文章编号:1671-4598(2023)08-0016-08 DOI:10.16526/j.cnki.11—4762/tp.2023.08.003 中图分类号:TP311.55 文献标识码:A

基于频谱增强的软件多故障定位

陈 琪[1,2],周世健[1],樊 鑫[1,2],邹凯胜[1,2],肖 鹏[1,2]
(1.南昌航空大学 软件学院,南昌 330038;
2.南昌航空大学 软件测评中心,南昌 330038)

摘要:检测故障是繁琐而耗时的,为了提高传统软件故障定位方法的效率以及精确度,提出了一种基于频谱的故障定位新方法;充分利用了失败的测试用例与故障之间的关系,通过使用频谱增强的方式,采取逻辑与运算的关键技术和方法,对失败的测试用例进行精简,优化失败测试用例中的频谱信息,从而得到频谱增强后的测试用例,克服了冗余测试用例对定位效果的消极影响,再根据新的频谱信息计算可疑度值,最后生成优化后的可疑度排序列表;首次将频谱增强的方法同时运用到单故障与多故障程序场景中,在包含植入故障的西门子程序和真实故障的Defects4j程序中,经实验检测证明本研究方法能够显著减少代码检查的范围,尤其是在高性能范围内(EXAM5%),并且仅通过检查Top-1至少能有效地定位超过原有约20%的故障,结果表明基于频谱增强的故障定位方法有效提升了检测率,可以更好地帮助程序员精准定位故障位置。

关键词:频谱增强;多故障定位;测试用例;软件调试;程序谱

Software Multiple Fault Localization based on Spectrum Enhancement

CHEN Qi[1,2], ZHOU Shijian[1], FAN Xin[1,2], WU Kaisheng[1,2], XIAO Peng[1,2]
(1. School of Software, Nanchang Hangkong University, Nanchang 330038, China;
2. Software Testing and Evaluation Center, Nanchang Hangkong University, Nanchang 330038, China)

Abstract: Detecting faults is tedious and time-consuming, in order to improve the efficiency and accuracy of traditional software fault location methods, a new method of fault location based on spectrum is proposed; the relationship between failed test cases and faults is fully utilized, and the key techniques and methods of logic conjunction operations are adopted to streamline the failed test cases and optimize the spectrum information in the failed test cases by using spectrum enhancement. The method of spectrum enhancement is applied to both single-fault and multi-fault program scenarios for the first time. In Siemens programs with embedded faults and Defects4j programs with real faults, experimental tests have demonstrated that this research method can significantly reduce the scope of code checking, especially in the high performance range (EXAM 5%), and can effectively locate more than about 20% of the original faults by checking Top-1 only, showing that the spectrum enhancement-based fault location method effectively improves the checking rate and can better help programmers pinpoint the location of faults.

Keywords: spectrum enhancement; multiple fault localization; testcase; softwaredebugging; programspectrum

0 引言

软件正成为低成本实现各种系统中复杂功能的首选技术,日常生活的许多方面都依赖于软件的正确操作,因此保证软件的质量和及时的维护在每个开发过程中都起着至关重要的作用。测试指的是检查软件的正确性和它是否符合给定的规范的实践状态。虽然测试用例自动生成工具愈渐成熟,仍然存在部分由开发人员手工制作的测试用例,在暴露错误行为方面测试是有效的,但识别失败背后的根本原因仍然主要是一种手动活动,需要大量的时间和成本。

各种故障定位方法用来帮助开发人员定位故障的根本原因,例如,基于频谱的故障定位[1-6],基于切片的故障定位[7-8],基于机器学习的故障定位[9-10],和基于突变的故障定位[11-14]。其中,基于频谱的故障定位技术是研究最广泛的故障定位技术之一。尽管SBFL是一种特别轻量级的方法,但与其他方法相比,它已被证明具有竞争力,且具备与语言无关、易于使用,并且在测试执行时间相对开销较低的特性。

理想的故障定位技术总是将故障程序实体排在前列。然而在实践中,尽管已经提出了各种SBFL技术,如Jaccard/Ochiai[2],Op2[4]和Tarantula[1],没有一种技术可以总是表现最好的,开发人员通常必须在找到真正的错误之前检查各种非故障实体(即程序频谱中的一个维度)。利用失败和成功的测试用例提供的证据,根据被测实体参与错误行为的怀疑程度进行分析和排序。直观地说,一个实体在测试用例失败时使用得越多,它就越可疑,即它就越有可能与这种不正确性的原因有关。类似地,成功的测试用例所使用的实体越多,它就越被认为不可疑。

现有的SBFL技术的一个局限性是,它们不能完全捕获

收稿日期:2023-02-22; **修回日期**:2023-02-27。
基金项目:江西省自然科学基金资助项目(20212BAB212009)。
作者简介:陈 琪(1998-),女,江西南昌人,硕士,主要从事软件故障定位方向的研究。
引用格式:陈 琪,周世健,樊 鑫,等.基于频谱增强的软件多故障定位[J].计算机测量与控制,2023,31(8):16-23.

每个已执行的测试用例的结果与所涉及的程序单元之间的本地关系。例如，SBFL 技术，如 Dstar[15]，会假设两个失败的测试用例对在两个测试用例中执行的一个程序单元的怀疑性贡献相同，而不管在测试用例中涉及的程序实体的数量不同。在实践中，一个失败的测试用例可能比其他用例更有价值，因为它只涉及几个程序实体。其次，在许多现有的 SBFL 技术中，存在一个被忽略的事实，即程序实体可能以不同的方式协作贡献每个测试用例的结果。

我们的关键见解是，存在一种更丰富的频谱增强形式，考虑了如何增强失败的测试用例中的频谱分析，可以提供更有效的故障定位。详细来说，存在两个测试和，它们都是失败的测试，其中覆盖了 100 个程序实体，而只覆盖了 10 个程序。根据我们的直觉，在故障定位方面比较有帮助，因为有一个更小的搜索空间来定位故障（s）。由于每一个执行失败的测试用例都必然执行了错误语句，即错误语句必然被覆盖，那么[16]已经证实了在单故障情况下，将所有执行失败的测试用例的覆盖情况取交集，则会缩小故障语句的覆盖范围，从而提高查找效率。经过频谱增强后的测试用例作为新的增强版测试用例来对程序进行验证。传统的 SBFL 技术忽略了这些有用的信息，并认为和在 SBFL 上做出了相同的贡献，例如，和执行的程序实体将被作为相同的处理，而不管测试所覆盖的实体数量。

为了克服现有的 SBFL 技术的局限性，我们通过明确地考虑失败测试用例的贡献，更有效地增强了现有的程序频谱。基于以上见解，我们提出了基于频谱增强的软件多故障定位方法（SBFL（E），spectrum-based fault localization (enhanced)），充分利用了执行失败测试用例的程序语句覆盖情况，缩减了运行时的执行失败测试用例数量，对基于频谱的故障定位方法进行改良，并将单故障扩展到多故障实验，极大提高了软件故障定位的效率和有效性。

1　软件故障定位

1.1　基于频谱的故障定位

SBFL 使用程序频谱和测试结果来定位软件故障。程序频谱是测试用例的覆盖信息和执行结果的集合。一个程序的频谱，通常以一个矩阵的形式出现，它描述了该程序的动态行为。测试结果将记录测试用例是否失败或通过。通过较多失败测试用例的实体，通过较少成功测试用例的实体就越可疑[17-18]。相反，实体通过越少的失败测试用例，通过越多成功的测试用例，实体的可疑度值就越低。程序谱中所涉及的实体类型可以是语句、分支、函数、谓词等，本文以语句作为实体。

SBFL 的主要思想是与故障相关的执行信息更有可能是导致程序失效的原因，这些信息往往包含在失败的测试用例，而更少发生在成功的测试用例当中。如图 1 所示，SBFL 分析了实体与测试用例通过或失败之间的动态相关性，这个相关性近似于一个实体导致程序失效的可能性。频谱信息矩阵由 M 个测试用例和每个测试用例中的 N 条语句构成，最后得到测试结果向量 $\boldsymbol{R}$。对于一个语句 X_MN，其值是二进制的（0/1），“1”表示相应的语句被测试用例执行，“0”表示相应的语句没有被测试用例执行。在结果向量 $\boldsymbol{R}$ 中，其值也是二进制的，“0”表示测试用例的结果为通过，“1”表示测试用例的结果为失败。

N 条语句测试结果

$$M\text{ 个测试用例}\begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1N} \\ x_{21} & x_{22} & \cdots & x_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ x_{M1} & x_{M2}) & \cdots & x_{MN} \end{bmatrix}\begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_M \end{bmatrix}$$

图 1　SBFL 覆盖信息矩阵图

程序执行信息从图 1 中提取，表示为四元组（N_{ef}，N_{ep}，N_{nf}，N_{np}），其中 N_{ef} 代表语句被执行的失败测试用例数，N_{ep} 代表语句被执行的通过测试用例数，N_{nf} 代表语句未被执行的失败测试用例数，N_{np} 代表语句未被执行的通过测试用例数。根据以上 4 个参数，程序中各单元元素的可疑值可以通过表 1 中的可疑值计算公式来计算，这里只列出了部分但常用的公式。如果一个语句的 N_{ef} 值相对较高，而 N_{ep} 值较低，则该语句的可疑分值较高。

表 1　SBFL 可疑度值计算公式

名称	公式
Tarantula	$\dfrac{\dfrac{N_{ef}}{N_{ef}+N_{nf}}}{\dfrac{N_{ef}}{N_{ef}+N_{nf}}+\dfrac{N_{ep}}{N_{ep}+N_{np}}}$
Ochiai	$\dfrac{N_{ef}}{\sqrt{(N_{ef}+N_{nf})\cdot(N_{ef}+N_{ep})}}$
Dstar	$\dfrac{N_{ef}^2}{N_{ep}+N_{nf}}$
Barinel	$\dfrac{1-N_{ep}}{N_{ef}+N_{ep}}$
Jaccard	$\dfrac{N_{ef}}{N_{ef}+N_{nf}+N_{ep}}$

1.2　单故障与多故障定位

单故障程序指的是在程序中只存在唯一确定的一处故障，导致整个程序的失效；多故障程序则指错误数目不定，存在两处或两处以上的故障影响导致程序失效。尽管各种故障定位技术已经被提出，但它们在多故障程序中的应用仍然有限。多故障定位（MFL，multiple fault localization）是指在一个有故障的软件程序中识别多重故障（一个以上的故障）的定位行为。与单故障定位（SFL，single fault localization）假设软件系统中只包含一个故障的传统做法相比，这种方法更加复杂、繁琐，而且成本高。工作流程对比如图 2 所示。

单故障程序中所有失败的测试用例均指向相同的故障，因此测试用例的数量与质量则显得尤为关键，成为优化故障定位方式的突破口。在多故障环境下，除了受到测试用例的影响，更令人棘手的是故障干扰的问题。

这些相互作用可能表现为导致测试用例失败，而通常情况下，由于任何单一故障的存在都不会失败。另外，一

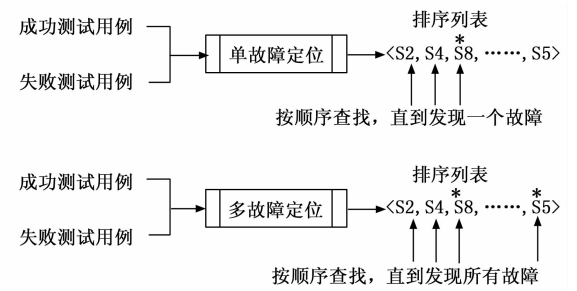


图 2 单故障与多故障定位

个测试通常由于一个故障而失败的测试用例，在增加了另一个故障后，该测试用例可能不再失败，因为它干扰了第一个故障的导致程序失效的作用。当然也存在多个故障于同一个程序中，但不会以通过检查测试用例可以观察到故障的方式相互干扰。在这种情况下。程序中每个故障的影响似乎是独立于其他故障。本文则对故障干扰不做出假设，专注于提升测试用例尤其是失败的测试用例，使用传统的定位技术同时定位多个故障是困难的，因为它们依赖于失败的测试执行来识别错误的语句。

综合来看，单故障与多故障定位的差异除了存在于需要定位的故障数目的不同，也不免对测试用例的统计处理以及失败测试用例的质量造成消极影响。当故障之间的干扰导致程序包含多个故障时，现有故障定位技术的有效性会降低。程序中的故障越多，开发人员就定位故障的复杂性就越高，从而大大增加故障定位的难度与开销成本。

2 动机实例

SBFL 是基于程序执行的统计数据设计的，其中包括测试覆盖率和测试结果。执行统计数据可以被视为 SBFL 分析的信息源，其中决定了 SBFL 的准确性的关键在于测试用例的性质。冗余的测试用例大大增加了开发人员的时间成本与故障定位的精确性，一个含故障程序实例如表 2 所示。

在现有的研究方法中，我们考虑了一个有缺陷的程序来评估我们技术的效率。Zakari 等人[19]用一个例子来解释 3 种方法：一次调试方法、并行调试方法和多次调试方法。在本文中，我们采用了同样的例子程序来展示我们的方法的工作过程。余晓菲等人[16]已经证实了频谱增强在单故障程序中的有效性，因此表 2 实例仅对多故障程序进行详细分析。在表 2 中，我们显示了示例程序。原始的程序测试套件 T 由 10 个测试用例（T0，T1，T2，T3，T4，T5，T6，T7，T8，T9）和 13 条程序语句组成，其中语句 5 和语句 10 都是错误的。标有“●”的语句执行标志着该语句在该测试运行中被测试用例执行，否则为空。测试用例（T6，T7，T8，T9）是失败的测试用例，而（T0，T1，T2，T3，T4，T5）是通过的测试用例。在这个例子中，每个语句的可疑性是用 Tarantula 系数计算的，这是 SBFL 中常用的故障定位技术之一。

对于单次故障的调试很简单，根据语句的可疑度得分，开发人员通过使用 Tarantula，检查 4 条语句就能清楚地识别出有故障的语句 10（可疑度值为 0.57）。然而，在多故障情况下，在语句 10 中发现的故障必须被修复，并且程序必须被重新测试以发现第二个故障语句 5（可疑度值为 0.23）。在多故障调试过程中，开发人员在找到语句 10 中的第一个故障后不会停止，而是继续搜索下一个故障，直到找到所有故障或预先指定的搜索上限。设置上限的方法（即开发人

表 2 多故障实例

Mid() {Int x,y,z,m;	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	Ta	Tb	Sus (TAR)	Rank (TAR)	Sus (E-TAR)	Rank (E-TAR)
	3,3,5	1,2,3	3,2,2	5,5,5	1,1,4	5,3,4	3,2,1	5,4,2	2,1,3	5,2,6						
1：read("enter 3numbers:",x,y,z);	•	•	•	•	•	•	•	•	•	•	•	•	0.5	5	0.5	4
2：$m=z$;	•	•	•	•	•	•	•	•	•	•	•	•	0.5	5	0.5	4
3：if ($y<z$)	•	•	•	•	•	•	•	•	•	•	•	•	0.5	5	0.5	4
4：if($y<z$)	•	•			•	•			•	•	•		0.43	10	0.5	4
5：$m=z$; //fault1. Correct：$m=y$		•	•	•	•	•			•		•		0.23	11	0.29	11
6：else if($x<z$)	•				•	•			•	•	•		0.5	5	0.78	1
7：$m=x$;	•				•				•	•	•		0.6	1	0.67	2
8：else			•	•			•	•				•	0.6	1	0.5	4
9：if($x>y$)			•	•			•	•				•	0.6	1	0.5	4
10：$m=z$; //fault2. Correct:$m=y$			•				•	•				•	0.57	4	0.67	2
11：else if ($x>z$)				•									0.00	12	0.00	12
12：$m=x$;													0.00	12	0.00	12
13：print("middle number is:", m);	•	•	•	•	•	•	•	•	•	•	•	•	0.5	5	0.5	4
} Pass/Fail Status	P	P	P	P	P	P	F	F	F	F	F	F				

员在进入下一个迭代之前可以搜索可疑语句排名列表的 70%或 60%）已被现有研究采用[20-21]。在我们的研究中，我们没有设置搜索上限，而是在找到所有故障之前停止搜索。

采取多故障场景下频谱增强的方法，实例为双故障程序，覆盖第一个故障语句 5 的失败测试用例仅为 T8，因此频谱增强后的测试用例 Ta 保留 T8 的覆盖信息并剔除原有 T8。失败测试用例 T6 和 T7 覆盖了第二个故障语句 10，且覆盖信息高度重合，对二者进行测试用例精简，采取逻辑与运算得到测试用例 Tb，并将 T6 和 T7 从测试用例集中去除。频谱增强后可用测试用例为 T0～T5 以及 T9～Tb，再进行 SBFL 可疑度值计算。

表 2 清楚地显示，Tarantula 计算的错误语句的等级为 4 和 11。在采用频谱增强的方法进行优化后，我们的技术分别给有问题的语句分配了等级 2 和等级 11。这表明，我们的研究方法比其他现有的方法更有效地优化了可疑性分数，并且能够定位那些使程序失效的语句。

3　基于频谱增强的软件故障定位

对基于频谱的故障定位方法而言，故障定位的效果和效率都高度依赖于测试用例，所以测试用例的选取以及使用测试用例的多少都对实验有至关重要的影响。针对测试用例的研究，本文引入频谱增强方法，充分利用了执行失败测试用例的程序语句覆盖信息，精简了运行时的执行失败测试用例数量，从而提升了测试用例的质量以及软件故障定位的效率。

由于语句执行的覆盖信息都为二进制表示方法，因此将所有失败测试用例的覆盖信息做逻辑与运算，则能够精简频谱信息，从而缩小查找故障语句的范围，并且通过频谱增强后的测试用例一定会覆盖存在故障的语句。我们将与运算操作取得的覆盖信息这个操作称为频谱增强。再使用增强过的频谱信息进行 SBFL 可疑度值计算，最后根据优化后的可疑度排名列表进行更高效的故障定位。整体流程如图 3 所示。

单故障场景下，频谱增强后得到唯一的失败测试用例。程序运行时，唯一的故障语句必然被所有失败测试用例所覆盖，否则测试用例通过，运行结果正确，程序成功执行。而多故障场景中，并不是所有的故障语句都被每一个失败测试用例所覆盖，我们采取根据多故障数量 k 进行频谱增强的方式，最后得到 k 个频谱增强后的失败测试用例。

将失败测试用例集设为 T，$T=\{T_1, T_2, \cdots, T_i, \cdots, T_m\}$，其中 T_i 为第 i 个失败测试用例的执行情况，具体失败测试用例的覆盖信息设为集合 t，$t=\{t_{i1}, t_{i2}, \cdots, t_{ij}, \cdots, t_{in}\}$，其中 i 为执行失败测试用例编号，j 为语句行号，n 为总语句数，t_i 为第 i 个执行失败测试用例的执行情况，t_ij 为第 i 个执行失败测试用例的第 j 条语句的覆盖情况，覆盖情况取值为 0 或 1，0 为未被覆盖语句，1 为被覆盖语句 b。频谱增强 $E=\{e_1, e_2, \cdots, e_l, \cdots, e_n\}$，其中：

$$e_l=\begin{cases}(1, \forall j \in \{1,\cdots,m\}, & t_{jl}=1\\ 0, & \text{其他}\end{cases} \tag{1}$$

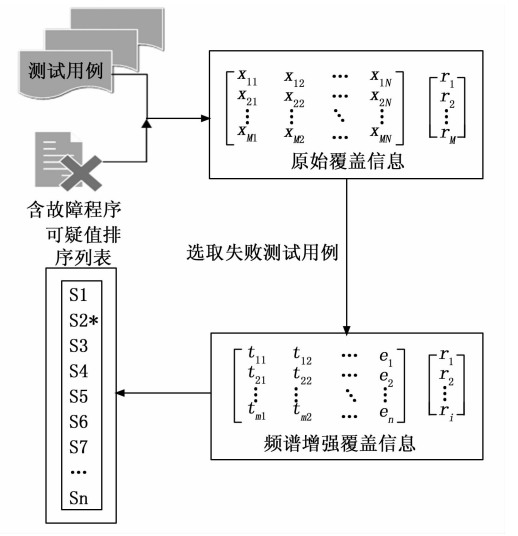


图 3　SBFL（E）流程图

以表 2 中多故障程序 Mid 为例，执行失败测试用例数为 4，语句总数为 13，故障，有 4 个失败执行覆盖 t_6，t_7，t_8 和 t_9，其中以 t_6 为例，$t_6=\{1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1\}$。利用频谱增强的定义，通过频谱增强方法可以得到该例子中 $E_1=\{e_1, e_2, \cdots, e_l, \cdots, e_n\}$。

4　实验结果与分析

为验证本文方法的有效性，实验比较了 SBFL（未经过频谱增强）和 SBFL（E）（经过频谱增强）的故障定位性能。实验运行环境为 Windows10 和 Ubuntu 系统，2.60 GHz Intel（R）i5 四核处理器，16 GB 物理内存。

4.1　实验数据集

我们采取基准程序进行了 5 个案例研究，其中 3 个程序是西门子（Siemens）程序套件的一部分，其余两个程序来自 Defects4j，数据集的详细信息在表 3 中列出。基于人工故障的西门子程序选取了 Tcas，Totinfo 和 Printtokens2；基于真实故障的 Defects4j 程序选取了 Apache Common Lang 和 JFree Chart。这些 C 语言（Siemens）和 JAVA（Defects4j）程序被广泛用于软件测试和故障定位的实验中，也用于我们之前的研究工作当中。

表 3　数据集

名称	错误版本数	代码行数	测试用例数
Tcas	41	178	1 578
Totinfo	23	406	1 052
Printtokens2	10	510	4 115
Chart	26	7 057	2 205
Lang	65	1 731	2 245

所有基准程序的单故障版本可以分别从 SIR（software-artifact infrastructure repository）和 Defects4j 资源库下载。在多故障版本中，故障是手动播种到原始程序中的。由于大多数程序都是单故障，我们将单故障版本结合起来，建立多故障版本。Rui[22] 和 Zakari[19] 也创建了包含 2、3、4 和 5 个故障的多故障版本，以产生 Siemens-M 案例研究。

4.2 评价指标

在本文中，我们采用 *EXAM* 和 Top-*N* 用于评估我们提出的 SBFL（E）方法的性能的指标，并将其与其他故障定位技术的有效性进行比较。

根据 Wong 等人提出的评价标准[23]，我们的研究采用 *EXAM* 作为评价指标，其定义为发现程序中所有故障时需要检查的代码行数占程序总代码行数的百分比。具体公式为：

$$EXAM = \frac{\sum_{1 \leqslant i \leqslant n} exam_i}{N} \times 100\% \tag{2}$$

其中：$exam_i$ 代表检测到第 i 个故障时需要检查的代码行数，n 代表程序中的故障总数，N 代表程序中的代码行总数。这种评价指标适用于单故障定位和多故障定位。当用 *EXAM* 分数来评价前者时，可以表示为 $EXAM = \frac{exam}{N} \times 100\%$，即需要检查的代码行数占代码行总数的百分比。根据 Ghosh 等人[24] 的研究，他们通过 3 个指标 $EXAM_F$，$EXAM_M$，以及 $EXAM_L$ 来扩展 *EXAM* 分数。$EXAM_F$ 代表在查找到第一个故障之前需要检查的语句的百分比；$EXAM_M$ 表示查找到第二个故障之前需要检查的语句的百分比；$EXAM_L$ 表示最后一个故障需要检查的语句的百分比。

Top-*N* 用于评估故障元素的绝对排名，它通过检查排名列表的前 n 个位置来计算成功定位的故障总数。N 越小，故障定位技术就越有效。值得注意的是，程序员在实践中会更多地关注最可疑的元素，他们不是一个一个地检查排名列表中的语句，而是在排名列表中的不同位置之间表现出某种形式的跳跃，直到关于故障原因的假设得到确认[25]。此外，排名靠前的位置很少被程序员跳过，所以 Top-1 对于评估故障定位技术的有效性和准确性极为关键。

4.3 结果分析

基于上述提出的方法，本节对实验进行了详细描述和分析。我们建立了一个名为 SBFL（E）的故障定位技术，它通过频谱增强来优化 SBFL，为了评估我们方法的有效性，我们采用了 4 个常用的 SBFL 技术作为基准。

在我们的实验研究中，我们执行了 5 个基准程序，包括 3 个西门子套件和两个 Defects4j 实用程序，每个程序中都包含一个故障。图 4 显示了 Dstar，Jaccard，Ochiai 和 Tarantula 对 3 个西门子程序（Tcas，Totinfo 和 Printtokens2）的推广技术的比较；Defects4j 数据集中 Chart 和 Lang 的整体表现也可以在图 5 中见证。在每张图中，横轴表示 *EXAM* 分数的百分比，而纵轴表示能检测到存在故障版本的百分比。

从图 4 和图 5 的结果分析中，我们发现由频谱增强的技术在寻找单一故障方面比现有的技术表现得更好，尤其是 Ochiai（E）方法 *EXAM* 在分数为 15%～20%时表现突出，它能够定位 90%以上的故障版本。Jaccard（E）和 Dstar（E）也在图 5 中的相同范围内保持了类似的趋势，能够定位大约 95%的故障版本。

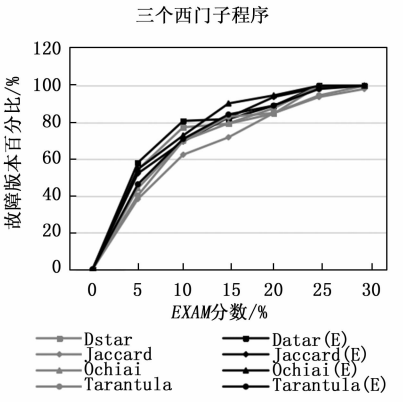


图 4 单故障西门子程序 *EXAM* 指数

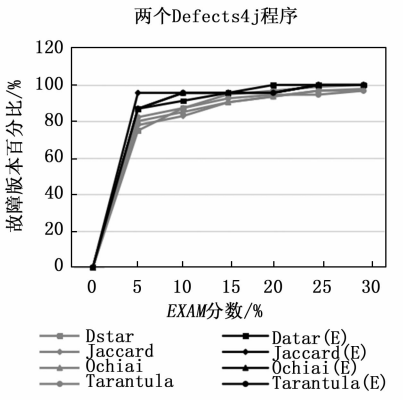
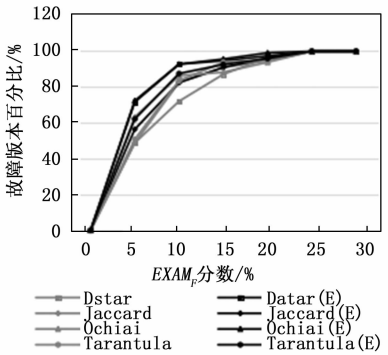
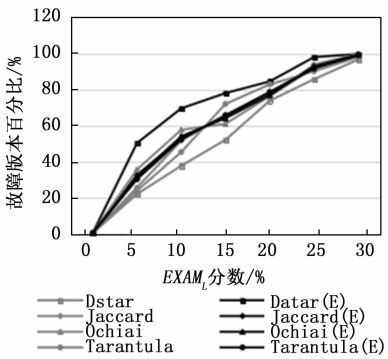


图 5 单故障 Defects4j 程序 *EXAM* 指数

我们在单故障程序中手动添加了一个故障，从而得到双故障程序。我们执行每个有两个故障的基准程序，计算了 $EXAM_F$ 和 $EXAM_L$ 值，以评估我们技术的准确性，如图 6 和 7 所示。并介绍了 4 种技术对西门子程序的 $EXAM_F$ 和 $EXAM_L$ 的比较。同样，对于 Defects4j 程序，4 种技术的比较见图 8 和 9。从结果分析中，我们可以看到，所提出的技术 SBFL（E）在大多数情况下都优于传统 SBFL。由于确保频谱增强技术确实在多故障程序中起到了优化作用，我们在每个西门子和 Defects4j 基准程序中手动添加到 3 个故障，准备了三故障程序来进行进一步实验研究。图 10～12 分别给出了 4 种技术对西门子程序的 $EXAM_F$，$EXAM_M$，以及 $EXAM_L$ 值的比较。对于 Defects4j 程序，类似的比较如图 9 所示。综合来看，我们提出的技术比现有的 SBFL 技术表现得更好。

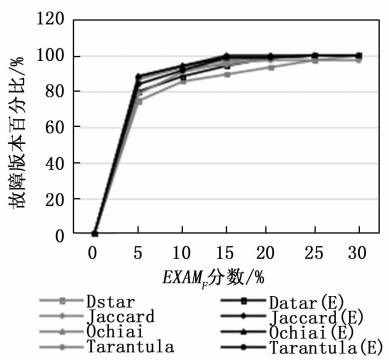


(a)三个西门子程序

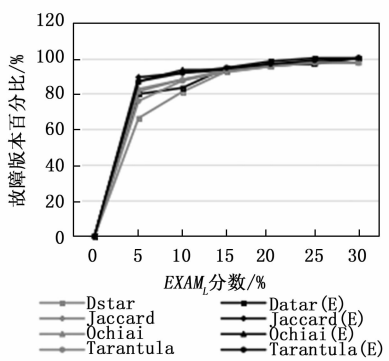


(b)三个西门子程序

图 6　双故障西门子程序 $EXAM_F$ 和 $EXAM_L$ 指数

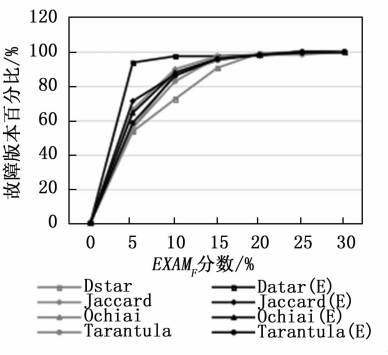


(a) 两个Defects4j程序

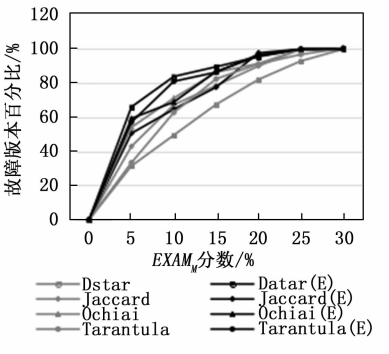


(b) 两个Defects4j程序

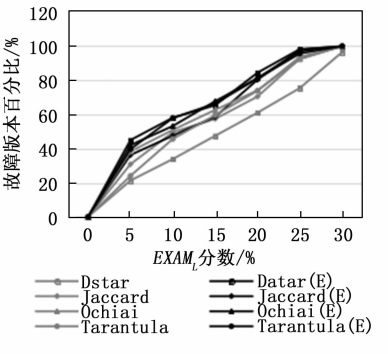
图 7　双故障 Defects4j 程序 $EXAM_F$ 和 $EXAM_L$ 指数



(a)三个西门子程序



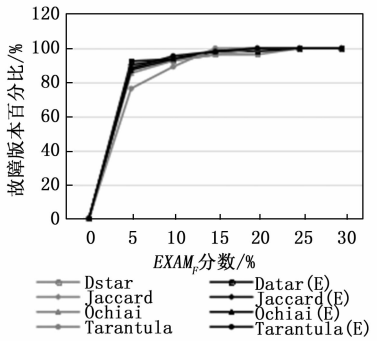
(b)三个西门子程序



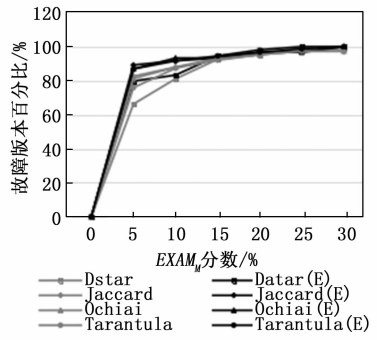
(c)三个西门子程序

图 8　三故障西门子程序 $EXAM_F$、$EXAM_M$ 和 $EXAM_L$ 指数

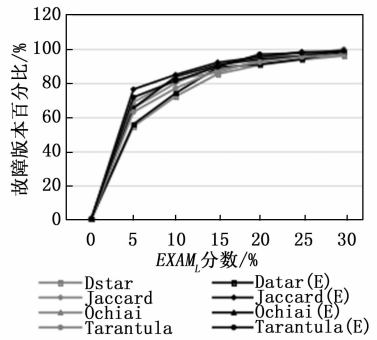
为了评估他们对顶级可疑元素的表现，使用了 Top-n 作为测量指标。95%的审查都是从排序列表中的第 1 个可疑元素开始的，这意味着首位很少被程序员跳过。因此，如何提高故障定位的准确性，特别是 Top-1 可疑元素，对故障定位的实用性至关重要。如表 4 所示，SBFL（E）技术通过检查单故障程序中的 Top-1、Top-3 和 Top-5 可疑语句，分别成功定位了数量更多的故障。表 4 突出了西门子程序的性能，每一种技术都得到了不同程度的改善，其他采用频谱增强技术在 Top-1 上比传统技术可以多定位到约 20%以上的故障，在 Top-3 和 Top-5 上都有同比增长。同样，表 4 也显示了 Defects4j 中同样的 8 种技术进行比较，所有的 SBFL 技术的效率都因为频谱增强所提升。最明显的变化是 Dstar（E）在重要的 Top-1 上，故障定位数量从 49



(a)两个Defects4j程序



(b)两个Defects4j程序



(c)两个Defects4j程序

图 9 三故障 Defects4j 程序 $EXAM_F$、$EXAM_M$ 和 $EXAM_L$ 指数

增加到 61。

表 4 单故障程序中 Top-N 指数名称

名称	Siemens			Defects4j		
	Top-1	Top-3	Top-5	Top-1	Top-3	Top-5
Dstar	22	39	42	49	64	64
Dstar(E)	25	41	48	61	62	62
Jaccard	22	38	42	56	60	60
Jaccard(E)	28	34	48	61	62	62
Ochiai	28	39	44	57	64	64
Ochiai(E)	29	40	48	61	62	62
Tarantula	24	40	44	55	56	60
Tarantula(E)	33	45	49	61	62	62

在多故障程序中，Top-N 在 SBFL 和 SBFL（E）技术之间展示了不同程度的增长。如表 5 所示，在双故障的西门子程序中，Dstar（E）通过检查前 1、3 和 5 条可疑语句，可以分别定位 24、53 和 66 个故障（约占所有故障的 20.3%、44.9%和 55.9%）。Dstar（E）和 Tarantula（E）保持着类似的趋势，且表现比 Ochiai（E）好。如表 5 所示，在 3 个故障的西门子程序中，Dstar（E）通过检查前 1、3 和 5 条可疑语句，分别可以定位 36、67 和 88 个故障（约占所有故障的 19.6%、36.4% 和 47.8%）。Jaccard（E），Ochiai（E）和 Tarantula（E）的存在相同的趋势，且差异较小。如表 5 和 6 所示，在多故障 Defects4j 程序中的优势更为显著，SBFL（E）在 Top-1，Top-3 以及 Top-5 中都保持较好的性能，尽管在少数情况下，与传统的 SBFL 技术相比，它未能定位更多的故障。

表 5 双故障程序中 Top-N 指数名称

名称	Siemens			Defects4j		
	Top-1	Top-3	Top-5	Top-1	Top-3	Top-5
Dstar	24	53	66	64	86	92
Dstar(E)	29	58	75	71	85	89
Jaccard	18	53	61	66	87	92
Jaccard(E)	22	58	65	72	85	89
Ochiai	23	52	66	64	87	90
Ochiai(E)	24	58	59	69	88	90
Tarantula	20	50	59	58	82	91
Tarantula(E)	27	61	70	67	84	90

表 6 三故障程序中 Top-N 指数名称

名称	Siemens			Defects4j		
	Top-1	Top-3	Top-5	Top-1	Top-3	Top-5
Dstar	22	62	71	48	76	95
Dstar(E)	36	67	88	66	113	137
Jaccard	16	52	70	48	79	98
Jaccard(E)	35	60	74	67	117	141
	17	53	66	50	78	96
Ochiai(E)	40	68	84	67	120	140
Tarantula	12	47	64	41	74	93
Tarantula(E)	37	76	93	67	111	133

5 结束语

故障定位是困难且代价高的，这就是为什么我们在过去 20 年中看到了各种故障定位技术被提出。多故障定位正在成为软件测试研究领域中最关键的问题之一。在实践中，我们发现在传统的 SBFL 中冗余的失败测试用例会给故障定位效率带来消极影响，这阻碍了程序员在调试故障时的效率和生产力。此外，我们注意到使用频谱增强的逻辑与运算可以有效精简失败测试用例的频谱信息，优化可疑度值以提高故障元素的等级。

在本文中，我们提出了基于频谱增强的多故障定位方

法。首先，使用频谱增强的方法精简失败的测试用例，再利用 SBFL 的统计公式对新的频谱信息进行评估，并根据其可疑度对潜在的故障语句进行排序。最后，被 SBFL（E）排在前 N 位的语句根据其新的分数排序。所提出的故障定位方法缓解了 SBFL 的缺点，提高了故障定位的准确性。通过使用 5 个开源基准程序的案例研究来评估我们框架的性能。为了显示我们提出的技术的效率，将我们的工作与 4 个密切相关的故障定位技术，即 Dstar、Jaccard、Ochiai 和 Tarantula 进行了比较。实验结果表明，与典型的 SBFL 定位方法相比，基于频谱增强的方法可以显著减少代码检查的范围，尤其是在高性能范围内（即 $EXAM5\%$ 和 $Top\text{-}1$），明显提高了故障定位的性能。

但是在这个领域还有很大的发展空间，也有一些问题需要解决。存在程序中如果失败测试用例过少的问题，而测试用例质量及其数量对结果有很重要的影响；以及频谱增强的思想能否运用到其他故障定位方法当中。后期研究人员还需要努力减少其负面影响，甚至解决这些问题。虽然本实验选择的基准程序在故障定位领域具有代表性，具有说服力，但在今后的工作中，我们将把重点放在更真实的工程程序上。总的来说，随着研究界对 SFL 研究的极大兴趣，以及最近该领域出版物的一致性，我们希望在未来几年会有更多具体的解决方案来解决多故障问题。

参考文献：

[1] JONES J A，HARROLD M J，STASKO J. Visualization of test information to assist fault localization [C] //ICSE. ACM，2002：467－477.

[2] RUI A，ZOETEWEIJ P，GEMUND A. On the accuracy of spectrum-based fault localization [C] // TAICPART-MUTATION. IEEE，2007：89－98.

[3] JONES J A，HARROLD M J. Empirical evaluation of the tarantula automatic fault-localization technique [C] //Proc. 20th IEEE/ACM Int. Conf. Autom. Softw. Eng.（ASE），2005：273－282.

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

[5] YOO S. Evolving human competitive spectra-based fault localization techniques [C] //SSBSE，Springer，2012：244－258.

[6] XIE X Y，KUO F C，CHEN T Y，et al. Provably optimal and human-competitive results in SBSE for spectrum based fault localization [C] //SSBSE，Springer，2013：224－238.

[7] MAO X，YAN L，DAI Z，et al. Slice-based statistical fault localization [J]. Journal of Systems & Software，2014，89：51－62.

[8] XUAN J，MONPERRUS M. Test case purification for improvingfault localization [C] //FSE. ACM，2014：52－63.

[9] MIN F，GUPTA R. Learning universal probabilistic modelsfor fault localization [C] //PASTE. ACM，2010：81－88.

[10] XUAN J，MONPERRUS M. Learning to combine multiple ranking metrics for fault localization [C] //ICSME. IEEE，2014：191－200.

[11] YOO S，MOON S，KIM Y，et al. Ask the mutants：Mutatingfaulty programs for fault localization [C] //ICST. IEEE，2014：153－162.

[12] ZHANG L，ZHANG L，KHURSHID S. Injecting mechanical faultsto localize developer faults for evolving software [C] //OOPSLA，2013：765－784.

[13] PAPADAKIS M，TRAON Y L. Using mutants to locate unknown faults [C] // ICST. IEEE，2012：691－700.

[14] LI X，ZHANG L. Transforming programs and tests in tandemfor fault localization [C] //OOPSLA. ACM，2017：92－122.

[15] WONG W E，DEBROY V，GAO R，et al. The DStar method for effective software fault localization [C] // IEEE Transactions on Reliability，2013，63（1）：290－308.

[16] 余晓菲，张　仕，蔡　蕊，等，基于改良程序谱的软件故障定位方法 [J]. 计算机工程与科学，2018，40（2）：275－281.

[17] XIE X，CHEN T，KUO F，et al. A theoretical analysis of the risk evaluation formulas for spectrum-based fault localization [J]. ACM Transactions on Software Engineering and Methodology（TOSEM），2013，31：1－31，40.

[18] ZOU D，LIANG J，XIONG Y，et al. An empirical study of fault localization families and their combinations [J]. IEEE Transactions on Software Engineering，2021，47（2）：332－347.

[19] ZAKARI A，LEE S P，ABREU R，et al. Multiple fault localization of software programs：a systematic literature review [J]. Information and Software Technology，2020，124：106312.

[20] YAN Z，ZAN W，FAN X，et al. Localizing multiple software faults based on evolution algorithm [J]. Journal of Systems and Software，2018，139：107－123.

[21] ZAKARI A，LEE S P. Simultaneous isolation of software faults for effective fault localization [C] // 2019 IEEE 15th International Colloquium on Signal Processing & Its Applications（CSPA），IEEE，2019：16－20.

[22] RUI A，ZOETEWEIJ P，GEMUND A. Simultaneous debugging of software fault [J]. Journal of Systems and Software，2011，84（4）：573－586.

[23] WONG W E，WEI T，YU Q，et al. A crosstab-based statistical method for effective fault localization [C] // 2008 1st International Conference on Software Testing，Verification，and Validation，IEEE，2008：42－51.

[24] GHOSH D，SINGH J. Spectrum-based multi-fault localization using chaotic genetic algorithm [J]. Information and Software Technology，2021，133：106512.

[25] CUI Z，JIA M，CHEN X，et al. Improving software fault localization by combining spectrum and mutation [C] // IEEE Access，2020，8：172296－172307.