的已有研究成果成功应用到软件企业的关键是进行成本收益分析，精确的成本收益分析离不开合理的成本收益模型。目前主要从成本和收益两个角度出发，来识别出模型内部的影响因素。这些因素包括：代码修改分析、修改影响分析、测试用例运行环境准备、测试用例执行开销、测试用例结果检查、测试用例选择、压缩、优先级等方法和测试用例缺陷检测能力降低造成的损失等[5-7]。

2.2　测试用例集缩减问题研究框架

软件持续演化将造成测试用例集规模的持续增加。在满足指定测试需求覆盖前提下，会出现冗余测试用例。本文综

述的 TSM 方法，通过识别并移除这类测试用例来降低回归测试成本。通过对已有研究工作的深入分析和总结，本文给出图 3 所示的研究框架。具体来说，首先对被测程序和相关制品(Artifact)进行控制流或数据流分析，生成原有测试用例集 T 覆盖的测试需求集 R，通过代码插桩(Instrument)并执行 T 中所有测试用例，可以搜集到每个测试用例的测试需求覆盖信息，随后借助 TSM 方法，使得缩减后的测试用例集 T' 覆盖的测试需求与 T 保持一致。最后通过评测指标来对 TSM 方法的成本和收益进行评估。

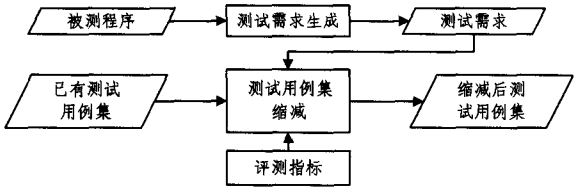

图 3　测试用例集缩减问题研究框架

3　基于源代码的测试用例集缩减方法

3.1　传统测试用例集缩减问题研究

3.1.1　传统测试用例集缩减问题描述

传统的 TSM 问题仅从测试需求覆盖出发，Harrold 等人首次对传统 TSM 问题给出如下一般性描述[8]。

给定：原有测试用例集 $T=\{t_1,t_2,\cdots,t_m\}$，T 覆盖的测试需求集 $R=\{r_1,r_2,\cdots,r_n\}$，T 的子集 $T_1,T_2,\cdots,T_n$，其中 T_i 含有所有覆盖测试需求 r_i 的测试用例。

问题：寻找 T 的子集 $T'(T'\subseteq T)$，使得 $\forall r\in R(\exists t\in T\wedge t$ 覆盖 $r\Rightarrow\exists t'\in T'\wedge t'$ 覆盖 $r)$。

在传统 TSM 问题描述中，需要覆盖的测试需求与测试人员选择的测试充分性准则[9]相关。测试充分性准则的设定一般建立在对被测程序的控制流或数据流分析的基础之上。若以语句覆盖准则为例，则需要覆盖的测试需求 R 应包含已有测试用例集 T 可覆盖的所有可执行语句集。其中最优 TSM 问题是寻找 T 的最小子集 T'，通过将最优 TSM 问题多项式时间内规约为最小集合覆盖问题，可证实该问题是一个 NP-Complete 问题[8]。

3.1.2　传统测试用例集缩减方法研究

本文将传统 TSM 方法划分为 3 类：线性规划法、贪心法和元启发式搜索法。本节将依次对每一类方法中的经典研究工作进行分析和总结。

TSM 问题作为一个典型的组合优化问题，可以采用整数线性规划法进行求解[10-12]。该线性规划模型在满足对已有测试需求 R 的覆盖前提下，将最小化选择出的测试用例数作为优化目标。其模型可定义如下：

$$\text{Min } x_1+x_2+\cdots+x_m$$

$$\text{s. t.}$$

$$a_{11}x_1+a_{12}x_2+\cdots+a_{1m}x_m\geqslant 1$$

$$a_{21}x_1+a_{22}x_2+\cdots+a_{2m}x_m\geqslant 1$$

$$\cdots\cdots$$

$$a_{n1}x_1+a_{n2}x_2+\cdots+a_{nm}x_m\geqslant 1$$

其中变量 x_i 取值为 1，表示选择第 i 个测试用例，否则取值为 0。系数 a_{ij} 取值为 1，表示第 j 个测试用例覆盖第 i 个测试需求，否则取值为 0。

整数线性规划法虽然在理论上可以算出最优解，但在实际测试过程中存在时间复杂度高的问题，其运算开销随测试需求数呈指数级增长，所以难以适用于大规模软件的测试。因此研究人员提出了多种贪心算法。

一种简单可行的方法(本文统称为 G 方法)[12]是依次从 T 中选出一个对 R 中测试需求覆盖能力最强的测试用例 t，然后从 R 中移除 t 覆盖的测试需求，直到 R 中所有测试需求均被覆盖到。在每次选择时，若存在多个需求覆盖能力相同的测试用例，则每次从中随机选择一个。该方法的最坏时间复杂度为 $O(m\times n\times\min(m,n))$。

随后研究人员通过分析测试用例与测试需求的覆盖信息，识别出与 TSM 问题密切相关的两类测试用例：基本测试用例和 1-to-1 冗余测试用例，并分别定义如下：

定义 1(基本测试用例)　若存在测试需求 r_i，且仅有一个测试用例可以覆盖 r_i，则称该测试用例为基本测试用例。

因为某些测试需求仅被基本测试用例覆盖到，所以在执行测试用例集缩减时应优先选择这类测试用例。

定义 2(1-to-1 冗余测试用例)　若测试用例 i 覆盖的测试需求是测试用例 j 覆盖的测试需求的子集，则称测试用例 i 相对于测试用例 j 为冗余测试用例。

根据定义 2 可知，若测试用例 i 被选入缩减后的测试用例集中，则可以用测试用例 j 进行代替且不会影响原有测试需求的覆盖能力。所以在执行测试用例集缩减时，应尽量移除这类冗余测试用例。

Harrold 等人提出一种启发式算法 HGS 法[8]，该方法的核心思想是尽早选出更多的基本测试用例。具体来说：首先将测试需求 R 进行分类$\{R_1,R_2,\cdots,R_k\}$，其中 R_i 包含所有仅被 i 个测试用例覆盖的需求集，不难看出，所有覆盖 R_1 需求的测试用例均是基本测试用例。随后从 T 中迭代选择基本测试用例，直到所有测试需求均被覆盖到。迭代过程可参考文献[8]。HGS 法的最坏时间复杂度为 $O((m+n)\times n\times k)$，其中 $k=\max_i\{T_i\}$。

Chen 和 Lau 提出另一种启发式算法 GRE 法[13,14]。GRE 法不断应用如下 3 种策略，直到所有需求均被覆盖到。(1)选择所有基本测试用例；(2)移除所有 1-to-1 冗余测试用例；(3)选出对剩余需求覆盖能力最强的测试用例。其中在应用策略 3 时，需要保证策略 1 和 2 均成功应用结束。GRE 法最坏时间复杂度为 $O((n+m^2k)\times\min(m,n))$。

Jone 和 Harrold 在执行测试用例集缩减时考虑的是 MC/DC (Modified Condition/Decision Coverage) 覆盖准则[15]。MC/DC 覆盖准则在测试充分性上要强于判定或条件覆盖准则。其在满足判定/条件覆盖准则基础上，还需额外证实判定语句中的每个条件可独立影响判定取值。他们分别提出两种策略：Break-down 和 Build-up，并对这两种策略的执行时间和测试用例集规模缩减率进行了比较。实证结果表明，Break-down 策略虽然在执行时间上要高于 Build-up 策略，但却可以获得更好的测试用例集规模缩减率。

形式化概念分析(Formal Concept Analysis)是一种用于分析对象和相关属性的分层聚类方法，Tallam 和 Gupta 将形式化概念分析应用到 TSM 问题[16]，他们将测试用例视为对象，测试需求视为属性，通过分析测试用例和测试需求的覆盖信息构建出概念格(Concept Lattice)。随后在概念格上执行

两类移除操作：(1)若测试用例 i 覆盖的需求是测试用例 j 覆盖的需求的超集，则移除测试用例 j；(2)若可覆盖需求 i 的测试用例是可覆盖需求 j 的测试用例的子集，则移除需求 i。在移除完部分测试用例和需求后，他们提出一种贪心算法 Delayed-Greedy。实证结果表明，该方法在缩减后的测试用例集规模上等于或小于其他贪心法(例如 G 方法和 HGS 方法)。

元启发式搜索技术提供了一个高层框架，基于特定的启发式技术(典型技术包括爬山法、模拟退火和遗传算法等)，可以采用合理的计算开销去求解组合优化问题。Mansour 和 El-Fakih 提出一种混合遗传算法来求解 TSM 问题[17]。遗传算法是一种全局搜索算法，它主要受到生物界进化规律(即适者生存和优胜劣汰机制)的启发，针对具体问题，遗传算法需要对候选解进行染色体编码和适应值函数设定。针对 TSM 问题，他们用长度为 m 的二进制串来编码染色体，取值为 1 表示选择相应测试用例，否则取值为 0。基于上述染色体编码，他们随后定义了一系列染色体演化操作：变异操作和交叉操作。在演化过程中存在两类问题：首先是在演化过程中会生成一部分无效解，对这类无效解，可通过随机添加测试用例，使之变成有效解；其次一些有效解可能包含的测试用例数较多，可采用爬山法来进一步减少测试用例数。通过应用这两种策略可以提高遗传算法的有效性。当候选最优解在 20 轮迭代后仍保持不变，则种群演化结束并返回当前最优解。

最后本节从方法类型、发表文献、测试需求、实现评测程序的编程语言和采用的评测程序这几个角度对传统的 TSM 方法进行总结，评测程序规模的判断依据如下：将代码行数超过 1 万行的程序设定为大规模程序，将代码行数介于 1 千行到 1 万行之间的程序设为中规模程序，而将代码行数小于 1 千行的程序设为小规模程序。最终总结结果如表 2 所列。

表 2　传统测试用例集缩减方法总结

方法类型	发表文献	测试需求	编程语言	评测程序规模和个数
线性规划法	Black et al.[10]	Def-use 覆盖	C	7 个小规模程序
	Zhong et al.[11]	语句覆盖	C	7 个小规模程序、2 个中规模程序和 2 个大规模程序
	Zhang et al.[12]	方法覆盖和语句覆盖	Java	1 个中规模程序和 3 个大规模程序
贪心法(G 方法)	Zhang et al.[12]	方法覆盖和语句覆盖	Java	1 个中规模程序和 3 个大规模程序
贪心法(HGS 法)	Harrold et al.[8]	Def-use 覆盖	C	11 个小规模程序
	Zhong et al.[11]	语句覆盖	C	7 个小规模程序、2 个中规模程序和 2 个大规模程序
	Zhang et al.[12]	方法覆盖和语句覆盖	Java	1 个中规模程序和 3 个大规模程序
贪心法(GRE 法)	Chen et al.[13,14]	分支覆盖	C	1 个简单程序
	Zhong et al.[11]	语句覆盖	C	7 个小规模程序、2 个中规模程序和 2 个大规模程序
	Zhang et al.[12]	方法覆盖和语句覆盖	Java	1 个中规模程序和 3 个大规模程序
贪心法(其他)	Tallam et al.[16]	分支覆盖和定义使用对	C	6 个小规模程序和 1 个中规模程序
	Jone et al.[15]	MC/DC 覆盖	C	1 个小规模程序和 1 个中规模程序
元启发式搜索技术(遗传法)	Zhong et al.[11]	语句覆盖	C	7 个小规模程序、2 个中规模程序和 2 个大规模程序
	Mansour et al.[17]	语句覆盖	C	7 个小规模程序和 4 个中规模程序

3.2　基于多目标优化的测试用例集缩减问题研究

传统 TSM 方法仅考虑在满足指定测试需求的覆盖前提下，最小化缩减后的测试用例集规模。实证研究表明：这类方法识别出的部分冗余测试用例可检测出被测程序的内在缺陷，这类测试用例的移除会降低原有测试用例集的缺陷检测能力。该问题的一种有效解决方法是采用多目标优化技术。

Jeffrey 和 Gupta 发现不同覆盖准则因关注的程序实体不一样，会导致它们之间的测试充分性并不一致。因此他们引入选择性冗余概念[18,19]，即若测试用例在考虑第一个测试充分性准则 C_1 时，被标记为冗余测试用例，则进一步考虑第二个测试充分性准则 C_2，若不是冗余测试用例，则保留该测试用例并称该测试用例相对于 C_1 来说属于选择性冗余。在实证研究中，他们将 C_1 设置为判定覆盖准则，C_2 设置为 all-uses 覆盖准则，并基于 HGS 法[8]实现了他们所提的 TSM 方法，结果表明通过保留少量选择性冗余测试用例，可以大幅度提高缩减后的测试用例集的缺陷检测能力。

Black 等人在执行测试用例集缩减时，同时考虑了两种优化目标[10]：(1)在满足指定测试需求前提下，最小化缩减后测试用例集的规模；(2)最大化历史缺陷检测能力。他们采用加权求和方式将这两种优化目标进行拟合。在传统线性规划法基础上，保持原有约束条件不变，并将优化目标重新描述如下：

$$\text{Min } \alpha(x_1+x_2+\cdots+x_m)+(1-\alpha)(e_1x_1+e_2x_2+\cdots+e_mx_m)$$

其中，e_i 取值为 1，表示测试用例 t_i 在过去曾经检测过缺陷，否则取值为 0。该优化目标采用权重因子 $\alpha(0\leqslant\alpha\leqslant1)$ 来平衡两种不同优化目标间的相对重要程度。若 α 取值为 0.5，则表示这两个优化目标的重要性保持一致。

Hsu 和 Orso 对 Black 等人的研究工作[10]进行拓展并开发出 MINTS 工具，其拓展点包括借助最新的整数线性规划求解器进行求解，在测试用例集缩减时可以支持任意多个优化目标[20]。他们采用 3 种方式将多个不同优化目标进行拟合。这 3 种方式分别是：加权求和、优先排序和混合法。其中方式 1 与 Black 等人采用的拟合方式相同，方式 2 需要专家预先将优化目标按重要程度从高到低进行排序，并且每次仅考虑一个优化目标。具体来说，首先针对第一个优化目标进行优化，其结果在考虑下一个优化目标时构成约束条件并进行优化，依此类推，直到所有优化目标均被考虑到。方式 3 将优化目标进行分组，同一组内的优化目标采用优先排序方式，而不同组则采用加权求和方式。

Yoo 和 Harman 在实际测试过程中，发现多个优化目标之间可能存在竞争或冲突问题，他们将帕累托效率(Pareto Efficiency)应用到 TSM 问题上[21,22]。在实证研究中，他们考虑的优化目标与 Hsu 和 Orso 保持一致。最终该方法可获得一组非主导且等价最优的测试用例子集。

最后本节从发表文献、考虑的优化目标、优化目标的拟合方式、实现评测程序的编程语言和评估程序的规模对基于多目标优化的 TSM 方法进行了系统总结，最终结果如表 3 所列。

表3 基于多目标优化的测试用例集缩减方法总结

参考文献	优化目标	拟合方式	编程语言	评测程序规模和个数
Black et al.[10]	(1)最小化测试用例集规模 (2)最大化历史缺陷检测能力	加权求和	C	7个小规模程序
Jeffrey et al.[18,19]	(1)判定覆盖准则 (2)all-uses数据流覆盖准则	选择性冗余	C	7个小规模程序和 1个大规模程序
Yoo et al.[21,22]	(1)最小化测试用例集规模 (2)最大化测试用例集执行时间 (3)最大化历史缺陷检测能力	帕累托效率	C	4个小规模程序、 2个中等规模程序和 3个大规模程序
Hsu et al.[20]	(1)最小化测试用例集规模 (2)最大化测试用例集执行时间 (3)最大化历史缺陷检测能力	(1)加权求和 (2)优先排序 (3)混合法	C	7个小规模程序和 1个大规模程序

3.3 其他测试用例集缩减方法研究

Harder等人基于操作抽象(Operational Abstraction)进行测试用例集缩减[23]。操作抽象是一种对程序行为的形式化数学描述。他们借助动态不变式检测器Daikon[1)]来获取操作抽象。在执行测试用例集缩减时,若移除测试用例t不会对已经检测到的程序不变式产生影响,则称测试用例t是冗余测试用例。

传统的TSM方法需要借助代码插桩来搜集测试用例的测试需求覆盖信息。代码插桩需要获取源代码,并且会增加程序的执行时间。所以这类方法难以适用于基于组件的程序或网络应用程序的测试,因为前者无法获取组件源代码,而后者对程序的运行性能要求较高。McMaster和Memon提出一种基于调用栈(Call Stack)覆盖准则的测试用例集缩减方法[24]。调用栈在基于栈的执行环境中可表示当前活跃方法的调用有序序列。该方法包括两个阶段:调用栈的信息收集和测试用例集缩减。实证研究表明:这类方法通过增加少量的运行时间开销,可以大幅度缩减测试用例集的规模,并且只会少量降低其缺陷检测能力。

郝丹等人提出一种按需测试用例集缩减方法[25]。他们在测试用例集缺陷检测能力缩减率不超过$l\%$的前提下,采用整数线性规划法进行求解。其步骤包括:(1)在不同置信水平上,在单个语句级别上搜集缺陷检测能力损失数据,并构造缺陷检测损失表;(2)基于上述统计数据,将按需测试用例集缩减问题进行建模,他们分别基于局部约束和全局约束提出两种整数线性规划模型;(3)对上述模型进行求解。

除此之外,也有研究人员从分析测试需求间的相互关系入手,通过优化测试需求来进行测试用例集缩减。Marre和Bertolino发现部分测试需求间存在包含关系[26],并将其定义如下:

定义3(测试需求的包含关系) 若测试需求i包含测试需求j,则覆盖测试需求i的测试用例一定覆盖测试需求j。

在给定需要覆盖的测试需求后,通过分析测试需求间的包含关系,可以获得生成集(Spanning Set)。他们采用遗传算法来求解生成集,并基于生成集对测试用例集进行缩减。

章晓芳等人在进行规约测试时,首先对测试需求间的相互关系进行分析并建模为需求约简模型[27],随后基于该模型,对测试需求进行优化并完成对测试用例集的缩减。随后陈振宇等人首先对测试需求和测试用例进行分析并建模为需求分析图,然后借助一种图收缩方法对测试需求进行优化并完成测试用例集的缩减[28]。

顾庆和陈翔等人基于部分测试需求覆盖进行测试用例集缩减[29,30]。他们首先根据代码修改信息,将测试需求集划分为关注需求集和无关需求集,在执行测试用例集缩减时在覆盖关注需求集的同时,要求避免覆盖无关需求集。最终结果表明该方法可以选出更少的测试用例,同时可以维持一定的缺陷检测率[29]。随后他们通过代码修改分析,将测试需求按照重要程度划分为3类,并提出两种基于重要测试需求覆盖的测试用例集缩减方法[30]。陈翔和顾庆等人基于传统HGS法识别出冗余测试用例,并从中选出一部分来满足需求间的组合覆盖,实证结果表明通过添加少量测试用例,可以有效提高缩减后测试用例集的缺陷检测能力[31]。

4 基于模型的测试用例集缩减方法

目前学术界和工业界逐渐借助模型(例如有限状态自动机、状态图和UML模型等)来辅助软件的开发和维护。其中一种典型模型是扩展有限状态自动机(Extended Finite State Machine,EFSM),其对应的测试需求是状态、变迁或变迁序列。基于EFSM的测试用例一般描述为一个变迁序列。

Vaysburg和Korel等人提出一种基于依赖性分析的测试用例集缩减方法[32,33]。首先对EFSM进行控制或数据依赖分析,随后构造出静态和动态依赖图,以提取静态或动态交互模式。在指定待测变迁时,若两个测试用例对于该变迁存在相同的交互模型,则认为这两个测试用例等价。最后保留等价测试用例集中的任意一条测试用例并删除其他冗余测试用例。Chen等人同样基于EFSM提出一种测试用例集缩减方法[34]。给定EFSM和一系列基本代码修改操作,通过对不同类型的代码修改操作(包括添加、修改和删除操作)进行分析来识别出交互模式,这些交互模式可以捕获出相应代码修改操作对模型的影响和副作用。最后他们基于测试用例对交互模式的覆盖相应提出测试用例集缩减方法。

5 实证研究

5.1 评测程序

我们将已有实证研究中采用的来自实验室或开源软件界的评测程序进行了搜集和汇总,最终结果如表4所列,其依次罗列了程序名称、编程语言、代码规模、程序功能描述、首次使用时间和累计使用次数,其中罗列的大部分评测程序都可以从内布拉斯加大学林肯分校的SIR库中下载[35]。在表4中,我们将评测程序按照累计使用次数从高到低进行排序。从排序结果可以看出:最常用的面向结构程序是以C语言编程实

1) http://groups.csail.mit.edu/pag/daikon/

现的西门子套件(Siemens Suite)和 space 程序。接下来对这些重要的评测程序依次进行介绍。

表 4 评测程序总结

程序名	编程语言	代码规模	功能描述	首次使用时间	累计使用次数
Siemens 套件	C	小	西门子套件,包含 7 个小规模程序	1998	16
Space	C	中	针对数组语言的一个解释器	1999	12
Flex	C	大	词法分析程序生成器	2009	2
Jtopas	Java	中	解析文本数据的辅助程序	2011	2
Xml-security	Java	大	实现 XML 安全标准的程序	2011	2
Xmlppm	C	中	XML 文件压缩程序	2008	1
Tcc	C	大	小型 C 编译器	2008	1
Tar	C	大	Unix 系统内的压缩工具	2008	1
Pdftohtml	C	中	将 PDF 文件转换为 HTML 或 XML 格式的程序	2008	1
Grep	C	大	模式识别程序	2010	1
Gzip	C	中	数据压缩/解压缩程序	2010	1
Sed	C	大	流编辑器	2010	1
Jmeter	Java	大	一个完成负载测试的 Java 桌面应用程序	2011	1
Ant	Java	大	类似于 Make 的项目构建工具	2011	1
Nanoxml	Java	中	一个针对 Java 的小型 XML 解析器	2011	1

西门子套件包括 7 个小规模 C 语言程序。其中 tcas 是防止航空器空中相撞系统,schedule 和 schedule2 是优先级调度器,tot_info 针对指定的输入数据生成统计信息,print_tokens 和 print_tokens2 是词法分析器,replace 程序完成模式匹配和替换。这些程序早期被西门子研究院用于研究控制流和数据流覆盖准则的缺陷检测能力[36]。西门子套件中的每个程序均配有大量测试用例,测试用例的设计过程包括两个阶段:首先基于黑盒测试中的等价类划分技术,然后采用白盒测试技术,确保每个可执行的语句、分支和定义使用对至少被 30 个测试用例覆盖到。研究人员也为每个基准程序创建了大量缺陷版本,这些缺陷版本一般通过修改一行代码生成,少量通过修改 2-5 行代码生成。注入的缺陷来源于实际的编程经历。在生成的大量缺陷版本中,被保留下的缺陷版本至少被 3 个、至多被 350 个测试用例覆盖到。

Space 程序是为欧洲航天局开发的,针对数组定义语言(ADL)的一个解释器。该程序包含 35 个版本,每个仅包含单一缺陷,其中 30 个缺陷版本是在开发过程中发现的。测试用例集的构建过程也包括两个阶段:第一阶段随机生成 10000 个测试用例;第二个阶段通过添加测试用例,使得程序对应控制流图中每条可执行边至少被其中 30 个测试用例覆盖到。最终生成的测试用例集包括 13585 个测试用例。

随着 Java 编程语言和 JUnit 测试框架的日益成熟,研究人员在实证研究中逐渐采用基于 Java 编程语言实现的评测程序来提高研究工作的说服力,常用的程序包括 Jtopa、Xml-security、JMeter、Ant 和 NanoXML。这些程序均配备了大量基于 JUnit 测试框架的测试用例,根据粒度大小,测试用例可以分为两类:基于方法级别的测试用例和基于类级别的测试用例。

5.2 评测指标

目前对 TSM 方法有效性的评测指标主要分两个方面:有效性和执行效率。其中针对执行效率的评测指标主要是 TSM 方法的具体执行时间。而针对有效性的两个常用评测指标是测试用例集规模缩减率和缺陷检测能力缩减率。这两个指标常用于 TSM 技术的成本收益分析,其中测试用例集规模缩减率对应于成本,而缺陷检测能力缩减率对应于收益。

假设缩减前的测试用例集为 T,缩减后的测试用例集为 T',测试用例集 T 可以检测出的缺陷数为 n,测试用例集 T' 可以检测出的缺陷数为 n'。则测试用例集规模缩减率为:

$$(1-|T|/|T'|)\times 100\% \quad (1)$$

测试用例集缺陷检测能力缩减率为:

$$(1-n'/n)\times 100\% \quad (2)$$

上述两个评测指标并未考虑测试用例的需求覆盖信息和缺陷检测信息。McMaster 和 Memon 提出一种新的评测指标[37],即缩减后测试用例集检测每个缺陷的平均预期概率,其计算过程如下:

给定测试需求 i 和缺陷 k,则定义缺陷相关性(Fault Correlation)的计算公式为:

$$faultCorr(i,k)=|covReq(i)\cap detFault(k)|/|covReq(i)| \quad (3)$$

其中,$covReq(i)$返回可覆盖测试需求 i 的所有测试用例,而 $detFault(k)$返回可检测缺陷 k 的所有测试用例。

给定缺陷 k,则可以通过式(3)计算出检测该缺陷的预期概率:

$$expProbFindFault(k)=\max_{i\in R}(faultCorr(i,k)) \quad (4)$$

基于式(4),最终可以算出检测每个缺陷的平均预期概率为(其中 K 是所有可能缺陷集):

$$\sum_{k\in K} expProbFindFault(k)/|K| \quad (5)$$

实证研究表明,该评测指标有助于在执行测试用例集缩减时选择出合适的覆盖准则。

5.3 实证研究结论总结

5.3.1 不同 TSM 方法和覆盖准则间的比较

Chen 和 Lau 采用模拟方式[38]将 GRE 法与 HGS 法进行比较,结果表明没有一种方法总是最优;并且由于采用模拟数据进行比较,所得结论很难适用于实际软件测试。

钟浩等人[11]在实证研究中选择了 4 种经典 TSM 方法(即 HGS 法、GRE 法、遗传算法和整数线性规划法),在选择评测程序时,除了小规模的西门子套件,他们还额外考虑了 2 个中规模程序(XMLPPM 和 PdfToHtml)和 2 个大规模程序(TCC 和 tar)。他们首先分析了缩减时间与测试用例集复杂度间的关系,通过测试用例集包含的测试用例数和测试用例最多可覆盖的需求数来计算其复杂度。总体来说,缩减时间与测试用例集的复杂度呈正相关关系。随后在比较不同方法的执行时间时,发现遗传算法耗时最长,HGS 法最短,线性规划法和 GRE 法居中。最后他们分析了不同方法的测试用例集规模缩减率,其中遗传算法的效果最差,其他方法效果相当,特别是整数线性规划法总是可以获得最好效果。

钟浩等人采用 C 编程语言实现的评测程序对不同的 TSM 方法进行了系统比较。随着 Java 编程语言和 JUnit 单元测试框架的成熟,张路等人[12]也选择了 4 种典型 TSM 方法(包括 G 方法、HGS 法、GRE 法和整数线性规划法),在选择评测程序时,采用了 1 个中规模程序(jtopas)和 3 个大规模程序(xmlsec、jmeter 和 ant)。结果表明,上述 4 种 TSM 方法均可以有效缩减测试用例集规模,同时不会大幅度降低原有测试用例集的缺陷检测能力。除此之外,他们发现不同方法

对 TSM 的成本收益影响不大，但测试用例粒度（类级别或方法级别）和充分性覆盖准则（语句覆盖或方法覆盖）的选择对成本收益影响较大。

也有研究人员通过实证研究对不同覆盖准则进行比较，McMaster 和 Memon[37]采用了 3 个 GUI 应用作为评测程序，采用了 5 种覆盖准则：事件覆盖、事件交互覆盖、语句覆盖、函数覆盖和调用栈覆盖，采用了 HGS 作为测试用例集缩减方法。他们采用的评测指标包括：测试用例集规模缩减率 M_1、测试用例集缺陷检测能力缩减率 M_2 和检测每个缺陷的平均预期概率 M_3。结果表明：基于调用栈覆盖准则的 TSM 方法在缩减后的测试用例集上取得最高的 M_3 取值，基于事件覆盖的 M_3 取值最小，其他准则居中。该结论与覆盖准则在其他评测指标上的结论保持一致，从而验证了该准则可以作为覆盖准则优劣的一个有效评测指标。

5.3.2 *传统 TSM 方法对原有测试用例集缺陷检测能力的影响*

传统 TSM 方法通过识别并移除冗余测试用例来降低回归测试开销，但该方法可能会移除掉一部分具有缺陷检测能力的测试用例。Wong 等人首先对该问题进行了研究[39]。他们采用的评测程序为 10 个小规模 UNIX 程序，针对每个程序，采用随机测试方法来构建测试用例池，随后基于测试用例池构造出多组具有不同语句块覆盖率的测试用例集。他们采用手工方式植入缺陷，这些缺陷分为 4 组，其中 Quartile1 组内的缺陷可以被测试用例池中 0～25%的测试用例覆盖到，Quartile2－4 组内的缺陷则可以分别被 25%～50%、50%～75%和 75%～100%的测试用例覆盖到。实证研究结果表明：(1)若测试用例集的语句块覆盖率较高，则应用 TSM 方法后，测试用例集规模缩减率较高。例如若测试用例集的语句块覆盖率介于 90%到 95%之间，则测试用例集规模缩减率平均为 44.23%。(2)应用 TSM 方法后，缺陷检测能力缩减率很小。例如若测试用例集的语句块覆盖率介于 90%到 95%之间，则缺陷检测能力缩减率平均为 1.45%。(3)应用 TSM 技术后，Quartile1 和 Quartile2 组内的缺陷检测能力缩减率分别为 0.39%和 0.66%，而 Quartile3 和 Quartile4 组内的缺陷检测能力缩减率仅为 0.098%和 0。

随后他们进一步选择了中等规模程序 space 进行验证[40]。与上述评测程序不同，他们基于 space 程序的操作版本来设计测试用例池，在构造测试用例集时采用两种方式：(1)随机选择指定数量的测试用例；(2)不断选择测试用例，直至达到指定的语句块覆盖率。同时 space 程序的缺陷均来自软件的实际开发过程中。实证研究结果表明：(1)若测试用例集的语句覆盖率较低，则应用 TSM 方法后，测试用例集规模缩减率为 0。(2)采用两种不同方式构造出的测试用例集，应用 TSM 方法后，测试用例集规模缩减率仅为 7.28%。

在张路等人的实证研究中[12]，他们选择了 4 个 Java 编程实现的评测程序。最终得到的实证结论与 Wong 等人的实证结论保持一致。

但与 Wong 和张路等人的实证研究结论截然不同，Rothermel 等人[41,42]选择 HGS 法作为测试用例集缩减方法，选择西门子套件和 space 程序作为评测程序，实证研究结果表明：(1)应用 TSM 方法，测试用例集规模缩减率较大，平均缩减率为 48%。(2)应用 TSM 方法后，缺陷检测能力缩减率较大，平均缩减率为 48%，其中超过一半以上的测试用例集，其缺陷检测能力缩减率大于 50%，有的甚至达到 100%。

通过对实证研究过程的比较和分析，Rothermel 等人认为实证研究的结论受到多种因素的影响并得到截然不同的两种结论。这些影响因素包括被测程序特征（例如程序规模和内在结构）、测试用例集特征（例如规模和测试充分性）、测试用例特征（例如缺陷检测能力）和缺陷特征（例如是否容易被测试用例检测到）。

6 在特定测试领域内的应用

目前 TSM 方法已经成功从传统软件测试扩展到不同特定测试领域内，这些领域包括图形用户界面测试、Web 应用测试和缺陷定位等。

事件驱动型应用是目前广泛应用的一类软件，它们以事件序列作为输入，随后经历内部状态变更，并最终输出事件序列。常见的事件驱动型应用包括图形用户界面、Web 应用和网络协议等。McMaster 和 Memon[43,44]将调用栈覆盖准则应用到图形用户界面的测试中，并通过实证研究验证了该覆盖准则的可行性。Web 应用系统作为一种主流的应用，其可靠性、安全性和可用性日益得到研究人员的关注。目前基于用户对话的 Web 应用测试是一种主流测试方法，借助捕获重放机制，测试人员通过搜集用户对话信息，可以生成大量测试用例。Sprenkle 等人提出 3 种基于需求的 TSM 方法和 3 种基于形式化概念分析的 TSM 方法，在实证研究中对这些不同的 TSM 方法从测试用例集规模、程序覆盖率、缺陷检测能力和方法执行开销上进行了对比。实证结论表明：基于形式化概念分析的 TSM 方法相对于基于需求的 TSM 方法更为经济有效[45]。随后 Sampath 等人在缩减后的测试用例集中通过对测试用例按照优先级进行排序以进一步提高回归测试效果[46]。

与传统的软件调试技术不同，缺陷定位（Fault Localization）技术通过对源代码、测试结果以及各种程序行为特征信息的计算分析，给出缺陷在源代码中的可能位置。Yu 等人选择了 10 种不同的 TSM 方法（基于语句的和基于向量的）和 4 种缺陷定位方法（Tarantula、SBI、Jaccard 和 Ochiai）来分析 TSM 方法对缺陷定位效果的影响，结果表明缺陷定位效果与选择的 TSM 方法有关。即基于语句的 TSM 方法在测试用例集规模上缩减率较高，但对缺陷定位效果存在负面影响，而基于向量的 TSM 方法在测试用例集规模上缩减率较小，但对缺陷定位效果的影响较小[47]。随后顾庆等人通过在识别出的冗余测试用例中挑选出少量测试用例来进一步提高缺陷定位效果[48]。

结束语 通过上述分析，可以看出 TSM 问题是回归测试中的一个有意义的研究热点，有着丰富的理论价值和应用前景。虽然国内外学者在该领域已经取得了一定的研究成果，但还存在一些值得进一步关注的问题，主要包括：

(1)新颖 TSM 方法的提出。传统 TSM 问题已被证实为是一个 NP-Complete 问题[8]，虽然研究人员已经提出一些有效的启发式方法，但研究成果表明这些方法离最优解仍有一定差距。今后仍需要研究人员继续提出更多新颖有效的 TSM 方法。一种方法是借助基于搜索的软件工程（Search based Software Engineering）研究领域的最新研究成果[49]。

(2)基于变异测试分析的 TSM 方法研究。变异测试[50]是一种基于缺陷的软件测试技术,已有研究在评估 TSM 方法有效性时一般采用手工植入的缺陷进行评估,他们假设这类缺陷与软件开发过程中产生的实际缺陷性质相似。但一些研究工作表明变异测试分析中产生的变异缺陷与实际缺陷更为相似,而且手工植入的缺陷对实证结论的有效性存在影响[51]。所以需要借助变异缺陷对已有的 TSM 方法的有效性进行重新评估。

(3)更为充分的实证研究。从表 4 可以看出,大部分实证研究采用的评测程序是西门子套件程序和 space 程序,这些程序因规模较小,所得结论很难适用于实际企业的大规模软件测试。在今后的实证研究中需要进一步识别出影响 TSM 方法有效性的内在影响因素,并采用有效统计方法去分析这些影响因素的实际影响效果。这些影响因素主要包括被测程序特征、测试用例集特征、代码修改类型、缺陷特征等。通过分析部分影响因素还可以进一步识别出相关属性。例如针对测试用例集特征可以考虑如下属性:测试用例集规模、测试用例输入的异构程度、测试用例的覆盖能力等。

(4)基于模型的 TSM 方法研究。目前基于 EFSM 模型的 TSM 已经取得一些研究进展,但与基于源代码的 TSM 研究相比,该领域的研究工作还较少。同时还可以考虑更多其他在软件开发流程中使用的模型,例如 UML 模型等。

(5)为 TSM 方法寻求新的应用领域。虽然 GUI 应用测试、Web 应用测试和缺陷定位上取得了一定的研究成果,但在这些应用领域里,还有很多研究工作可以继续开展。同时还需要积极探索 TSM 方法在其他特定测试领域内的应用,包括 Web 服务测试、并发程序测试、物联网和云计算平台测试等。

(6)与工业界实践相结合。论文中提到的方法一般借助对照实验方式进行有效性评估,所得结论难以适用于来自企业的大型软件产品。其次已有的 TSM 方法并未与企业目前已有的开发、测试和调试流程相结合。将学术界的研究成果成功应用到企业实践中,需要进一步提高相关工具的自动化程度、改善工具的用户界面并保证工具具有一定的鲁棒性。目前一种可行方案是借用 Eclipse 插件方式进行开发,可以有效地与开发工具进行融合。

相信针对这些问题提出的有效解决方案将提高软件企业内部的回归测试效率并大幅度降低软件开发和维护的费用。

参 考 文 献

[1] Rothermel G, Untch R J, Chu C. Prioritizing Test Cases for Regression Testing[J]. IEEE Transactions on Software Engineering, 2001, 27(10): 929-948

[2] Harrold M, Orso A. Retesting Software during Development and Maintenance[C]// Proceedings of Frontiers of Software Maintenance. Beijing: IEEE Press, 2008: 99-108

[3] Beizer B. Software Testing Techniques[M]. New York: Van No strand Reinhold, 1990

[4] Leung H, White L. Insights into Regression Testing[C]// Proceedings of the International Conference on Software Maintenance. Pittsburg: ACM Press, 1989: 60-69

[5] Malishevsky A G, Rothermel G, Elbaum S. Modeling the Cost-benefits Tradeoffs for Regression Testing Techniques[C]// Proceedings of the International Conference on Software Maintenance. Montreal: IEEE Press, 2002204-213

[6] Do H, Rothermel G. An Empirical Study of Regression Testing Techniques incorporating Context and Lifetime Factors and Improved Cost-benefit Models[C]// Proceedings of the Symposium on the Foundations of Software Engineering. Graz: Springer Press, 2006: 141-151

[7] Do H, Rothermel G. Using Sensitivity Analysis to Create Simplified Economic Models for Regression Testing[C]// Proceedings of the International Symposium on Software Testing and Analysis. Seattle: ACM Press, 2008: 51-62

[8] Harrold M J, Gupta R, Soffa M L. A Methodology for Controlling the Size of a Test Suite[J]. ACM Transactions on Software Engineering and Methodology, 1993, 2(3): 270-285

[9] Zhu H, Hall P, May J. Software Unit Test Coverage and Adequacy[J]. ACM Computing Survey, 1997, 29(4): 366-427

[10] Black J, Melachrinoudis E, Kaeli D. Bi-criteria Models for All-uses Test Suite Reduction[C]// Proceedings of the International Conference on Software Engineering. Edinburgh: IEEE Press, 2004: 106-115

[11] Zhong H, Zhang L, Mei H. An Experimental Study of Four Typical Test Suite Reduction Techniques[J]. Information and Software Technology, 2008, 50(6): 534-546

[12] Zhang L M, Marinov D, Zhang L, Khurshid S. An Empirical Study of Junit Test-Suite Reduction[C]// Proceedings of the International Symposium on Software Reliability Engineering. Hiroshima: IEEE Press, 2011: 170-179

[13] Chen T Y, Lau M F. A New Heuristic for Test Suite Reduction [J]. Information and Software Technology, 1998, 40(5/6): 347-354

[14] Chen T Y, Lau M F. On the Divide-and-conquer Approach towards Test Suite Reduction[J]. Information Sciences, 2003, 152 (1): 89-119

[15] Jones J A, Harrold M J. Test-Suite Reduction and Prioritization for Modified Condition/Decision Coverage[J]. IEEE Transactions on Software Engineering, 2003, 29(3): 195-209

[16] Tallam S, Gupta N. A Concept Analysis Inspired Greedy Algorithm for Test Suite Minimization[C]// Proceedings of the 6th workshop on Program Analysis for Software Tools and Engineering. Lisbon: ACM Press, 2005: 35-42

[17] Mansour N, El-Fakih K. Simulated Annealing and Genetic Algorithms for Optimal Regression Testing[J]. Journal of Software Maintenance: Research and Practice, 1999, 11(1): 19-34

[18] Jeffrey D, Gupta N. Test Suite Reduction with Selective Redundancy[C]// Proceedings of the International Conference on Software Maintenance. Budapest: IEEE Press, 2005: 549-558

[19] Jeffrey D, Gupta N. Improving Fault Detection Capability by Selectively Retaining Test Cases during Test Suite Reduction[J]. IEEE Transactions on Software Engineering, 2007, 33(2): 108-123

[20] Hsu H Y, Orso A. MINTS: A General Framework and Tool for Supporting Test-suite Minimization[C]// Proceedings of the International Conference on Software Engineering. Vancouver: IEEE Press, 2009: 419-429

[21] Yoo S, Harman M. Pareto Efficient Multi-objective Test Case Selection[C]// Proceedings of the International Symposium on

Software Testing and Analysis. London: ACM Press, 2007: 140-150

[22] Yoo S, Harman M. Using Hybrid Algorithm for Pareto Efficient Multi-objective Test Suite Minimization[J]. Journal of System and Software, 2010, 83(4): 689-701

[23] Harder M, Mellen J, Ernst M D. Improving Test Suites via Operational Abstraction[C]// Proceedings of the International Conference on Software Engineering. Portland: IEEE Press, 2003: 60-71

[24] McMaster S, Memon A M. Call Stack Coverage for Test Suite Reduction[C]// Proceedings of the International Conference on Software Maintenance. Budapest: IEEE Press, 2005: 539-548

[25] Hao D, Zhang L, Wu X, et al. On-demand Test Suite Reduction [C]// Proceedings of the International Conference on Software Engineering. Zurich: IEEE Press, 2012. 738-748

[26] Marre M, Bertolino A. Using Spanning Sets for Coverage Testing[J]. IEEE Transactions on Software Engineering, 2003, 29 (11): 974-984

[27] 章晓芳,徐宝文,聂长海,等. 一种基于测试需求约简的测试用例集优化方法[J]. 软件学报, 2007, 18(4): 821-831

[28] Chen Z Y, Xu B W, Zhang X F, et al. A Novel Approach for Test Suite Reduction based on Requirement Relation Contraction[C]// Proceedings of the ACM Symposium on Applied Computing. Fortaleza: ACM Press, 2008: 390-394

[29] 顾庆,唐宝,陈道蓄. 一种面向测试需求部分覆盖的测试用例集约简技术[J]. 计算机学报, 2011(5): 879-889

[30] Zhang L J, Chen X, Gu Q, et al. CATESR: Change-aware Test Suite Reduction based on Partial Coverage of Test Requirements [C]// Proceedings of International Conference on Software Engineering & Knowledge Engineering. San Francisco Bay: Knowledge Systems Institute Graduate School, 2012: 217-224

[31] Chen X, Zhang L J, Gu Q, et al. A Test Suite Reduction Approach based on Pairwise Interaction of Requirements[C]// Proceedings of the Symposium on Applied Computing. TaiChung: ACM Press, 2011: 1390-1397

[32] Vaysburg B, Tahat L H, Korel B. Dependence Analysis in Reduction of Requirement based Test Suites[C]// Proceedings of the International Symposium on Software Testing and Analysis. Roma: ACM Press, 2002: 107-111

[33] Korel B, Tahat L, Vaysburg B. Model based Regression Test Reduction using Dependence Analysis[C]// Proceedings of the International Conference on Software Maintenance. Montreal: IEEE Press, 2002: 214-225

[34] Chen Y, Probert R L, Ural H. Regression Test Suite Reduction using Extended Dependence Analysis[C]// Proceedings of the International Workshop on Software Quality Assurance. Dubrovnik: ACM Press, 2007: 62-69

[35] Do H, Elbaum S, Rothermel G. Supporting Controlled Experimentation with Testing Techniques: an Infrastructure and its Potential Impact[J]. Empirical Software Engineering, 2005, 10 (4): 405-435

[36] Hutchins M, Foster H, Goradia T, et al. Experiments on the Effectiveness of Dataflow and Control-flow-based Test Adequacy Criteria[C]// Proceedings of the International Conference on Software Engineering. Sorrento: ACM Press, 1994: 191-200

[37] McMaster S, Memon A M. Fault Detection Probability Analysis for Coverage-Based Test Suite Reduction[C]// Proceedings of the International Conference on Software Maintenance. Paris: IEEE Press, 2007: 335-344

[38] Chen T Y, Lau M. A Simulation Study on some Heuristics for Test Suite Reduction[J]. Information and Software Technology, 1998, 40(13): 777-787

[39] Wong W E, Horgan J R, London S, et al. Effect of Test Set Minimization on Fault Detection Effectiveness[J]. Software Practice and Experience, 1998, 28(4): 347-369

[40] Wong W E, Horgan J R, Mathur A P, et al. Test Set Size Minimization and Fault Detection Effectiveness: A Case Study in a Space Application[J]. The Journal of Systems and Software, 1999, 48(2): 79-89

[41] Rothermel G, Harrold M J, Ostrin J, et al. An Empirical Study of the Effects of Minimization on the Fault Detection Capabilities of Test Suites[C]// Proceedings of International Conference on Software Maintenance. Maryland: IEEE Press, 1998: 34-43

[42] Rothermel G, Harrold M, Ronne J, et al. Empirical Studies of Test Suite Reduction[J]. Software Testing, Verification, and Reliability, 2002, 4(2): 219-249

[43] McMaster S, Memon A M. Call Stack Coverage for GUI Test-Suite Reduction[C]// Proceedings of International Symposium on Software Reliability Engineering. North Carolina: IEEE Press, 2006: 33-44

[44] McMaster S, Memon A F. Call-Stack Coverage for GUI Test Suite Reduction[J]. IEEE Transactions on Software Engineering, 2008, 34(1): 99-115

[45] Sprenkle S, Sampath S, Gibson E, et al. An Empirical Comparison of Test Suite Reduction Techniques for User-session-based Testing of Web Applications[C]// Proceedings of International Conference on Software Maintenance. Budapest: IEEE Press, 2005: 587-596

[46] Sampath S, Bryce R. Improving the Effectiveness of Test Suite Reduction for User-Session-Based Testing of Web Applications [J]. Information and Software Technology, 2012, 54(7): 724-738

[47] Yu Y, Jones J A, Harrold M J. An Empirical Study of the Effects of Test-suite Reduction on Fault Localization[C]// Proceedings of the International Conference on Software Engineering. Leipzig: ACM Press, 2008: 201-210

[48] Zhang X, Gu Q, Chen X, et al. A Study of Relative Redundancy in Test-suite Reduction while Retaining or Improving Fault-localization Effectiveness[C]// Proceedings of the ACM Symposium on Applied Computing. Sierre: ACM Press, 2010: 2229-2236

[49] Harman M. The Current State and Future of Search Based Software Engineering[C]// Proceedings of Workshop on the Future of Software Engineering. Minneapolis: ACM Press, 2007: 342-357

[50] 陈翔,顾庆. 变异测试:原理、优化和应用[J]. 计算机科学与探索, 2012, 6(12): 1057-1075

[51] Andrews J H, Briand L C, Labiche Y. Is Mutation an Appropriate Tool for Testing Experiments[C]// Proceedings of International Conference on Software Engineering. St Louis: ACM Press, 2005: 402-411