针对基于变异错误定位的一种动态变异执行策略

龚 沛[1] 耿楚瑶[2] 郭俊霞[1] 赵瑞莲[1]
(北京化工大学计算机系 北京 100029)[1] (北京邮电大学信通学院 北京 100876)[2]

摘 要 在软件调试过程中,如何快速、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。基于变异的错误定位方法是一种通过分析被测程序与程序变异体之间的行为相似性来估计语句出错概率、进行错误定位的方法。该方法有较高的错误定位精确度,但由于需对大量程序变异体执行测试用例集,因此其变异执行开销较大。为此提出了一种动态变异执行策略,它通过搜集测试用例执行信息,动态地调整变异体及测试用例的执行顺序,以减少其变异执行开销。实验结果表明,在6个程序包的127个错误版本上,应用提出的动态变异执行策略可在保证错误定位精确度的前提下,减少23%~78%的变异执行开销,显著提高了基于变异的错误定位方法的效率。

关键词 错误定位,变异分析,变异执行策略
中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2016.2.043

Dynamic Mutation Execution Strategy for Mutation-based Fault Localization

GONG Pei[1] GENG Chu-yao[2] GUO Jun-xia[1] ZHAO Rui-lian[1]
(Department of Computer Science, Beijing University of Chemical Technology, Beijing 100029, China)[1]
(School of Information and Communication Engineering, Beijing University of Posts and Telecommunications, Beijing 100876, China)[2]

Abstract During software debugging process, how to locate faults in programs quickly and accurately is an issue of common concern to developers. Mutation-based fault localization is an approach by estimating the possibilities of statements that incur error to locate faults on the basis of the similarity between the programs under test and corresponding mutants. This approach shows a high precision on fault localization but requires a large execution cost, since it needs to execute the test suite on a lot of mutants. For reducing unnecessary execution cost, this paper presented a dynamic mutation execution strategy, which dynamically adjusts the execution orders of both mutants and test cases according to previous execution information. Empirical studies were conducted on 127 faulty versions from 6 program packages. The results indicate that this strategy can reduce 23% up to 78% mutation execution cost under the case of keeping fault location precision.

Keywords Fault localization, Mutation analysis, Mutation execution strategy

1 引言

出于软件质量保证的需要,基于覆盖的错误定位技术(Coverage-based Fault Localization, CBFL)受到了学术界的广泛关注。CBFL通过搜集测试用例在被测程序上的覆盖信息以及通过或失败信息,利用怀疑度公式计算程序各语句的怀疑度值,得到被测程序的怀疑度表[1,2]。怀疑度值越高的语句意味着该语句出错的可能性越大,在怀疑度表中的位置就越靠前。按照怀疑度表的顺序检查语句,若发现错误的语句越靠前则说明错误定位的精确度越高。因此,如何准确地得到语句的怀疑度是CBFL技术的关键。近年来,对于CBFL技术的研究主要集中在如何给出一种有效的怀疑度计算公式,目前常用的怀疑度计算公式有Tarantula[3]、Ochiai[4]和Op2[5]等。程序覆盖信息的搜集粒度主要包括语句[2-5]和分支[6,7]等。最新研究表明,将变异分析技术与CBFL相结合可以获得更好的错误定位效果[8-10]。变异分析是一种基于故障植入的测试技术,首先对被测程序语句植入模拟实际故障的人工故障,这些带有故障的程序被称为变异体。在变异体上执行测试用例,若一个测试用例在被测程序与变异体上的执行结果不同,则称变异体被该测试用例杀死(Killed);反之则称变异体未被该测试用例杀死(Not Killed)。然后根据变异体是否被测试用例杀死的信息,对测试用例和被测程序进行分析[10]。

2012年,Papadakis和Le Traon提出了一种基于变异的错误定位方法(Mutation-based Fault Localization, MBFL)[8]。MBFL对程序的每个语句生成若干变异体并根据变异体是否

到稿日期:2014-12-26 返修日期:2015-04-21 本文受国家自然科学基金项目(61170082,61472025),教育部新世纪优秀人才支持计划(NCET-12-0757)资助。

龚 沛(1990-),男,硕士生,主要研究方向为变异测试与错误定位,E-mail:gongpei_medo@126.com;耿楚瑶(1994-),女,主要研究方向为通信工程与网络信息抽取,E-mail:alizalovelofe@hotmail.com;郭俊霞(1977-),女,博士,讲师,主要研究方向为网络信息定向抽取技术及测试,E-mail:gjxia@mail.buct.edu.cn;赵瑞莲(1964-),女,博士,教授,CCF会员,主要研究方向为软件测试与软件可靠性,E-mail:rlzhao@mail.buct.edu.cn。

被测试用例杀死的信息，利用怀疑度公式计算各变异体的怀疑度，每个语句以其变异体怀疑度的最大值作为该语句的怀疑度[9]，得到被测程序的怀疑度表，据此进行错误定位分析。这一过程导致了庞大的变异执行开销，影响了 MBFL 方法在实际软件开发过程中的应用。目前，降低变异执行开销的技术主要有选择变异、变异体抽样[9-11]和应用高效的执行策略[12]等。选择变异和变异体抽样通过减少变异体的数量来减少变异执行开销。有研究表明，这两种技术在减少变异执行开销的同时，一定程度上也降低了 MBFL 的错误定位准确度，并且选择变异的方式与变异体抽样的概率依赖于经验，存在着随机性与不稳定性[9,10]。Zhang[12]等人受测试用例预优化及测试用例约减技术的启发，提出了一种针对变异测试的变异执行策略，能够降低变异执行的开销，但该变异执行策略的目的在于尽早发现能杀死变异体的测试用例，因此其对 MBFL 方法并不适用。

MBFL 方法根据每个语句的怀疑度进行错误定位分析，需计算各个语句变异体的怀疑度最大值，那么可能是怀疑度最大值的变异体应被所有测试用例执行，而对于那些不可能是怀疑度最大值的变异体，只需执行部分测试用例以确定其怀疑度值不可能为最大即可。因此减少不可能是怀疑度最大值的变异体及其测试用例的执行开销，既可保证 MBFL 方法的错误定位精确度，又可降低变异执行的开销，提高 MBFL 方法的效率。根据这一思路，本文提出了一种针对 MBFL 的动态变异执行策略(Dynamic Mutation Execution Strategy，DMES)，在 MBFL 对每个语句的所有变异体执行测试用例集的过程中，动态调整变异体和测试用例的执行顺序并终止不必要的变异体和测试用例的执行，以更快地得到其语句的怀疑度，在保证错误定位精确度不变的前提下，提高 MBFL 方法的执行效率。

本文选取 6 个程序包的 127 个错误版本程序作为被测程序，对应用和未应用 DMES 策略的 MBFL 方法进行错误定位效果及其执行开销实验评估，结果表明：在保证错误定位精确度不变的情况下，应用 DMES 的 MBFL 相比于原始的 MBFL，可以有效地减少变异执行的开销，提高了基于变异的错误定位效率。

2 基于变异的错误定位

基于变异的错误定位方法 MBFL 是一种对被测程序执行测试用例，利用其通过或失败信息，以及变异体被测试用例杀死与否信息进行错误定位分析的技术[8-10]。MBFL 方法首先对被测程序执行所有测试用例，获得覆盖信息以及测试通过或失败信息，对失败的测试用例所覆盖的语句植入若干故障，生成相应的变异体，再对变异体执行测试用例集，得到其被杀死与否的信息，在此基础上根据怀疑度公式计算每个变异体的怀疑度，进而得到每个语句的怀疑度值，将所有语句怀疑度从高到低排序得到被测程序的怀疑度表，以此进行错误定位分析。

在 MBFL 中，变异体怀疑度用来衡量变异体与被测程序行为的相似程度，语句怀疑度用来度量其可能出现错误的概率，定义如下：

假设 P 为被测程序，T 为 P 的测试用例集，T 又可分为通过(Passed)的测试用例集 T_p 和失败(Failed)的测试用例集 T_f。执行结果与预期结果相同的测试用例称为 Passed 的测试用例；反之则为 Failed 的测试用例。Failed 的测试用例集 T_f 覆盖的语句集合记为 $cov(T_f)$。假设 s 为 P 中的一条语句并且 $s \in cov(T_f)$，那么在语句 s 上植入若干故障生成变异体，其变异体集合记为 M_s。对于变异体 $m \in M_s$，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 $Sus(m)=Sus(a_{np},a_{nf},a_{kp},a_{kf})$，常用表 1 中的怀疑度公式进行计算，其中 $a_{np}=|T_p \cap T_n|$，$a_{kp}=|T_p \cap T_k|$，$a_{nf}=|T_n \cap T_f|$，$a_{kf}=|T_f \cap T_k|$，4 个参数还满足：

$$totalfailed=|T_f|=a_{kf}+a_{nf} \quad (1)$$

$$totalpassed=|T_p|=a_{kp}+a_{np} \quad (2)$$

则语句 s 的怀疑度 $Sus(s)$ 定义为 s 的变异体集合 M_s 中变异体怀疑度的最大值，即 $Sus(s)=\mathrm{Max}(Sus(m_1),Sus(m_2),\cdots,Sus(m_n))$，$m_1,m_2,\cdots,m_n \in M_s$。

表 1 常用的怀疑度公式

名称	表达式
Tarantula	$\dfrac{\frac{a_{kf}}{a_{kf}+a_{nf}}}{\frac{a_{kf}}{a_{kf}+a_{nf}}+\frac{a_{kp}}{a_{kp}+a_{np}}}$
Ochiai	$\dfrac{a_{kf}}{\sqrt{(a_{kf}+a_{nf})\times(a_{kf}+a_{kp})}}$
Op2	$a_{kf}-\dfrac{a_{kp}}{a_{kp}+a_{np}+1}$

例如，假设被测程序 P 的测试用例集 T 包含 6 个测试用例 T1－T6。为计算 P 中语句 s 的怀疑度 $Sus(s)$，MBFL 方法首先对被测程序执行测试用例集 T 得到如图 1 左上部中所示的被测程序执行信息，其中“C”行表示测试用例集 T 执行覆盖“1”或未覆盖“0”语句 s 的信息；“R”行表示测试用例集 T 执行通过“P”或失败“F”信息。然后对 s 植入 6 个故障得到 6 个变异体 M1－M6，对 M1－M6 执行 T1－T6 得到图 1 左下部所示的变异体执行信息，其中每个元素表示一个变异体-测试对(Mutant-Test Pair，MTP)的执行结果，如〈M1，T1〉表示变异体 M1 上执行测试用例 T1 的结果，1 表示 Killed，0 表示 Not Killed，-表示语句 s 未被测试用例 T1 覆盖，T1 无法杀死该语句对应的变异体，故不需要在变异体 M1 上执行 T1。根据图 1 左部的被测程序执行信息和变异体执行信息，利用表 1 中的怀疑度公式，如 Ochiai 公式计算每个变异体的怀疑度，其结果如图 1 右侧最后一列所示。语句 s 的怀疑度值为 M_s 中变异体怀疑度的最大值 0.82。

变异执行开销可用 MTP 的执行次数进行评估[12]，根据图 1，MBFL 方法对 6 个变异体各执行 4 个测试用例，故计算语句 s 的怀疑度共需要执行 24 次 MTP。

	T1	T2	T3	T4	T5	T6					
被测程序执行信息											
C	0	1	1	1	1	0					
R	P	F	P	P	F	P	怀疑度计算结果				
变异体执行信息							a_{np}	a_{nf}	a_{kp}	a_{kf}	Sus
M1	-	1	0	0	0	-	4	1	0	1	0.71
M2	-	0	1	0	1	-	3	1	1	1	0.50
M3	-	1	1	0	1	-	3	0	1	2	**0.82**
M4	-	0	1	0	1	-	3	1	1	1	0.50
M5	-	1	1	0	1	-	3	0	1	2	**0.82**
M6	-	1	0	0	0	-	4	1	0	1	0.71

图 1 MBFL 使用 Ochiai 公式计算语句怀疑度的相关信息

3 针对 MBFL 的动态变异执行策略 DMES

在保证 MBFL 错误定位精确度不下降的前提下，为尽可能地减少 MBFL 方法所需的变异执行开销，本文提出了一种针对 MBFL 的动态变异执行策略 DMES：在语句怀疑度计算过程中，优化变异执行过程以尽可能减少变异执行开销，提高 MBFL 方法的效率。优化变异执行包括两部分：变异体执行优化(Mutation Execution Optimizing，MEO)和测试用例执行优化(Test cases Execution Optimizing，TEO)。MEO 策略关注变异体的执行顺序，利用 T_f 在变异体上的执行信息确定其被 T_p 执行的顺序，期望尽可能早地执行具有怀疑度最大值的变异体并终止后续变异体的执行，以减少不必要的变异体执行开销。TEO 策略是在 MEO 的基础上，针对已确定的变异体执行顺序，关注测试用例集 T_p 的执行顺序，期望尽可能早地确认一个变异体的怀疑度是否可为其语句的怀疑度，以减少变异体被不必要的测试用例执行的开销。

3.1 变异体执行优化 MEO

观察可知，恰当的变异体执行顺序有助于较早地获得其语句的怀疑度。例如图 1 中，若按从 M1 到 M6 的顺序执行变异体，每个变异体按 T2 到 T5 的顺序执行测试用例(T1、T6 由于不覆盖该语句，无需执行便可知其结果为 Not Killed)，则至少需要执行 3 * 4=12(执行 M1、M2、M3)次 MTP 才能得到其变异体怀疑度的最大值 0.82。若按 M3，M1，M2，M4－M6 的顺序执行变异体，则仅需执行 4 次 MTP(仅在 M3 上执行 T2、T3、T4 和 T5)即可获得变异体怀疑度的最大值 0.82。

因此，变异体执行优化 MEO 策略的设计思路是：怀疑度值越大的变异体越早执行，一旦确定语句的怀疑度则终止后续变异体的执行。由怀疑度计算公式可知，较高的 a_{kf} 可能导致较大的怀疑度值，因此 MEO 策略根据 a_{kf} 值确定变异体的执行顺序：即 a_{kf} 值高的变异体优先执行。

因 $a_{kf}=|T_f \cap T_k|$，所以 MEO 策略首先对 M_s 执行 Failed 的测试用例集 T_f，统计每个变异体被 T_f 杀死的测试用例个数，得到每个变异体的 a_{kf} 值，据此确定变异体被测试用例集 T_p 执行的顺序。

此外，变异体执行 T_p 前，若能预测 T_p 的执行结果并估计其怀疑度上限，则在一定条件下可减少后续变异体的执行。例如，若已知变异体 m 的怀疑度为 0.82，下一个变异体 m' 的怀疑度上限值为 0.71，则变异体 m' 的怀疑度不可能为其语句怀疑度，因此不必对 m' 执行 T_p。通过分析表 1 常用的 3 个怀疑度公式，可知当 T_p 中的测试用例都不能杀死变异体 m，即 a_{np} 取最大值 $totalpassed$，a_{kp} 取最小值 0 时，变异体 m 的怀疑度达到其上限。此时，若存在其他变异体的怀疑度大于或等于此上限，则表明该变异体的怀疑度对语句怀疑度的计算无贡献，故无需对该变异体执行 T_p，减少了变异执行开销。

以 Ochiai 公式为例，变异体的怀疑度上限值 $Sus_{upper}(m)$ 的计算公式如下：

$$Sus_{upper}(m)=\frac{a_{kf}}{\sqrt{totalfailed \times a_{kf}}} \tag{3}$$

例如，图 1 中若按 MBFL 方法计算语句怀疑度需执行 24 次 MTP(对 6 个变异体各执行 4 个测试用例)。若 MBFL 应用本文的 MEO 策略，则对变异体 M1－M6 执行测试用例集合 T_f(T2 和 T5)，需执行 6 * 2=12 次 MTP，得到各变异体的 a_{kf} 值如图 1 右部第 4 列所示。按 a_{kf} 降序对变异体进行排序，则变异体被 T_p 执行的顺序为 M3、M5、M1、M2、M4、M6，计算其怀疑度上限分别为 1、1、0.71、0.71、0.71、0.71。依次对 M3、M5 执行 Passed 的测试用例 T3 和 T4 之后，得到当前最大怀疑度值为 0.82。由于 0.82 大于后续变异体(M1、M2、M4 和 M6)的怀疑度上限值 0.71，终止后续变异体的执行。因此，应用 MEO 策略的 MBFL 在保持语句怀疑度值为 0.82 的情况下，只需执行 12+2 * 2=16 次 MTP 即可计算出其语句怀疑度值 0.82。相比于原始 MBFL 的 24 次 MTP 执行，少执行了 8 次 MTP，明显提高了 MBFL 方法的执行效率。

3.2 测试用例执行优化 TEO

对一个变异体而言，恰当的 T_p 测试用例执行顺序也有助于较早地终止该变异体的执行，从而提高语句怀疑度的计算效率。这是由于变异体怀疑度值的计算也与 a_{kp} 有关，若能较早地执行 T_p 中能杀死变异体的测试用例，则可较早确定该变异体的怀疑度是否可能为其语句怀疑度。有研究表明，一个测试用例在同一语句的变异体集 M_s 上具有相似的执行结果[12]。故 T_p 在变异体 $m(m \in M_s)$ 上执行的顺序可由其已杀死的变异体数确定，即在变异体集 M_s 执行 T_p 的过程中，若 $t_i \in T_p$ 杀死的变异体越多，则在当前变异体的执行过程中，t_i 越早被执行。

因此，测试用例执行优化 TEO 策略的设计思路是：对一个变异体 m 执行 T_p 时，杀死 M_s 中变异体越多的测试用例越优先执行；执行过程中若能确定 m 的怀疑度不可能为其语句怀疑度，则终止后续测试用例在 m 上的执行。具体而言，TEO 策略记录 T_p 中每一个测试用例杀死 M_s 中变异体的个数作为其历史执行信息 $history$，据此确定下一个变异体 m' 上 T_p 的执行顺序。在对 m' 执行 T_p 的过程中，若 m' 的怀疑度小于当前语句怀疑度 $cur_{\max}$，则 m' 的怀疑度不可能为其语句怀疑度，故终止后续测试用例的执行。

由表 1 中怀疑度计算公式可知，一个变异体被 T_p 中越多的测试用例杀死，即 a_{kp} 越大则该变异体的怀疑度越小。因此，TEO 在对每个变异体执行 T_p 之前，可计算得到一个使变异体的怀疑度小于当前 cur_{max} 所必须的杀死该变异体的 T_p 测试用例数目，称作阈值 $threshold$。若当前杀死变异体 m 的 T_p 测试用例数 cur_{kp} 大于或等于 $threshold$ 值，表明继续执行 T_p 将使变异体 m 的怀疑度呈下降趋势，则可终止后续 T_p 测试用例的执行。变异体 m 的阈值 $threshold$ 可通过怀疑度计算公式得到，以 Ochiai 公式为例，利用式(1)和式(2)化简消去 a_{np} 和 a_{nf}，得到式(4)：

$$Sus_{upper}(m)=\frac{a_{kf}}{\sqrt{totalfailed \times (a_{kf}+a_{kp})}} \tag{4}$$

当 $Sus(m)$ 取当前语句怀疑度 $cur_{\max}$ 时，a_{kp} 值即为 m 的阈值 $threshold$。但 $cur_{\max}$ 初始值为 0，此时其阈值为最大值 $totalpassed$。故 $threshold$ 的计算公式为：

$$threshold=\begin{cases}\left\lceil \dfrac{a_{kf}^2}{cur_{\max}^2 \times totalfailed}-a_{kf} \right\rceil, & cur_{\max}\neq 0\\ totalpassed, & cur_{\max}=0\end{cases} \tag{5}$$

例如，图 1 中应用 MEO 策略，变异体执行优先顺序为 M3、M5、M1、M2、M4、M6。执行 M3 时，应用 TEO 策略，此

时当前语句怀疑度为初始值($cur_{max}=0$)且无任何历史信息，故其 $threshold=totalpassed=2$ 且可顺序执行 T3 和 T4。执行 T3 后，当前语句怀疑度 cur_{max} 为 0.82，历史信息 *history* 为〈1,0〉(表明 T_p 中 T3 Killed 变异体 M3，而 T4 Not Killed)。执行 M5 时，应用 TEO 策略，先计算其 *threshold* 值，此时 M5 的 a_{kf} 为 2，故其 $threshold=\lceil 2^2/(0.82^2*2)-2\rceil=1$，按照 *history* 应先执行 T3 再执行 T4。执行 T3 后，得到当前杀死变异体 M5 的 T_p 测试用例数 cur_{kp} 为 1，满足 $cur_{kp}\geqslant threshold$，故终止后续 T_p 执行，即不再对 M5 执行 T4。由此可以看出，在 MEO 上应用 TEO 策略，计算其语句怀疑度需执行 MTP 15(12+2+1=15)次。若仅应用 MEO 策略而不应用 TEO 策略，则对 M5 还需执行 T4，所需的 MTP 执行次数为 16。可见，在 MEO 策略上再应用 TEO 策略，可以进一步减少变异执行开销，提高 MBFL 的效率。

3.3 动态变异执行策略 DMES

针对 MBFL 的动态变异执行策略 DMES 由 MEO 和 TEO 两部分构成。若已知被测程序 P，测试用例集 T，T 在 P 上执行的 Passed 或 Failed 信息 R，被 Failed 的测试用例所覆盖的语句 s，以及怀疑度计算公式 *Sus*，则计算 s 怀疑度 $Sus(s)$ 的算法如图 2 所示。算法首先对语句 s 植入故障，得到语句 s 的变异体集合 M_s，并初始化语句 s 的怀疑度 cur_{max} 和历史信息 *history*(1－3 行)；然后应用 MEO 策略，对所有变异体执行 T_f，统计每个变异体 m 的 a_{kf} 值，计算其怀疑度上限 $Sus_{upper}(m)$ 并按 a_{kf} 降序确定变异体被 T_p 执行的顺序 seq_m (5－12 行)；根据 seq_m 的顺序，应用 TEO 策略，在变异体 m 执行 T_p 前，依据历史执行信息 *history* 确定 T_p 在 m 上的执行顺序 seq_{tp}，并按式(5)计算 m 的 *threshold* 值(14－15 行)。对变异体 m 依次执行 seq_{tp} 测试用例并更新 *history*，计算当前杀死变异体的 T_p 测试用例数 cur_{kp}，若 cur_{kp} 大于或等于 *threshold*，则终止变异体 m 的执行，否则计算 $Sus(m)$ 并更新 cur_{max}(16－24 行)。若 cur_{max} 大于或等于下一个变异体的怀疑度上限，则 cur_{max} 为语句怀疑度 $Sus(s)$(25 行)，算法结束。

假设语句 s 的变异体数为 $|M_s|=n$、通过测试用例数为 p、失败测试用例数为 q，对复杂度进行分析：由于算法包含两个二层循环，其时间复杂度分别为 $O(np)$ 和 $O(nq)$，因此算法的时间复杂度为 $O(n*\max(p,q))$；算法临时占用的存储空间主要包括 M_s 和 *history*，因此空间复杂度为 $O(\max(n,p))$。

Algorithm: MEO&TEO
Input: 被测程序 P，测试用例集 T，T 在被测程序上的执行结果向量 R，被 T_f 覆盖的一个语句 s，怀疑度公式 Sus
Output: 语句 s 的怀疑度 Sus(s)
1. M_s ← Mutate(s) //对语句 s 植入若干故障，得到变异体集合 M_s
2. cur_{max} ← 0
3. history ← $\vec{0}$ //初始化历史执行信息
4. //history 是一个向量，其分量个数为 $|T_f|$
5. for all m ∈ M_s do
6. for t ∈ T && R(t) = Failed do
7. Execute 〈m,t〉 //对变异体 m 执行 Failed 的测试用例 t
8. end for
9. Count a_{kf} of m
10. Calculate Sus_{upper}(m) //计算变异体 m 怀疑度上限
11. end for
12. seq_m ← Reorder M_s based on a_{kf} //确定变异体的执行顺序
13. for m: seq_m do
14. seq_{tp} ← Reorder T_p based on history //确定变异体 m 上 T_p 执行顺序
15. Calculate threshold //计算阈值
16. for t: seq_{tp} do
17. Execute〈m,t〉
18. Update history
19. Count cur_{kp}
20. if $cur_{kp}\geqslant$ threshold then goto line26
21. //终止后续 T_p 执行，跳转到下一变异体的执行
22. end for
23. Calculate Sus(m)
24. cur_{max} ← Max(Sus(m), cur_{max})
25. if $cur_{max}\geqslant Sus_{upper}$(NEXT(m)) then return Sus(s) ← cur_{max}
26. end for

图 2 应用 MEO&TEO 的语句怀疑度计算算法

4 实验评估

为验证本文提出的 DMES 能否在不降低 MBFL 错误定位精确度的同时，减少变异执行的开销，实验选取 6 个基准程序包，分析比较原始 MBFL 与应用 DMES 的 MBFL 方法在错误定位精确度以及 MTP 执行次数两方面的优劣。

4.1 基准程序与工具

实验选择错误定位研究广泛使用的 6 个程序包作为基准[3-5,8-10]，其中 5 个程序包 tcas、totinfo、schedule、printtokens 及 replace 来自 Simens 程序集，另一个为 space 程序包，它们包含的测试用例集和错误版本等信息均可从软件基础设施库下载[13]，这些程序包的相关信息如表 2 所列。6 个程序包中共含有 150 个错误版本，实验排除了部分不适合的错误版本，最终以 127 个错误版本作为实验评估的被测程序。排除原因主要有：程序包中的测试用例集未能检测出错误、发生段错误导致无法搜集完整的覆盖信息或出错语句过多导致评估困难等。实验使用 Gcov 工具获取被测程序的覆盖信息，使用 Proteum/IM2.0 变异测试工具生成变异体[14]。实验对表 2 中前 5 个程序包的所有错误版本生成变异体并获取变异体执行信息。因 space 程序版本规模较大，为减少开销，实验仅对 space 程序包的一个错误版本进行变异体生成，其他错误版本共用其变异体执行信息。

表 2 基准程序相关信息

程序名	错误版本数	使用版本数	代码行数	测试用例数	生成变异数
printtokens	7	3	341～343	4130	4235～4263
schedule	9	5	290～294	2650	2203～2247
tcas	41	40	133～137	1608	4739～5323
totinfo	23	23	272～273	1052	6314～6438
replace	32	28	508～515	5542	10661～10988
space	38	28	5882～5904	13585	38830

4.2 实验结果与分析

为检验 DMES 是否会对 MBFL 的错误定位精确度产生影响，本文使用表 1 中的 3 种怀疑度公式：Tarantula、Ochiai 及 Op2，以表 2 程序的 127 个错误版本为被测程序，对应用 DMES 前后的 MBFL 错误定位精确度进行了统计分析，结果如表 3 所列，其中每一行为按怀疑度表检查一定比例的代码时，检查到错误语句的被测程序占 127 个被测程序的比例；ORIG 和 DMES 分别表示原始 MBFL 和应用 DMES 的 MBFL 在不同怀疑度公式下的错误定位效果。例如，第一行的第

二列和第三列分别表示检查到1%的代码时，原始MBFL和应用DMES的MBFL使用Tarantula怀疑度计算公式，检查到错误语句的被测程序均占总被测程序的24%。

由表3可以看出，使用表1中相同怀疑度公式对应的两列完全相同，这说明使用表1的任意怀疑度公式的MBFL方法，在应用DMES前后均具有相同的错误定位效果，故应用DMES策略不会影响MBFL方法的错误定位精确度，即应用和未应用DMES的MBFL方法具有相同的错误定位效果。

另一方面，从表3还可发现，使用Ochiai怀疑度计算公式，除了在20%和60%的代码检查比外，在其他9个代码检查比上都优于或等于使用Tarantula或Op2公式的错误定位精确度，说明使用Ochiai怀疑度计算公式的MBFL可以获得较好的错误定位效果。为减小实验开销，本文仅使用Ochiai公式的MBFL方法，分析应用本文DMES策略对MBFL执行效率的影响。

表3 MBFL应用DMES前后错误定位精确度对比

代码	Tarantula		Ochiai		Op2	
检查比(%)	ORIG	DMES	ORIG	DMES	ORIG	DMES
1	24%	24%	38%	38%	31%	31%
10	80%	80%	86%	86%	74%	74%
20	91%	91%	**90%**	**90%**	83%	83%
30	96%	96%	98%	98%	91%	91%
40	98%	98%	98%	98%	94%	94%
50	98%	98%	98%	98%	95%	95%
60	99%	99%	**99%**	**99%**	100%	100%
70	100%	100%	100%	100%	100%	100%
80	100%	100%	100%	100%	100%	100%
90	100%	100%	100%	100%	100%	100%
100	100%	100%	100%	100%	100%	100%

同样以6个程序包的127个错误版本为被测程序，利用原始MBFL、应用MEO的MBFL以及MEO上再应用TEO的MBFL计算其语句怀疑度，所需的变异-测试对MTP执行次数如图3所示，其中横轴上的每根柱子对应一个错误版本，纵轴是其MTP执行次数，单位为百万次。每根柱可分为3根，分别表示原始的、应用MEO、同时应用MEO及TEO的MBFL在相应错误版本上计算其语句怀疑度所需的MTP执行次数，如图3(a)右部所示。图3中每根累积柱的上部为应用MEO时减少的MTP执行次数；中部为在MEO基础上再应用TEO减少的MTP执行次数；下部为应用MEO及TEO时所需要的MTP执行次数。由图3可以看出，在所有程序包的大多数版本上，应用MEO和TEO都可以显著减少MBFL所需的MTP执行次数。

表4给出了在各个程序包的错误版本上，应用MEO以及同时应用MEO和TEO的MBFL相对于原始MBFL减少的MTP执行次数百分比的平均值(Aver)、最大值(Max)和最小值(Min)。由表4可知，同时应用MEO和TEO两种策略相比于原始MBFL平均减少的MTP执行次数百分比分别是：printtokens为22.8%；schedule为30.3%；tcas为77.9%；totinfo为36.0%；replace为59.8%；space为56.4%。平均而言，可以减少58.4%的MTP执行次数。

但也有一些错误版本比较特殊，如totinfo的错误版本v3和v14，对应图3(c)的柱3和柱14，原始MBFL方法所需的MTP的执行次数明显少于totinfo的其他错误版本。分析原因发现这两个版本的Failed测试用例极少，只有3个和2个，分别占其测试用例集的0.3%和0.2%，并且Failed的测试用例只覆盖了少量的语句，约占totinfo可执行语句数的10%，而MBFL仅对被Failed测试用例所覆盖的语句生成并执行变异体，因此这两个版本的MTP执行次数明显较少。又如replace的错误版本v26，对应图3(d)中的柱24，应用MEO只减少了0.3%的MTP执行次数。分析可知其大部分变异体的a_{kf}都较大且怀疑度都较高，因此MEO无法较早确定变异体怀疑度的最大值并终止后续变异体的执行，使得应用MEO的效果较差。再如space的错误版本v4和v6，对应图3(e)的柱2和柱4，应用MEO和TEO总共只减少了4%和3%的MTP执行次数，分析原因发现这两个错误版本中Failed的测试用例数目较多，分别为12353和12653，占测试用例集的比例分别为91%和93%，不满足MEO的假设：Failed的测试用例占测试用例集的比例较小，应用MEO对变异体执行T_f测试用例时执行了大量的MTP，因此应用MEO效果不明显。

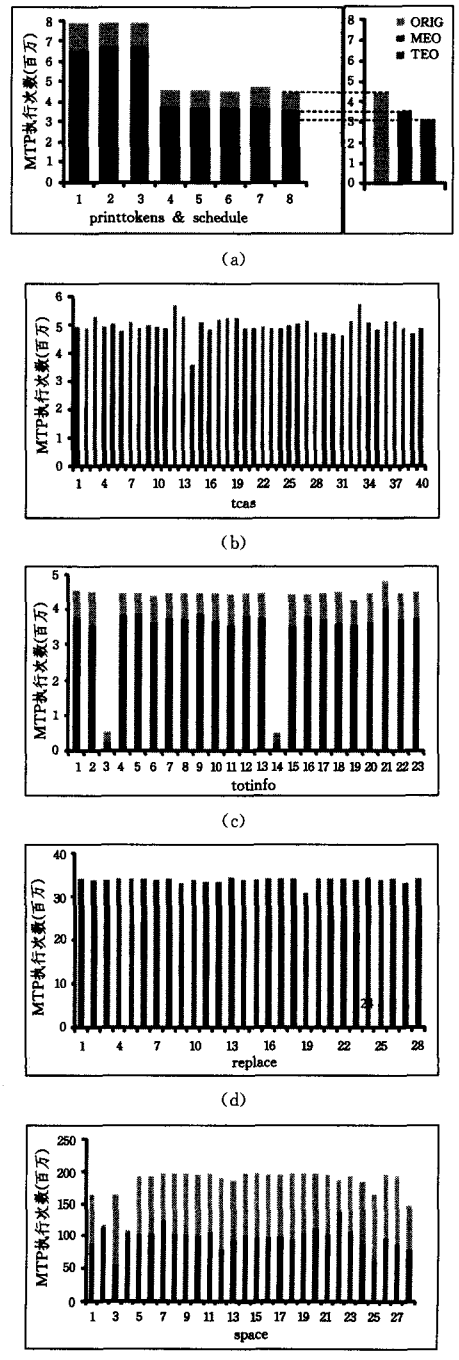

图3 MTP执行次数对比

(下转第229页)

diction:Bayesian combined neural network approach[J]. Journal of Transportation Engineering,2006,132(2):114-121

[15] He Wei. Research on Prediction of Traffic Flow Using Fuzzy Neural Networks[D]. Lanzhou:Lanzhou Jiaotong University, 2012(in Chinese)
何伟. 模糊神经网络在交通流量预测中的应用研究[D]. 兰州:兰州交通大学,2012

[16] Zhang Xiao-li. A Study on Short-term Traffic Volume Forecasting Based on Non-Parametric Regression[D]. Tianjin:Tianjin University,2007(in Chinese)
张晓利. 基于非参数回归的短时交通流量预测方法研究[D]. 天津:天津大学,2007

[17] Li Zhen-long,Zhang Li-guo,Qian Hai-feng. Review of the Short-term Traffic Flow Forecasting Based on the Non-parametric Regression[J]. Journal of Transportation Engineering and Information,2009,6(4):34-39(in Chinese)
李振龙,张利国,钱海峰. 基于非参数回归的短时交通流预测研究综述[J]. 交通运输工程与信息学报,2009,6(4):34-39

[18] Guo F,Krishnan R,Polak J W. Short-term traffic prediction under normal and incident conditions using singular spectrum analysis and the k-nearest neighbour method[C]// IET and ITS Conference on Road Transport Information and Control (RTIC 2012). IET,2012:1-6

[19] Elfaouzi N E. Nonparametric traffic flow prediction using kernel estimator[C]// Internaional symposium on transportation and traffic theory. 1996:41-54

(上接第 203 页)

表 4 应用 MEO 和 TEO 减少的 MTP 执行次数百分比

程序包	MEO			MEO 和 TEO		
	Aver	Max	Min	Aver	Max	Min
printtokens	15.9%	17.8%	14.8%	22.8%	27.8%	18.3%
schedule	19.5%	21.8%	17.9%	30.3%	37.0%	26.1%
tcas	60.2%	85.0%	40.5%	77.9%	89.3%	68.0%
totinfo	20.3%	61.0%	13.5%	36.0%	62.9%	25.7%
replace	39.3%	82.3%	0.3%	59.8%	89.6%	28.6%
space	45.6%	66.4%	3.0%	56.4%	75.2%	3.0%

结束语 基于变异的错误定位方法 MBFL 作为一种具有高定位精确度的错误定位技术,存在着执行开销过大的问题。本文从优化变异执行策略的角度出发,提出了动态变异执行策略 DMES,避免在变异体执行过程中执行不必要的变异-测试对。通过对 6 个程序包的 127 个错误版本进行实验分析,一方面验证了应用 DMES 并不会对 MBFL 的错误定位准确度产生影响;另一方面,验证了应用 DMES 可有效减少变异-测试对的执行开销。实验结果表明本文提出的 DMES 在不降低 MBFL 的定位准确度的条件下,可以有效减少计算语句怀疑度所需执行的变异-测试对次数,提高了 MBFL 方法的执行效率。

此外,现有针对 MBFL 的执行开销减少方法主要从减少变异体数量方面进行考虑,而本文方法则关注尽量避免变异体执行过程中不必要的执行开销。因此,后续研究可考虑将二者结合,进一步提高 MBFL 的执行效率。

参 考 文 献

[1] Masri W,Abou-Assi R,El-Ghali M,et al. An empirical study of the factors that reduce the effectiveness of coverage-based fault localization[C]//Proceedings of the 2nd International Workshop on Defects in Large Software Systems:Held in conjunction with the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2009). ACM,2009:1-5

[2] Yu K,Lin M X. Advances in Automatic Software Fault Localization Techniques[J]. Chinese Journal of Computers,2011,34(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

[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] Masri W. Fault localization based on information flow coverage [J]. Software: Testing, Verification and Reliability, 2010, 20 (2):121-147

[7] Santelices R,Jones J A,Yu Y,et al. Lightweight fault-localization using multiple coverage types[C]// IEEE 31st International Conference on Software Engineering,2009(ICSE 2009). IEEE, 2009:56-66

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

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

[10] 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. 2014:1293-1300

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

[12] 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

[13] 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

[14] Delamaro M E,Maldonado J C,Vincenzi A M R. Proteum/IM 2.0:An integrated mutation testing environment[M]// Mutation testing for the new century. Springer US,2001:91-101