面向故障定位的基于MC/DC的测试用例约简方法

王 瑞[1,2] 田宇立[1] 周东红[2] 李 宁[1] 李战怀[1]
(西北工业大学计算机学院 西安 710072)[1] (北京信息控制研究所 北京 100048)[2]

摘 要 对不断更新的软件进行回归测试时,持续增加的测试用例会造成累计测试用例数量庞大,进而影响测试成本。在故障定位领域,已有研究在考虑语句覆盖、路径覆盖等的基础上,提出了CMR&PVR等不同的测试用例约简方法。然而,这些方法在一定程度上影响了原始测试用例集的MC/DC(修订的条件/判定)覆盖率。提出一种以MC/DC覆盖为基础的综合测试用例约简方法MCDCR,利用该方法对原始测试用例集约简后,在确保原有故障定位准确性并保持较高约简比的同时,大幅提高了测试用例对程序的MC/DC覆盖率。采用Ochiai方法在Siemens程序集上进行了实验及验证,结果表明MCDCR约简方法的综合效果明显优于已有的约简方法。

关键词 软件故障定位,测试用例约简,MC/DC覆盖率
中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2015.10.034

Test-suite Reduction Based on MC/DC in Software Fault Localization

WANG Rui[1,2] TIAN Yu-li[1] ZHOU Dong-hong[2] LI Ning[1] LI Zhan-huai[1]
(School of Computer Science, Northwestern Polytechnical University, Xi'an 710072, China)[1]
(Beijing Institute of Information and Control, Beijing 100048, China)[2]

Abstract In the process of software regression testing, frequently modifying software leads to a huge test suite which makes testing more expensive. To address this problem, researches have proposed methods about test suite reduction in consideration of statement/path coverage. However, these methods more or less affect the integrity of MC/DC coverage of the original test suite. We proposed a new approach named MCDCR based on MC/DC coverage rate. Our MCDCR method can guarantee MC/DC coverage while doing no harm to the effectiveness of fault localization and test suite reduction rate. Experiment shows that MCDCR performs better than the existing reduction methods comprehensively.

Keywords Software fault localization, Test-suite reduction, MC/DC coverage

1 引言

在软件测试领域,为了减少软件故障定位所消耗的时间,近年来出现了很多自动化的故障定位方法[1,2]。依据其定位过程是否需要运行软件的准则,将其分为两大类[3]:基于静态分析的静态故障定位和基于测试的动态故障定位。基于测试的动态故障定位主要是设计测试用例并运行软件,依据程序执行成功或者失败的相关追踪信息进行故障定位,例如经典的Tarantula[1]等。

动态故障定位方法近年来受到了广泛的关注。动态故障定位与测试用例的执行紧密相关,测试用例越多,执行测试用例所花费的时间代价就越大。如何约简测试用例,提高测试效率,成为亟需解决的问题。测试用例约简问题的研究重点是从不同的角度判断测试用例的相似性和相关性[5-10]。陈翔等人综述了回归测试中现有的测试用例约简技术[4],而在动态故障定位领域中,如何在不降低故障定位准确率的前提下约简测试用例是人们关心的问题。Chen等人[5]提出的轻量级测试用例约简方法可以定位到更多的故障。Yu等人[7]提出了基于语句(SA)和基于向量(VA)的测试用例约简方法,使得约简后的测试用例满足了语句覆盖和路径向量覆盖。Jones等人[8]利用MC/DC覆盖信息对原始测试用例直接进行约简,保证了约简后测试用例的MC/DC覆盖率,但对Siemens套件中的tcas程序进行故障定位的准确性平均损失率高达43.7%。Gong等人[9]在传统的基于语句和路径覆盖约简的基础上分析了测试执行路径,消除了由于程序循环所产生的相似路径环,在大幅约简测试用例的同时提高了故障定位准确性。

MC/DC覆盖是一种适用于复杂组合条件的覆盖准则,被广泛应用于航天等安全关键领域的软件测试中,可检测出语句覆盖和路径覆盖无法发现的软件故障。文献[10]按照不同覆盖准则对原始用例集选择出符合某种覆盖的测试用例集进行实验,结果表明当选择出的测试用例集满足MC/DC覆盖时,故障定位效果优于用例集满足语句覆盖和分支覆盖的效果。为了均衡考虑测试用例集的约简比率、MC/DC覆盖

到稿日期:2015-01-12 返修日期:2015-04-18 本文受国家自然科学基金(61402370),中国航天科技集团公司航天科技创新基金(2014H03FK011)资助。

王 瑞(1978-),男,博士生,研究员,主要研究方向为软件测试与分析,E-mail:hustwr@aliyun.com;田宇立(1990-),男,硕士生,主要研究方向为软件故障分析与定位;周东红(1978-),男,硕士,工程师,主要研究方向为软件测试;李 宁(1978-),女,博士,讲师,主要研究方向为软件测试与分析、软件库挖掘;李战怀(1951-),男,博士,教授,博士生导师,主要研究方向为软件工程、数据管理。

率以及故障定位准确性，本文提出一种以 MC/DC 覆盖为基础的综合测试用例约简方法 MCDCR，使得约简后的测试用例集在约简比较高且不损失故障定位准确性的同时，提高了测试用例的 MC/DC 覆盖率。

本文第 2 节介绍了已有的基于语句覆盖和执行向量的测试用例约简方法；第 3 节详细描述了本文提出的以 MC/DC 覆盖为基础的综合测试用例约简方法 MCDCR；第 4 节展示了实验的结果；最后对本文工作进行了简单总结。

2 基于语句覆盖和执行向量的测试用例约简

2.1 基于语句覆盖矩阵的测试用例约简法(CMR)

语句覆盖矩阵(Coverage Matrix)是动态故障定位方法中普遍使用的一种信息。它指执行若干个测试用例时产生的语句覆盖信息。覆盖矩阵 *COVER* 中的行代表测试用例编号，列代表程序语句的编号，其中 $COVER[i,j]=0$ 表示第 i 个测试用例未通过第 j 条语句，$COVER[i,j]=1$ 表示第 i 个测试用例通过第 j 条语句。

通常认为被失败测试用例执行过的语句出错的概率更高，被成功用例和失败用例都执行过的语句出错概率相对较小。根据该分析，综合所有失败用例的覆盖信息，文献[9]定义了如下的故障定位需求向量 *FLreq* 和弱相关测试用例。*FLreq* 可描述故障定位中与故障可能直接相关的语句。

定义 1(故障定位需求向量 *FLreq*)　对于仅包含一处错误的程序，每个失败测试用例都应通过包含错误的语句，$FLreq= COVER(t_1)\cap COVER(t_2)\cap\cdots\cap COVER(t_m)$。而对于包含多处错误的程序，每个失败用例可能通过不同的错误语句，$FLreq=COVER(t_1)\cup COVER(t_2)\cup\cdots\cup COVER(t_m)$。

定义 2(弱相关测试用例)　假设程序有 n 条语句，若测试用例 t_1 的每个覆盖值与 t_2 的对应覆盖值相与为 0，则称测试用例 t_1 与 t_2 是弱相关的。$COVER(t_1)\cap COVER(t_2)=\langle t_{11},\cdots,t_{1i},\cdots,t_{1n}\rangle\cap\langle t_{21},\cdots,t_{2i},\cdots,t_{2n}\rangle=\langle z_1,\cdots,z_i,\cdots,z_n\rangle$，其中 $z_i=t_{1i}\wedge t_{2i}=0(1\leqslant i\leqslant n)$。

文献[9]中提出的基于语句覆盖矩阵的测试用例约简法(Coverage Matrix based Reduction，CMR)分析语句覆盖矩阵和故障定位需求向量 *FLreq*，删除与 *FLreq* 弱相关的测试用例，达到约简的目标。该方法主要约简了成功执行的测试用例，失败执行的测试用例保持不变。

2.2 基于执行路径向量的用例约简法(PVR)

语句覆盖矩阵可以反映出测试用例是否执行了某一条语句，却无法获知语句的执行次数、执行顺序等信息。有些测试用例虽然拥有相同的语句覆盖，但其执行路径有所不同。不同的执行路径包含了不同的程序语义信息，这些信息对于提升故障定位的准确率很有价值。

测试用例 t 的执行路径 PATH(t)是指测试用例 t 的执行语句序列。如果程序中含有循环结构，则路径 PATH(t)中会存在一个重复执行多次的语句序列。为了消除仅仅因为循环次数对执行路径产生的影响，可以对路径中的循环结构规范化。具体规范化方法如下：记录重复执行的语句序列和相应的重复次数，然后将这些语句序列从路径 PATH(t)中删除。

文献[9]中提出的基于执行路径向量(Path Vector Reduction，PVR)的约简方法是通过删除执行路径相同或相似的重复测试用例来达到约简用例的目标。该文献给出了相同路径和相似路径的定义，其中相同路径表示为 $PATH(t_1)==PATH(t_2)$，相似路径表示为 $PATH(t_1)\approx PATH(t_2)$。相似路径指进行循环结构规范化后路径相同的执行序列。与 CMR 方法不同，PVR 方法对失败用例和成功用例都会实行约简。

3 基于 MC/DC 覆盖的用例约简法(MCDCR)

3.1 MC/DC 覆盖准则

MC/DC(修订的条件/判定覆盖)(Modified Condition / Decision Coverage)准则最早由波音公司创建，是一种已被广泛地应用于软件测试的软件结构覆盖率测试准则。MC/DC 覆盖要求在每个判定中的每个条件都曾至少一次独立地影响判定的结果。

以图 1 所示的 Siemens 套件中的 tcas.c 程序片段为例。

```
If(enabled &&
( (tcas_equipped&&intent_not_known)
   || !tcas_equipped)))
{
...
}
```

图 1　tcas.c 程序片段

图 1 中 enabled，tcas_equipped，intent_not_known，!tcas_equipped 分别称为 4 个条件 C1，C2，C3，C4，这 4 个条件构成 1 个判定，该语句对应的判定函数是：

$f(C1,C2,C3,C4)=C1\ \&\ ((C2\ \&\ C3)|C4)$

具体的条件组合如表 1 所列，T 表示 true，F 表示 false，其中加 # 备注的是满足最小 MC/DC 覆盖的测试条件取值。

表 1　MC/DC Pair 示例

序号	条件取值	期望输出	MC/DC 最小条件取值集合
v1	f(T,T,T,T)	T	
v2	f(T,T,T,F)	T	#
v3	f(T,T,F,T)	T	#
v4	f(T,F,T,T)	T	
v5	f(T,T,F,F)	F	#
v6	f(T,F,T,F)	F	#
v7	f(T,F,F,T)	T	
v8	f(T,F,F,F)	F	
v9	f(F,T,T,T)	F	
v10	f(F,T,T,F)	F	#
v11	f(F,T,F,T)	F	
v12	f(F,F,T,T)	F	
v13	f(F,T,F,F)	F	
v14	f(F,F,T,F)	F	
v15	f(F,F,F,T)	F	
v16	f(F,F,F,F)	F	

每个条件独立影响判定结果的测试用例分析如下：

C1 独立影响：v2 f(**T**,T,T,F)，v10 f(**F**,T,T,F)

C2 独立影响：v2 f(T,**T**,T,F)，v6 f(T,**F**,T,F)

C3 独立影响：v2 f(T,T,**T**,F)，v5 f(T,T,**F**,F)

C4 独立影响：v3 f(T,T,F,**T**)，v5 f(T,T,F,**F**)

上例中的 v2 和 v10 称为一个 MC/DC Pair，即 v2 与 v10 相比，只有 C1 取值不同，且 C1 分别取 true 和 false 时，整个判定结果不同(分别为 true 和 false)，即 C1 条件独立影响整个判定的值。对条件 C1－C4 分别构造 MC/DC Pair，在删除重复条件取值后，可知满足该判定的 MC/DC 最小条件取值集合由以下 5 种条件取值构成：{v2，v3，v5，v6，v10}。

3.2 基于 MC/DC 覆盖的测试用例约简法

为了确保对测试用例进行 MC/DC 覆盖率约简后的故障

定位准确性，在提出的基于 MC/DC 覆盖的测试用例约简方法(MC/DC Reduction，MCDCR)中，首先利用基于覆盖矩阵和执行路径向量的方法(CMR&PVR)对测试用例约简。然后，在不降低故障定位准确率的前提下，根据用例对 MC/DC 覆盖率的贡献度(见定义 3)，将贡献度最大的用例逐渐添加到用例集中，直到所有的 MC/DC 条件被完全覆盖。

定义 3(测试用例 MC/DC 贡献度) 假设 uncoveredMCDC 表示尚未被覆盖的 MC/DC 条件集合，每个测试用例的贡献度指执行测试用例 t_i 时所覆盖的 uncoveredMCDC 中的 MC/DC 条件的数量。

MCDCR 方法需要有 3 个初始输入信息：1)未约简之前的原始测试用例集 allTest；2)针对被测程序 P，满足 MC/DC 最小覆盖的所有 MC/DC Pair 的条件取值集合 MCDCpairs；3)利用 CMR&PVR 方法约简后得到的最简测试用例集 simplestTests。

MDCDR 算法如图 2 所示，其核心思想是：针对逐步约简后的测试用例集合 targetTests(初值为 simplestTests)，迭代计算 allTest 与 targetTests 的差集中测试用例 MC/DC 贡献度最大的用例，将其添加至目标用例集合中，添加时 Failed 用例的优先级高于 Passed 用例；然后从剩余测试用例集中删除该条测试用例，从 uncoveredMCDC 中删除本轮迭代新覆盖到的 MC/DC pair。迭代到 uncoveredMCDC 为空，或测试用例 MC/DC 贡献度最大值为 0，或剩余测试用例集为空时，算法结束。

```
算法：MCDCReduce(alltests，MCDCpairs，simplestTests)
Input：allTests：the set of test cases for Program P
       MCDCpairs：all MC/DC pairs for Program P
       simplestTests：set of test cases reduced by CMR&PVR
Output：targetTests：testcases reduced by MCDCR
        uncoveredMCDC：uncovered MC/DC pairs by targetTests
1. coveredMCDCpairs=MC/DC pairs covered by simplestTests
2. tempMC/DC=allMC/DCpairs-coveredMC/DC
3. targetTests=simplestTests
4. tempTests=allTests-targetTests
5. uncoveredMC/DC=tempMC/DC
6. while(tempTests !=φ || tempMC/DC != φ){
7.   for each test t in tempTests {
8.        calculate t. contribution
9.   }
10.  if the Max(contribution) == 0 {
11.    uncoveredMC/DC=tempMC/DC;
12.    break;
13.  }else {
14.    nextTest=testcase which has max(contribution)
15.    tCoveredMC/DC=MC/DC pairs covered by nextTest
16.  }
17.  targetTests=targetTests+{nextTests}
18.  tempTests=tempTests-{nextTests}
19.  tempMC/DC=tempMC/DC-tCoveredMC/DC
20.}
21. if(tempMC/DC !=φ)
22.   uncoveredMC/DC=tempMC/DC
23. return targetTests，uncoveredMCDC
```

图 2 MCDCR 约简算法

MCDCR 算法约简结束时，若 uncoveredMCDC 为空，表明最终得到的 targetTests 覆盖了所有 MC/DC Pair；测试用例 MC/DC 最大贡献度为 0，表明剩余未选择的测试用例都不能覆盖新的 MC/DC Pair，无需再选择新的测试用例；最终若 uncoveredMCDC 不为空，表明原始测试用例集不能覆盖全部 MC/DC Pair，可根据未覆盖 MC/DC 信息指导测试用例的补充。

4 实验

4.1 实验原始数据

为了便于与已有的研究进行对比，使用了在软件故障定位领域中广泛使用的公开数据集 Siemens 程序套件(从 Software-artifact Infrastructure Repository 网站下载)进行实验验证。Siemens 数据集包含如表 2 所列的 7 个 C 语言程序。由于 schedule 与 schedule2，print_tokens 与 print_tokens2 在程序功能、规模、测试用例数目、程序逻辑等方面基本相同，最终实验选取了 schedule 和 print_tokens2 进行验证。

表 2 Siemens 程序集信息

测试程序	规模	错误版本数	测试用例数
tcas	135	41	1608
print_tokens	344	7	4130
print_tokens2	355	10	4115
replace	512	32	5542
schedule	292	9	2650
schedule2	262	10	2710
tot-info	273	23	1052

本实验修改了开源软件 Loupe[11] 的部分代码以获取每个程序的执行路径，针对被测程序，利用工具软件 LDRA testbed 获取满足 MC/DC 最小覆盖的所有 MC/DC Pair 构成的条件取值集合 MCDCpairs。

4.2 度量

故障定位准确率采用了公认效果较好的 Ochiai[12] 方法计算可疑度，如式(1)所示，根据每条语句可疑度(Suspiciousness)值进行降序排列，确定每条可疑语句的位次(Rank)。*Suspiciousness* 越大，表明该条语句越值得怀疑，*Rank* 值越小。

$$Suspiciousness=\frac{failed(s)}{\sqrt{totalfailed*(passed(s)+failed(s))}} \tag{1}$$

式中，$failed(s)$表示所有执行失败的用例中通过语句 s 的用例个数，$passed(s)$表示所有执行成功的用例中通过语句 s 的用例个数，$totalfailed$ 表示所有执行失败的用例个数。由于本实验的主要对比方法 CMR&PVR 在约简过程中会出现 $totalpassed$ 为 0 的情况，*tarantula* 的公式不能直接使用，因此本实验中采用 Ochiai 方法计算可疑度。

故障定位准确性采用式(2)的 *Score* 进行度量，*Score* 越高表明故障定位的准确性越高。其中 *Total* 表示所有参与排名的程序实体总个数。例如有 100 条语句进行排名，真正出错语句排在最前面，即 $Rank=1$ 时，*score* 为 99%，表明只需检查 1 条语句就可迅速定位到故障。

$$Score=(1-\frac{Rank}{Total})*100\% \tag{2}$$

实验中采用 Reduction[10] 度量测试用例的约简比例，用 McdcCov 反映测试用例集的 MC/DC 覆盖率。

$$Reduction=(1-\frac{\text{number of reduced testcases}}{\text{number of original testcases}})\times 100\% \quad (3)$$

$$McdcCov=\frac{N_{reduced\ MC/DCcov}}{N_{original\ MC/DCcov}}\times 100\% \quad (4)$$

式中，$N_{original\ MC/DC\ cov}$ 和 $N_{reduced\ MC/DC\ cov}$ 分别表示原始测试用例和约简后测试用例所能覆盖到的 MC/DC 最小条件取值集合中的条件个数。

本文提出一种新的度量 McdcScore 用于描述面向故障定位的测试用例约简方法的有效性。该度量是一个可以反映出 MC/DC 覆率和故障定位准确率的综合指标。

$$McdcScore=Score\times McdcCov \quad (5)$$

4.3 结果与分析

由于 Gong 等人在文献[9]的实验结果中已经验证了其提出的 CMR&PVR 方法在测试用例约简比率和约简后的故障定位准确率方面都优于文献[7]提出的基于语句(SA)的约简和基于向量(VA)的约简方法，因此本文的实验结果主要与原始数据(Original)、CMR&PVR 方法约简后的数据以及文献[8]中提出的测试用例约简方法所得结果进行对比，具体实验结果对比如表 3－表 8 所列，每个程序的度量值都是该程序多个错误版本度量值的平均值。

文献[9]在实现 CMR&PVR 方法时未描述实验的部分细节，因此本文的实现细节可能与原文略有差异，采用的可疑度计算公式也不同。此外，由于实际错误定位时，每个程序包含一个错误还是多个错误是未知的，且多个错误可以转化为单个错误逐一定位，因此本文实验中均按照单错误计算 *FL-req*。由于以上差异，表 3 中的 CMR&PVR 的约简比以及表 5 中的 Score 差与文献[9]中的实验结果并不完全相同，但整体趋势基本相同。

表 3　Siemens 程序集约简比 Reduction(%)

方法	tcas	tot-info	schedule	print_tokens2	replace	平均
MCDCR	98.93	72.01	25.38	72.68	50.43	63.89
CMR&PVR	99.33	72.01	25.38	72.68	50.45	63.97

表 3 的结果表明，MCDCR 方法仅比 CMR&PVR 方法的约简比低 0.08%，可以说基本持平。例如对 tcas 程序 1608 条测试用例的各个版本进行 MCDCR 约简后，平均剩余 17.2 条用例，采用 CMR&PVR 约简后的平均剩余为 10.8 条用例。与原始的 1608 条测试用例相比，增加的 6.4 条用例对执行时间的影响几乎可忽略不计。

表 4 列出了各个程序所有版本的平均 MC/DC 覆盖率。以 tcas 的 v1 版本为例来说明 MC/DC 覆盖率的计算：该程序满足最小 MC/DC 覆盖需要有 40 种条件取值，原始数据集的 1608 条测试用例最大可以覆盖其中 29 种取值，即该版本的原始 MC/DC 覆盖率为 72.5%(29/40)。利用 CMR&PVR 方法约简用例后，仅能覆盖 40 个条件中的 19 个，即相对 *McdcCov* 为 47.5%(19/40)，利用 MCDCR 方法约简后可以覆盖 29 个条件，即相对 *McdcCov* 为 72.5%(29/40)。从表 4 可知，与 CMR&PVR 方法相比，MCDCR 方法将约简后测试用例集的 MC/DC 覆盖率提高到最大值，即与原始用例集 MC/DC 覆盖保持相同水平，有效确保了测试用例的 MC/DC 覆盖的完整性。

表 4　Siemens 程序集的 MC/DC 覆盖率 McdcCov(%)

MC/DC 覆盖率	tcas	tot-info	schedule	print_tokens2	replace	平均
Original	74.47	82.61	60	95.30	65.76	75.63
MCDCR	74.47	82.61	60	95.30	65.76	75.63
CMR&PVR	45.01	82.61	60	95.30	64.05	69.39

值得注意的是，从实验中发现以上 5 个被测程序的测试用例集的原始 MC/DC 覆盖率都不能达到 100%，即从 MC/DC 覆盖率的角度看，Siemens 套件中的原始测试用例集是不充分的，若要满足最小 MC/DC 覆盖，需对用例进行扩充。

表 5 中的 Score 是按照式(2)计算的，多个语句可疑度相等时，本文采用 LastLine 策略，即取并列语句中 *Rank* 的最大值作为这些语句的 *Rank*。从表 5 可知，采用 CMR&VR 或 MCDCR 方法对测试用例约简后，故障定位的平均准确性都略有提高，且 MCDCR 方法略高于 CMR&PVR 方法。表 6 的数据显示 MCDCR 方法的综合约简效果优于 CMR&PVR 方法，McdcScore 提高了 3.57%，特别是 tcas 的 McdcScore 提高了 16.33%。综合分析以上数据可知，相对于原始测试用例集，MCDCR 方法在测试用例约简比高达 63.89%时，依然可与原始测试用例集保持相同的 MC/DC 覆盖率，且故障定位准确率略有提升，该结果优于已有方法的结果。

表 5　Siemens 程序集的故障定位准确性 Score(%)

Score	tcas	tot-info	schedule	print_tokens2	replace	平均
Original	50.13	82.06	96.46	99.38	89.75	83.56
MCDCR	55.27	86.22	96.04	99.38	89.15	85.21
CMR&PVR	51.42	86.22	96.04	99.38	89.15	84.44
MCDCR-Org	**5.14**	4.16	−0.42	0	−0.60	**1.66**
CMR&PVR-Org	1.29	4.16	−0.42	0	−0.60	0.89

表 6　Siemens 程序集综合 McdcScore(%)

McdcScore	tcas	tot-info	schedule	print_tokens2	replace	平均
Original	36.68	67.82	57.88	94.71	59.01	63.22
MCDCR	40.47	71.28	57.62	94.71	58.61	64.53
CMR&PVR	24.13	71.28	57.62	94.71	57.09	60.97
MCDCR-Org	**3.79**	**3.46**	**−0.25**	**0**	**−0.40**	**1.32**
CMR&PVR-Org	−12.54	3.46	−0.25	0	−1.92	−2.25

表 3 的数据显示 tcas 测试用例集的约简比高达 98%，测试用例样本较多时该值比较合理，但若测试用例样本较小时，MCDCR 方法是否会发生约简过度的可能，对故障定位准确性是否会发生负面影响。针对该问题，本文对 tcas 实施了进一步的实验，实验中对每个版本的 tcas 从原始测试用例(1608 个)中随机抽取若干测试用例，生成 200 个测试用例集，其中每个用例集中包含的用例个数为[1,1608]内的随机数，且 fail 和 pass 用例的比例与原始数据集比例保持一致。分别执行不同约简方法，得到如图 3 和图 4 所示的结果。

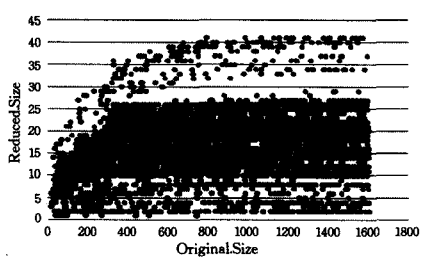

图 3　采用 CMR&PVR 方法的测试用例约简比

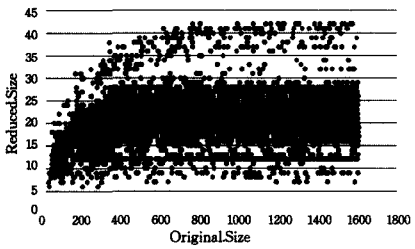

图 4　采用 MCDCR 方法的测试用例约简比

从图 3 和图 4 可知，利用 MCDCR 和 CMR&PVR 方法约简测试用例时，不论原始测试用例样本大小，约简后的测试用例数目始终维持在一个相对稳定的范围内(1－45 之间)。此外，CMR&PVR 方法约简后的测试用例个数为 1－5 的大约占到了 6%，对于条件判断比较复杂的 tcas 程序而言，这些用例明显不足，图 4 中 MCDCR 的约简结果更加合理。表 7 列出了实验的相关度量，结果表明，测试用例样本个数并不会显著影响 MCDCR 方法的综合约简效果。

表 7　tcas 随机测试结果(%)

	Reduction	McdcCov	Score	McdcScore
Original	/	74.26	52.27	38.11
MCDCR	95.95	74.26	59.96	43.83
CMR&PVR	96.88	57.17	58.81	33.39
MCDCR-Org	/	0	7.69	**5.72**
CMR&PVR-Org	/	−17.09	6.54	**−4.72**

表 8 将 MCDCR 方法与文献[8]中的两种 MCDC 约简方法进行了对比，列出了各种方法关于 tcas 程序的实验结果。其中文献[8]中的 break-down 和 build-up 约简的实验结果直接来源于该文献。由表 8 可知，break-down 约简和 build-up 约简后的用例集 size 略小于 MCDCR 法，但约简引起的故障定位准确性损失率却不容忽视，都高达 40%以上；而 MCDCR 对 tcas 程序进行约简后，表示故障定位准确性的 Score 由 50.13%提高到 55.27%(见表 5)。由于 MCDCR 的 Score 与文献[8]的故障定位准确性的计算方法不同，因此无法直接在表 8 中比较，但 MCDCR 方法对故障定位准确性并未产生显著的负面效应，即本文的 MCDCR 方法在保持较高约简比的同时并未降低故障定位准确性，与文献[8]的方法相比优势明显。

表 8　3 种 MCDC 约简方法对 tcas 程序的约简结果

	约简后用例集的平均 size	fault-detection loss
MCDCR	17.2	—
break-down	9.9	44.40%
build-up	10.1	43.70%

结束语　面向软件故障定位研究测试用例约简问题时，应综合考虑测试用例的约简比率、程序覆盖率以及约简对故障定位准确性的影响。已有的基于语句覆盖和基于路径向量的约简方法在约简比和故障定位准确性方面有着较好的表现，但面对安全攸关的软件时却无法满足测试用例的最小 MC/DC 覆盖。本文提出了以 MC/DC 覆盖为基础的综合测试用例约简方法 MCDCR，并提出用 McdcScore 综合度量故障定位准确率和 MC/DC 覆盖率。实验表明，本文提出的 MCDCR 方法的约简综合效果优于 CMR&PVR 方法。此外，该方法还可检测原始测试用例集的设计是否符合 MC/DC 最小覆盖，并获得未覆盖的 MC/DC 条件取值以指导测试用例的扩充。从实验中发现，Siemens 套件中提供的原始测试用例集对程序的 MC/DC 覆盖率未能达到 100%。目前，本文在获取程序执行路径时采用的方法效率较低。此外，本文实验采用的数据集是公开数据集，但程序规模尚不够大，今后有待于在大规模的实际工程数据中进行进一步验证和完善。

参 考 文 献

[1] Jones JA, Harrold MJ, Stasko J. Visualization of test information to assist fault localization[C]// 24th International Conference on Software Engineering(ICSE2002). 2002:467-477

[2] Wong WE, Debroy V. A survey of software fault localization. Technical report[R]. The University of Texas at Dallas, 2009

[3] 鞠小林，姜淑娟，张艳梅，等. 软件故障定位技术进展[J]. 计算机科学与探索，2012，6(6)：481-494
Ju Xiao-lin, Jiang Shu-juan, Zhang Yan-mei, et al. Advances in fault localization techniques[J]. Journal of Frontiers of Computer Science and Technology, 2012, 6(6): 481-494

[4] 陈翔，顾卫江，徐慧，等. 回归测试用例选择技术研究综述[J]. 计算机科学，2013，40(10)：1-9
Chen Xiang, Gu Wei-jiang, Xu Hui, et al. Regression Testing Selection Techniques: A State-of-the-art Review[J]. Computer Science, 2013, 40(10): 1-9

[5] Chen Zhen-yu, Xu Bao-wen, Zhang Xiao-fang, et al. A novel approach for test suite reduction based on requirement relation contraction[C]// 23rd ACM symposium on Applied computing (SAC2008). Fortaleza, Ceará, Brazil, 2008: 390-394

[6] Hao Dan, Xie Tao, Zhang Lu, et al. Test input reduction for result inspection to facilitate fault localization[J]. Automated Software Engineering, 2010, 17: 5-31

[7] Yu Yan-bing, Jones J A, Harrold M J. An empirical study of the effects of test suite reduction on fault localization[C]// Proceeding of 30th International Conference on Software Engineering (ICSE2008). Leipzig, 2008: 201-210

[8] Jones J A, Harrold M J. Test-suite reduction and prioritization for modified condition/decision coverage[J]. IEEE Transactions of Software Engineering, 2003, 29(3): 195-209

[9] Gong Dan-dan, Wang Tian-tian, Su Xiao-hong, et al. A test-suite reduction approach to improving fault-localization effectiveness [J]. Computer Languages, Systems & Structures, 2013, 39(3): 95-108

[10] Jiang Bo, Zhai Ke, Tse T H, et al. On the adoption of MC/DC and control-flow adequacy for a tight integration of program testing and statistical fault localization[J]. Information and Software Technology, 2013, 55(5): 897-917

[11] Yu Kai, Lin Meng-xiang, Gao qing, et al. Locating faults using multiple spectra-specific models[C]// 26th Annual ACM Symposium on Applied Computing (SAC2011). TaiChung, Taiwan, 2011: 1404-1410

[12] Abreu R, Zoeteweij P, Van Gemund A J C. An evaluation of similarity coefficients for software fault localization[C]// 2nd Pacific Rim International Symposium on Dependable Computing (PRDC2006). 2006: 39-46