面向语句的MBFL变异体约减策略

王林鑫 王微微 赵瑞莲 李 征
(北京化工大学信息科学与技术学院 北京 100029)

摘 要 在软件调试过程中如何高效、精确地定位程序中的错误代码是软件开发人员普遍关注的问题。MBFL是一种基于变异分析的错误定位技术,它在获得较高错误定位精度的同时会生成大量变异体,并在变异体上执行测试用例集,开销庞大。为了减少MBFL的变异执行开销,提出面向语句的变异体约减策略,通过分析测试用例的执行信息,按一定比例对每条由失败测试用例覆盖的语句生成的变异体集合进行约减。实验结果表明,在7个程序包的112个错误版本上,应用面向语句的变异体约减策略的MBFL,在保持较高错误定位精度的同时,能够有效减少73.51%～79.98%的变异执行开销。

关键词 错误定位,变异分析,变异体约减策略
中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2017.11.026

MBFL with Statement-oriented Mutant Reduction Strategy

WANG Lin-xin WANG Wei-wei ZHAO Rui-lian LI Zheng
(College of Information Science and Technology, Beijing University of Chemical Technology, Beijing 100029, China)

Abstract How to efficiently and accurately locate faults in programs during the process of software debugging is taken up as a matter of common concern for software developers. MBFL is a fault localization technique based on mutation analysis, which precisely identifies the root cause of fault but incurs a high execution cost, since it needs to execute the test suite on a large amount of mutants. For decreasing the execution cost of MBFL, this paper presented a statement-oriented mutant reduction strategy, which selects a certain proportion set of mutants generated by statements covered by failed tests, according to the previous execution information of test suite. Empirical studies were conducted on 112 faulty versions from 7 program packages. The results indicate that this strategy can reduce 73.51%～79.98% mutation execution cost under the case of keeping high fault location precision.

Keywords Fault localization, Mutation analysis, Mutant reduction strategy

1 引言

软件调试是软件开发和维护过程中的重要任务,难度较高且开销庞大。错误定位(Fault Localization)作为软件调试的关键步骤,近年来被学术界广泛关注[1]。基于程序谱的错误定位方法(Spectrum-based Fault Localization, SBFL)通过收集测试用例在代码实体上的执行信息,利用怀疑度公式来计算程序各个代码实体的出错概率,得到相应被测程序的怀疑度表(Rank List)[2]。在怀疑度表中,怀疑度值越高的语句排名越靠前,那么其它出错的概率越大。测试人员按怀疑度表的顺序检查语句时,如果发现实际出错的语句排名越靠前,则相应的怀疑度计算公式的错误定位精确度会越高。目前研究比较多的公式有Tarantula[3],Ochiai[4]和Op2[5]等。为了进一步提高错误定位的准确性,Papadakis等人[6-8]提出了基于变异分析的错误定位方法(Mutation-based Fault Localization, MBFL),即采用变异算子(Mutation Operator)在被测程序中植入若干错误从而形成变异体(Mutant),并统计变异体被测试用例检测出或未检测出的执行结果信息,利用上述怀疑度公式计算各个变异体的怀疑度值,并将其语句变异体中的最大怀疑度值作为此语句的怀疑度值。Moon[9]结合5个程序的14个错误版本,论证了他们提出的MBFL技术(MUSE)比如今最先进的SBFL技术(Op2)的精确度平均提高了25倍。

相较于传统的SBFL方法,MBFL具有更高的错误定位精度,但与变异测试类似,MBFL要采用变异算子生成大量的变异体,并在每个变异体上执行全部测试用例,开销巨大,从而影响MBFL在软件错误定位中的实际应用。近些年,对MBFL的效率优化,主要从变异算子的选择、变异体的抽样及变异体的执行3个方面展开研究[10]。Papadakis等人从变异体的抽样及变异算子的选择的角度出发,提出了变异体抽样

到稿日期:2016-10-10 返修日期:2016-12-15 本文受国家自然科学基金(61672085,61472025),教育部新世纪优秀人才计划项目(NCET-12-0757)资助。

王林鑫(1991－),男,硕士,主要研究方向为软件测试;**王微微**(1990－),女,博士,主要研究方向为Web应用测试;**赵瑞莲**(1964－),女,教授,博士生导师,主要研究方向为软件测试及依赖性分析,E-mail:rlzhao@mail.buct.edu.cn;**李 征**(1974－),男,教授,博士生导师,主要研究方向为模型分析及测试、基于搜索的软件工程。

策略[7]和选择变异策略[8]；另外，也有研究从优化变异体的执行的角度出发，Zhang[11]提出了测试用例预优化及测试用例约减策略，Gong[12]提出了动态变异执行策略 DMES。

选择变异策略(Selective Mutation Strategy，SELECTIVE)从变异算子的选择角度进行约减，仅选取高“充分度”变异算子生成变异体，但缺失的变异算子可导致生成的变异体种类缺失，进而降低错误定位精度。变异体抽样策略(Mutant Sampling Strategy，SAMPLING)采用变异算子的全集对被测程序生成变异体，在所有生成的变异体中以一定的概率分布进行随机抽取，从而达到约减的目的。SAMPLING 尽管使用了全集的变异算子，但忽略了每条语句生成的变异体数目各不相同，整体采样的方法不能保证每条语句中都有变异体被采样，导致部分语句的怀疑度不准确，从而影响最终的错误定位效果。本文在不降低 MBFL 方法错误定位精度的前提下，围绕变异算子的选择及变异体的抽样，提出面向语句的变异体约减策略(Statement-Oriented Mutant rEduction Strategy，SOME)。基于 SOME 的 MBFL 方法，在变异算子的选择过程中采用全集的变异算子对被失败测试用例覆盖的语句进行变异，保证变异体种类的完备性；另外，在变异体的抽样过程中，按一定比例对每条被覆盖语句上同种变异算子生成的变异体进行抽样，保证每条语句都有变异体覆盖，同时降低被覆盖语句的变异执行开销，提高 MBFL 方法的执行效率。

本文选取西门子测试集(Siemens suite)[3-9] 7 个程序包中的 112 个错误版本程序作为被测程序，对基于 SOME 策略和基于 SAMPLING 的 MBFL 的错误定位效果及变异执行开销进行对比。实验结果表明，在相同变异体执行开销范围内，基于 SOME 的 MBFL 比基于 SAMPLING 的 MBFL 有更高的错误定位精度；与原始的 MBFL 相比，SOME 在保证较高错误定位精度的同时，能够有效减少相应的变异执行开销。

2 MBFL 及变异约减

2.1 基于变异分析的错误定位方法(MBFL)

MBFL 采用变异算子对被测程序的所有可执行语句进行变异，并用变异体的怀疑度值来衡量每条语句的怀疑度值，主要基于以下两种观察现象。

(1)针对被测程序 P，一个执行结果为失败(failed)的测试用例 T_f 在含故障语句生成的变异体 m_f 上执行通过(passed)的可能性，要比在不含故障语句生成的变异体 m_c 上更大。此现象的原因在于，一个带故障的程序 P 可能通过变异含故障的语句而被部分自动修复，从而导致在 m_f 上从执行失败转变为执行通过的测试用例数目要比 m_c 多。

(2)针对被测程序 P，一个执行结果为通过(passed)的测试用例 T_p 在不含故障语句生成的变异体 m_c 上执行失败(failed)的可能性，要比在含故障语句生成的变异体 m_f 上更大。通过变异不含故障的语句，使正确的语句产生错误，从而导致在 m_c 上从执行通过转变为执行失败的测试用例数目要比 m_f 多。

在 MBFL 中，用来衡量变异体怀疑度变异体与被测程序行为的相似程度。假设 S_{failed} 为被测程序 P 中失败测试用例 T_f 覆盖的语句集合，且 $S_i \in S_{failed}$ $(i=1,2,3,\cdots,n)$，那么在语句 S_i 上通过采用变异算子植入故障来生成变异体，其相应集合记为 M_{si}。对于变异体 $m \in M_{si}$，杀死(Killed)m 的测试用例集记为 T_k，未杀死(Not Killed)m 的测试用例集记为 T_n，则变异体 m 的怀疑度 $Sus(m)$ 与集合 T_p 中未杀死变异体的测试用例数 a_{np}、T_p 中杀死变异体的测试用例数 a_{kp}、T_f 中未杀死变异体的测试用例数 a_{nf} 和 T_f 中杀死变异体的测试用例数 a_{kf} 有关，可表示为 $Sus(m)=Sus(a_{np},a_{nf},a_{kp},a_{kf})$。比较流行的怀疑度公式有：

$$Tanratula=\frac{\frac{a_{kf}}{a_{kf}+a_{nf}}}{\frac{a_{kf}}{a_{kf}+a_{nf}}+\frac{a_{kp}}{a_{kp}+a_{np}}} \tag{1}$$

$$Ochiai=\frac{a_{kf}}{\sqrt{(a_{kf}+a_{nf})(a_{kf}+a_{kp})}} \tag{2}$$

$$Op2=a_{kf}-\frac{a_{kp}}{a_{kp}+a_{np}+1} \tag{3}$$

用语句怀疑度来度量语句可能出现错误的概率。语句 S_i 的怀疑度 $Sus(S_i)$ 定义为 S_i 的变异体集合 M_{si} 中变异体怀疑度的最大值，即 $Sus(S_i)=\mathrm{Max}(Sus(m_1),Sus(m_2),\cdots,Sus(m_n))$，$m_1,m_2,\cdots,m_n \in M_{si}$，$i=1,2,3,\cdots,n$。确定语句 S_i 的怀疑度 $Sus(S_i)$ 之后，将其按数值从大到小排列，得到被测程序 P 的怀疑度表(Rank list)。

2.2 MBFL 约减策略

MBFL 的错误定位精确度高，但执行开销巨大，这是公认的要解决的问题。

2013 年，Papadakis 等人[7]提出了变异体抽样策略(Mutant Sampling Strategy，SAMPLING)，该策略从被测程序生成的所有变异体角度出发，在所有被 T_f 覆盖的语句生成的变异体集合中，以一定的概率分布随机抽样得到变异体，并计算其怀疑度。他们选择西门子测试集(Siemens suite)作为被测程序集，验证了在采用变异体抽样策略约减得到的变异体集合上，应用 MBFL 方法相较于 SBFL 方法有较高的错误定位精度，且当随机抽取 20% 的变异体时 MBFL 的效果较好。但是其随机抽样的方法没有从单条被 T_f 覆盖的语句的角度考虑，不能保证所有被覆盖的语句都有变异体覆盖。

2014 年，Papadakis 等人[8]又提出了选择变异策略(Selective Mutation Strategy，SELECTIVE)。该方法提出了特定的变异算子充分性准则，分析了不同变异算子对 MBFL 贡献程度的差异，从变异算子选择的角度考虑，保留少部分“充分度”较高的变异算子来生成相应的变异体。然而，不同变异算子对语句植入故障的能力不同，约减变异算子的种类会让部分语句的变异体怀疑度值产生显著变化，从而不能明显区分怀疑度表中的实际出错语句和正确语句。研究表明，选择变异策略 SELECTIVE 的效果不及基于随机选择的变异体抽样策略 SAMPLING 的效果好[13]。

综上所述，变异体抽样策略从被 T_f 覆盖的语句生成的变异体集合角度出发，按一定比例进行随机抽样，但忽略了每条语句生成变异体的情况不同；而选择变异策略则是挑选部分变异算子生成变异体，忽略了全集变异算子种类的完备性。

针对 MBFL，本文提出面向语句的变异体约减策略(Statement-Oriented Mutant rEduction Strategy，SOME)，采用全集变异算子，对所有被失败测试用例覆盖的语句进行变异，并逐条对其生成的变异体进行一定比例的抽样，减少后续

变异执行开销，提高错误定位的效率。

3　基于面向语句变异体约减的 MBFL

本节主要研究基于面向语句的 MBFL 变异体约减策略 SOME 和相应的 MBFL 方法框架。

3.1　SOME 方法概述

在 MBFL 中，不同变异算子对不同语句的植入故障能力不同。在前期的研究中发现，针对被测程序 P，对同一条语句用同一种变异算子、同一测试用例集和同一怀疑度公式生成的变异体中，30%～80%的变异体具有相同的怀疑度值。30%～80%的范围虽然比较大，但这也说明针对同一条语句，同类变异算子生成的变异体是可以约减的，而且此约减方式对整条语句的怀疑度计算结果的影响较小，能够保证错误定位的精确度。

SOME 的实现主要包含两个阶段：变异生成阶段和变异选择阶段，具体算法如算法 1 所示。

算法 1　Statement-Oriented Mutant rEduction Strategy

Input：被测程序 P，测试用例集 T，T 在 P 的执行结果向量 R 及覆盖信息向量 Cov，被失败测试用例 T_f 覆盖的语句 $S_i \in S_{failed}$(i=1，2，3，…，n)，变异算子 $Op_j \in Set\_TotalOp_k$(j=1，2，3，…，k；k 为变异算子种类数目)

Output：被测程序 P 约减后的变异体集合 Total_Reduction_Set_Mutant

```
1. //变异生成阶段
2. Sfailed←Cov(P,Tf,R)
3. //筛选被失败测试用例覆盖的语句 Sfailed
4. for each Si∈ Sfailed
5.     for each Opj∈Set_TotalOpk
6.         if Si← Success(Si,Opj)
7. //如果 Opj 适用于语句 Si，即 Opj 可以 Si 上找到相应变异位置
8.         Set_Mutant_Opj_Si← Muta_gen(Opj,Si)
9. //Opj 在 Si 上生成相应变异体，得到集合 Set_Mutant_Opj_Si
10.        else
11.        Set_Mutant_Opj_Si← NULL
12.        end if
13.    end for
14. end for
15. for each Si//变异选择阶段
16. Set_Mutant_Opj← Classify(Set_Mutant_Opj_Si,Si)
17. //对同种变异算子生成的变异体进行归类，其中 Set_Mutant_Opj_Si 不为 NULL
18. Reduction_Set_Mutant_Si← Muta_select(X%,Set_Mutant_Opj)
    X←10
19. //按照 10%的比例进行变异选择
20. end for
21. Total_Reduction_Set_Mutant=∑(i=1,n) Reduction_Set_Mutant_Si
22. return Total_Reduction_Set_Mutant
23. //被测程序 P 约减后的变异体集合 Total_Reduction_Set_Mutant
```

3.1.1　变异生成阶段

本节主要描述基于 SOME 的 MBFL 方法的变异生成阶段，即使用全集的变异算子对被失败测试用例覆盖的语句进行变异体的生成。

(1)针对被测程序 P，执行测试用例集 T，每个测试用例在被测程序 P 上依次运行，得到相应的测试执行结果向量 R 及覆盖信息向量 Cov，将执行结果为 passed 的测试用例记作 T_p，将执行结果为 failed 的测试用例记作 T_f；根据覆盖信息向量 Cov 筛选出被 T_f 覆盖的语句集合 S_{failed} 以及被测试用例 T_p 成功覆盖的语句集合 S_{passed}，所有被测试用例集 T 覆盖的语句集合 $S=S_{failed}+S_{passed}$。

(2)从集合 S_{failed} 中选取语句 S_i 作为候选变异算子集合 $Set\_TotalOp_k$ 的变异对象，其中 $Set\_TotalOp_k$ 表示种类完备的候选变异算子集合，共 k 种变异算子。

(3)针对语句 S_i，从 $Set\_TotalOp_k$ 中选取变异算子 Op_j，其中 j 的取值范围为 1～k，判断 Op_j 是否可以在 S_i 中找到变异点，如果能找到变异点，则 Op_j 在 S_i 上生成相应的变异体(一阶变异体)，这些变异体集合记作 $Set\_Mutant\_Op_j\_S_i$；如果 Op_j 在 S_i 中不能找到变异点，则选取下一个变异算子 Op_{j+1} 进行判断，直到所有变异算子都已经判断完毕，回到步骤(2)，选取集合 S_{failed} 中的下一条语句 S_{i+1}。

(4)当集合 S_{failed} 中所有语句都被可适用的变异算子变异，S_{failed} 上每条语句 S_i 均生成对应的变异体集合 $Set\_Mutant\_Op_j\_S_i$ 时，SOME 的变异生成阶段结束。

3.1.2　变异选择阶段

在得到语句 S_i 相应的变异体集合 $Set\_Mutant\_Op_j\_S_i$ 后，SOME 由变异生成阶段转入变异选择阶段，即逐条地对 T_f 覆盖的语句上生成的变异体进行抽样。

(1)根据语句 S_i 相应的变异体集合 $Set\_Mutant\_Op_j\_S_i$，对同种变异算子生成的变异体进行归类，不同种类变异算子生成的变异体集合可记作：$Set\_Mutant\_Op_1$，$Set\_Mutant\_Op_2$，$Set\_Mutant\_Op_3$，…，$Set\_Mutant\_Op_j$。

(2)针对同种变异算子 Op_j 生成的变异体集合 $Set\_Mutant\_Op_j$，随机选取其中 X%的变异体加入集合 $Reduction\_Set\_Mutant\_S_i$ 中。根据研究观察，本文将 X%的取值定为 10%，但如果按照 10%的比例进行选择之后结果小于整数 1，即没有 1 个变异体被选择出来，那么则将 X%的值每次增加 10%(X%≤100%)，直至在 $Set\_Mutant\_Op_j$ 中可以选取 1 个变异体加入集合 $Reduction\_Set\_Mutant\_S_i$ 中。

(3)依次选取语句 S_i 中 Op_j 生成的变异体集合 $Set\_Mutant\_Op_j$，重复步骤(2)，直至该语句的所有变异体集合均被选择完毕后，跳转至步骤(1)进行下一语句 S_{i+1} 的变异选择阶段。当集合 S_{failed} 中所有语句均得到相应的 $Reduction\_Set\_Mutant\_S_i$ 后，SOME 的变异选择阶段结束。

当 SOME 的变异生成阶段和变异选择阶段都相继结束后，最终由 S_{failed} 中每个被 T_f 覆盖的语句 S_i 得到的 $Reduction\_Set\_Mutant\_S_i$ 组成的集合 $Total\_Reduction\_Set\_Mutant$ 则是被测程序 P 运用 SOME 进行约减后的变异体集合。

3.2　基于 SOME 的 MBFL 框架

本文针对 MBFL 的变异体约减策略 SOME，利用测试用例的执行信息，逐条对被失败测试用例覆盖的语句生成的变异体进行约减，以减少 MBFL 的变异执行开销。基于 SOME 的 MBFL 基本框架如图 1 所示。

基于 SOME 的 MBFL 基本框架主要包含 4 个阶段：测试执行阶段、变异生成阶段、变异选择阶段以及怀疑度表生成阶段。

首先,在测试执行阶段对被测程序执行相应的测试用例集,搜集被测程序语句的覆盖信息及测试用例的执行结果,并将其提供给变异生成阶段。其次,在变异生成阶段利用测试执行阶段得到的信息筛选出被失败的测试用例覆盖的语句,并对这些语句植入人工故障从而生成变异体,将构成的变异体集合提供给下一阶段进行变异选择。再次,在变异选择阶段,针对每一条被失败测试用例覆盖的语句应用语句级别上的变异体约减策略。最后,对变异体执行测试用例,并计算语句上变异体怀疑度的最大值,即语句怀疑度,同时对被测程序的所有语句按其怀疑度值从大到小排序(未被失败的测试用例覆盖的语句的怀疑度为 0),得到被测程序的怀疑度表,用于辅助开发人员定位程序中出错语句的具体位置。

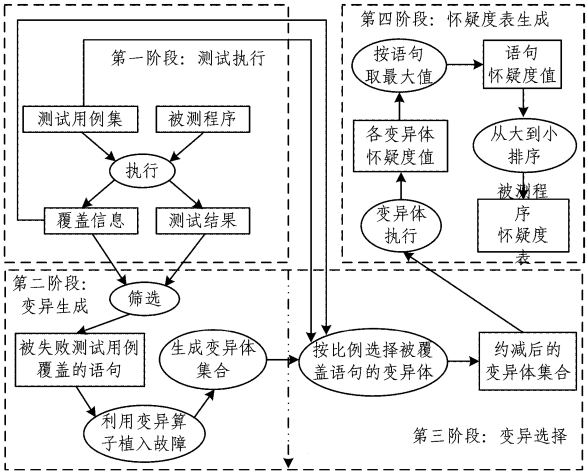

图 1 基于 SOME 的 MBFL 框架

4 实验及结果分析

为了验证面向语句的变异体约减策略 SOME 的有效性,本文提出如下 3 个研究问题,并通过实验与结果对其进行分析回答。

RQ1:相较于基于变异体抽样策略的 MBFL,基于 SOME 的 MBFL 是否具有更高的错误定位精度?

RQ2:基于 SOME 的 MBFL 是否依然能保证原始 MBFL 方法的错误定位精度?

RQ3:相较于原始 MBFL 方法,基于 SOME 的 MBFL 能减少多少变异执行开销?其与基于变异体抽样策略的 MBFL 的变异执行开销的对比情况如何?

4.1 实验设计

本实验使用 Gcov 工具在 Linux 环境下获取相关测试用例的覆盖信息。为获取被测程序中每条语句相应的变异体,采用了针对 C 语言的变异测试工具 Proteum/IM2.0[14-15]。同样在 Linux 环境下,使用 GCC 编译被测程序和相应的变异体。结合其本身的 shell 脚本和命令行组件,对其配置进行适当的修改并重新编译。该工具针对每个被测程序提供了完整的单元级变异算子集合。此版本的 Proteum 支持 6 组共 108 种变异算子,其中包含 4 组传统的变异算子:常量组(Constant)、变量组(Variable)、操作符组(Operator)和语句组(Statement),以及 2 组面向接口的变异算子(GroupI 和 GroupII)。

本实验选择西门子测试集作为实验基准,这些基准程序包含的测试用例集和错误版本等相关信息均可以从软件基础程序库(Software-artifact Infrastructure Repository,SIR)下载[16]。基准程序的相关信息如表 1 所列,7 个程序包中共包含 132 个错误版本,实验最终排除了 20 个错误版本:部分版本由于语句缺失或其程序包中所提供的测试用例集无法检测出故障、版本改动并未引发错误等原因而未被选为实验评估的被测程序;其他错误版本程序因执行过程中发生段错误而导致无法搜集完整的必要执行信息,或因出错的语句过多而导致评估困难,也未被选为实验评估的被测程序。本文最终以 112 个错误版本作为实验评估的被测程序。

表 1 基准程序的相关信息

基准程序名	错误版本数	使用版本数	排除版本	平均代码行数	测试用例数	变异体数
Schedule	9	5	v1,v5,v6,v9	296	2650	2203～2247
Schedule2	10	8	v4,v9	263	2710	2845～2980
Tcas	41	41	—	137	1608	4937～5323
Totinfo	23	23	—	273	1052	6314～6438
Printtokens	7	3	v1,v2,v4,v6	342	4130	4235～4263
Printtokens2	10	6	v1,v2,v3,v10	355	4115	10131～10162
Replace	32	26	v15,v17,v19,v25,v27,v32	513	5542	10661～10988

4.2 实验评估准则

在错误定位的过程中,用 Score 评价指标来评估对应的错误定位方法的优劣。Score 根据怀疑度表中的语句顺序(怀疑度从大到小排序)衡量顺序检查语句至定位实际错误语句的人工开销,其计算公式如下:

$$Score=\frac{Rank}{\text{Total executed statements}} \tag{4}$$

其中,Rank 指找到出错语句之前检查的语句数,即出错语句在怀疑度表中的排名,Score 数值越小,错误定位方法的定位精确度越高,故本文将其作为错误定位精确程度的评价指标。Rank 有两种特殊情况:若出错语句与未出错语句具有相同怀疑度且首先检测到出错语句,则此时的 Rank 为最优的 Best;若与出错语句怀疑度相等的所有语句被检查之后才检测到出错语句,则此时的 Rank 为最差的 Worst,其计算公式如下:

$$Rank=\frac{Best+Worst}{2} \tag{5}$$

在基于变异分析的错误定位中,变异执行开销可以用变异体-测试对(Mutant-Test Pair,MTP)的执行次数进行评估,并且评估的准确度不受具体实现与使用环境的影响[11]。

4.3 实验结果与分析

针对 RQ1 和 RQ2,以表 1 中的 112 个错误版本程序作为被测程序,主要使用在 SBFL 和 MBFL 中被广泛使用的怀疑度计算公式 Op2[5,17]和 Ochiai[4,12]进一步检验面向语句的变异体约减策略 SOME 是否对 MBFL 的错误定位精确度产生影响,并对基于 SOME 以及基于 SAMPLING 的 MBFL 的错

误定位精确度进行比较分析,结果如表 2 所列。表 2 中的每一行代表按怀疑度表检查第一列所示比例的代码时检查到错误语句的被测程序占 112 个被测错误版本程序的比例;TOTAL 表示未采用任何变异体约减策略的原始 MBFL,SAMPLING-10%和 SAMPLING-20%分别表示基于变异体抽样策略随机选取 10%和 20%变异体的 MBFL,SOME 表示基于面向语句的 MBFL 变异体约减策略;表 2 相应列表对应以上 4 种方法在不同怀疑度公式下的错误定位效果,例如第一行的第二列表示检查到 1%的代码时,未应用任何变异体约减策略且保留全集的原始 MBFL 方法 TOTAL 使用怀疑度计算公式 Op2 时,检查到错误语句的被测程序占总被测程序的 38.39%。

表 2　基于 SOME 和 SAMPLING 的 MBFL 错误定位精确度对比/%

代码检查百分比	TOTAL		SAMPLING-10%		SAMPLING-20%		SOME-10%	
	Op2	Ochiai	Op2	Ochiai	Op2	Ochiai	Op2	Ochiai
1	38.39	35.71	22.32	22.32	20.54	24.11	33.93	34.82
5	75.89	79.46	46.43	47.32	58.93	58.04	77.68	77.68
10	87.50	85.71	62.50	61.61	69.64	66.96	87.50	84.82
15	91.07	91.07	69.64	71.43	76.79	81.25	89.29	88.39
20	91.96	93.75	75.89	78.57	78.57	85.71	92.86	91.07
30	96.43	97.32	83.04	85.71	84.82	87.50	97.32	95.54
40	97.32	97.32	88.39	90.18	91.07	92.86	98.21	99.11
50	98.21	99.11	93.75	95.54	93.75	93.75	98.21	99.11
60	100.00	100.00	98.21	98.21	97.32	97.32	100.00	100.00
70	100.00	100.00	99.11	99.11	98.21	98.21	100.00	100.00
80	100.00	100.00	99.11	99.11	100.00	100.00	100.00	100.00
90	100.00	100.00	99.11	99.11	100.00	100.00	100.00	100.00
100	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00

从表 2 可以看出,基于 SOME 的 MBFL 方法在不同的怀疑度计算公式下均比基于 SAMPLING 的 MBFL 方法的错误定位精确度更高;从表 2 的最后 2 列可以发现,基于 SOME 的 MBFL 方法运用公式 Op2 比运用公式 Ochiai 计算所得的错误定位效果更好;另外,通过变异体抽样策略 SAMPLING-10%和 SAMPLING-20%的精确度对比可以发现,随机选取 20%变异体的方法要比随机选取 10%变异体的方法在 MBFL 上的定位效果更好。

另一方面,从表 2 可以看出,基于 SOME 的 MBFL 在使用怀疑度计算公式 Op2 后,在代码检查百分比为 5%,20%,30%以及 40%时,均比原始的 MBFL 的错误定位效果更好。进一步分析原因可知,本实验在计算 Rank 值时,应用 SOME 的 MBFL 中 Best 的取值与 TOTAL 相同,但是在对 Worst 取值时,SOME 能够有效区分真正出错语句与正确语句的怀疑度值,使真正出错语句的 Worst 值比 TOTAL 的 Worst 值更小,这就导致在取平均数时,前者的 Rank 值比后者小,故前者在怀疑度表中排名靠前。

上述结论同样可以从图 2 中得到验证。

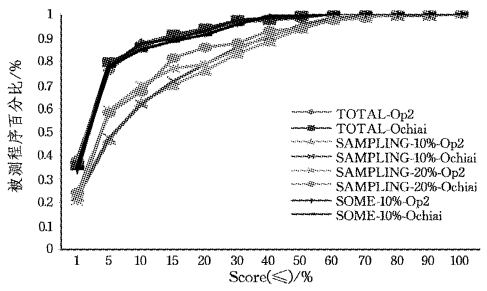

图 2　基于 SOME 和 SAMPLING 的 MBFL 错误定位精确度折线图

图 2 的横轴 x 为 Score 值,纵轴 y 为被测程序百分比,任意点(x,y)表示 Score 值小于 x 的被测程序占 112 个被测程序的比例为 y,上升速度越快的折线对应的错误定位方法错误的定位效果越好。因此,从图 2 中可以看出,基于 SOME 的 MBFL 方法对应 2 个公式的折线都位于 SAMPLING 方法的上方,故基于 SOME 的 MBFL 使用任意一个怀疑度公式的错误定位效果均优于 SAMPLING 方法。另外,从图 2 可以看出,基于 SOME 的 MBFL 在错误定位效果方面与原始的 MBFL 方法 TOTAL 基本保持一致。同时,我们进行曼-惠特尼 U 检验,假设基于 SOME 的 MBFL 在错误定位效果方面与原始的 MBFL 方法 TOTAL 无明显差异,在显著性水平为 0.01 的前提下对其进行相关显著性分析,得到的 p-value 值为 0.916,即保留原假设。

综上所述,对于 RQ1 和 RQ2,基于 SOME 的 MBFL 与原始 MBFL 方法均能保持较高的错误定位精度,且基于 SOME 的 MBFL 相较于基于变异体抽样策略随机抽取 20%变异体的 MBFL 具有更高的错误定位精度。

针对 RQ3,为了分析基于 SOME 的 MBFL 方法所需的变异执行开销与原始 MBFL 的变异执行开销之间的差距,并观察应用变异体抽样策略 SAMPLING 和应用 SOME 策略前后的 MBFL 变异执行开销情况,本实验同样以 7 个程序包的 112 个错误版本为被测程序,统计基于变异体抽样策略的 MBFL 以及基于 SOME 的 MBFL 所需的 MTP 执行次数,比较应用以上两种变异体约减策略的 MBFL 所需的 MTP 执行次数。本实验的变异执行开销受随机初始变异体和测试用例执行顺序的影响,故对以上两种 MBFL 方法均重复 10 次实验,结果如表 3 所列。

表 3 给出了在 7 个程序包的 112 个错误版本上,基于 SOME 以及基于 SAMPLING 的 MBFL 相较于原始的 MBFL 方法 TOTAL 减少的 MTP 执行次数百分比的最小值(Min)、平均值(Average)和最大值(Max)。从中可知,基于 SOME 的 MBFL 相较于 TOTAL 平均减少的 MTP 执行次数百分比即平均约减率如下:Schedule 为 63.88%;Schedule2 为 78.97%;Tcas 为 80.84%;Totinfo 为 80.09%;Printtokens 为 70.67%;

Printtokens2为 69.61%；Replace 为 82.22%。通过进一步分析表 3 最后一行的整体平均值得到，基于 SOME 的 MBFL 可以平均减少 75.18%的 MTP 执行次数，略低于基于 SAMPLING-20%的 MBFL 的平均约减率 78.12%；与此同时，基于 SOME 的 MBFL 最多可以约减 79.98%的 MTP 执行次数，高于基于 SAMPLING-20%的 MBFL 的最大约减率 79.89%。

表 3 应用 SOME 和 SAMPLING 的 MBFL 减少的 MTP 执行次数百分比/%

基准程序名	SOME-10%			SAMPLING-20%			SAMPLING-10%		
	Min	Average	Max	Min	Average	Max	Min	Average	Max
Schedule	60.49	63.88	79.05	78.49	80.03	80.82	88.96	90.04	91.62
Schedule2	78.19	78.97	79.39	78.98	79.34	80.78	88.79	89.28	89.92
Tcas	80.23	80.84	81.02	80.03	80.24	80.29	89.89	90.07	90.23
Totinfo	79.51	80.09	83.55	71.57	73.34	74.68	83.01	86.54	88.50
Printtokens	68.11	70.67	80.87	78.74	80.27	82.33	89.92	90.01	90.85
Printtokens2	67.35	69.61	72.64	70.23	71.66	78.41	79.67	86.59	89.01
Replace	80.68	82.22	83.37	80.71	81.94	81.95	89.22	90.91	91.34
整体平均值	73.51	75.18	79.98	76.96	78.12	79.89	87.07	89.06	90.21

综上所述，基于 SOME 的 MBFL 相较于原始 MBFL 方法平均减少了 75.18%的 MTP 执行次数，这充分说明 SOME 能够有效减少 MBFL 的变异执行开销。此外，基于 SOME 的 MBFL 在错误定位精确度上保持明显优势的同时，其变异约减效果与基于 SAMPLING-20%的 MBFL 变异约减效果基本保持一致，且明显优于基于 SAMPLING-10%的 MBFL。

结束语 本文提出了一种面向语句的 MBFL 变异体约减策略。该策略使用全集变异算子对所有被失败测试用例覆盖的语句进行变异，从语句级别上逐条对其生成的变异体进行一定比例的抽样，从而减少后续变异体的执行开销。通过对 7 个程序包的 112 个被测错误版本程序进行实验分析，一方面验证了基于 SOME 的 MBFL 在保持较高错误定位精度的同时能够有效减少变异-测试对的执行开销；另一方面，验证了基于 SOME 的 MBFL 与基于 SAMPLING 的 MBFL 在变异执行开销基本保持一致（即两者变异体约减程度保持一致）的同时，前者的错误定位效果比后者明显更好。

此外，本文主要从变异体数量约减方面进行研究，后续可以从优化变异执行方面进行考虑，将二者结合，尽量避免变异体执行过程中不必要的执行开销，进一步优化 MBFL 的执行效率。

参 考 文 献

[1] YU K, LIN M X. Advances in Automatic Software Fault Localization Techniques[J]. Chinese Journal on Computers, 2011, 24(8): 1411-1422. (in Chinese)
虞凯，林梦香. 自动化软件错误定位技术研究进展[J]. 计算机学报，2011，34(8)：1411-1422.

[2] WONG W E, DEBROY V. A survey of software fault localization[OL]. http://core.ac.uk/display/21888435.

[3] JONES J A, HARROLD M J. Empirical evaluation of the tarantula automatic fault-localization technique[C]// Proceedings of the 20th IEEE/ACM international Conference on Automated Software Engineering. ACM, 2005: 273-282.

[4] ABREU R, ZOETEWEIJ P, VAN GEMUND A J C. An evaluation of similarity coefficients for software fault localization[C]// 12th Pacific Rim International Symposium on Dependable Computing, 2006(PRDC'06). IEEE, 2006: 39-46.

[5] NAISH L, LEE H J, RAMAMOHANARAO K. A model for spectra-based software diagnosis[J]. ACM Transactions on Software Engineering and Methodology (TOSEM), 2011, 20(3): 1-32.

[6] PAPADAKIS M, LE T Y. Using mutants to locate "Unknown" faults[C]// 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation (ICST). IEEE, 2012: 691-700.

[7] PAPADAKIS M, LE TRAON Y. Metallaxis-FL: Mutation-based Fault Localization[J]. Software Testing, Verification and Reliability, 2015, 25(5-7): 605-628.

[8] PAPADAKIS M, LE TRAON Y. Effective Fault Localization via Mutation Analysis: A Selective Mutation Approach[C]// Proceedings of the 29th Annual ACM Symposium on Applied Computing(SAC). 2014: 1293-1300.

[9] MOON S, KIM Y, KIM M. Ask the mutants: Mutating faulty programs for fault localization[C]// 2014 IEEE Seventh International Conference on Software Testing, Verification and Validation (ICST). IEEE, 2014: 153-162.

[10] JIA Y, HARMAN M. An analysis and survey of the development of mutation testing[J]. Software Engineering, 2011, 37(5): 649-678.

[11] ZHANG L, MARINOV D, KHURSHID S. Faster mutation testing inspired by test prioritization and reduction[C]// Proceedings of the 2013 International Symposium on Software Testing and Analysis. ACM, 2013: 235-245.

[12] GONG P, ZHAO R, LI Z. Faster mutation-based fault localization with a novel mutation execution strategy[C]// 2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW). IEEE, 2015: 1-10.

[13] ZHANG L, HOU S S, HU J J, et al. Is operator-based mutant selection superior to random mutant selection? [C]// Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering. ACM, 2010: 435-444.

[14] DELAMARO M E, MALDONADO J C, MATHUR A P. Proteum-A Tool for the Assessment of Test Adequacy for C Programs User's guide[C]// PCS. 1999: 79-95.

[15] MALDONADO J C, VINCENZI A M R, DELAMARO M E. Proteum/IM 2.0: An Integrated Mutation Testing Environment [C]// Mutation 2000 Symposium. San Jose, CA: Kluwer Academic Publishers, 2000: 91-101.

[16] 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.

[17] XIE X, CHEN T Y, KUO F C, et al. A theoretical analysis of the risk evaluation formulas for spectrum-based fault localization[J]. ACM Transactions on Software Engineering and Methodology (TOSEM), 2013, 22(4): 31.