突变是测试实验的合适工具吗?

J.H.安德鲁斯
计算机科学系
西安大略大学
伦敦,加拿大
andrews@csd.uwo.ca

L.C. Briand Y. Labiche 软件质量工程实验室
加拿大渥太华卡尔顿大学系统与计算机工程系
{briand, labiche} @sce.carleton.ca

摘要
测试技术的实证评估在软件测试研究中占有重要的地位。一种常见的做法是检测故障，或者手工检测，或者使用变异算子检测。后者允许系统地、可重复地播种大量故障;然而，我们不知道这种方式获得的实证结果是否能得出有效的、有代表性的结论。本文基于一些具有全面测试用例池和已知故障的程序来研究这个重要的问题。结论是，基于迄今为止可用的数据，使用变异操作符正在产生值得信赖的结果(生成的突变类似于真实的故障)。然而，突变体似乎与手种故障不同，手种故障似乎比真实故障更难检测。

类别和主题描述符 D.2.5【测试与调试】

通用术语实验、验证。

关键字
真实断层，手种断层，突变体

1.介绍
实验是软件测试研究中必不可少的一部分。通常，实验被用来确定两种或两种以上的方法中哪一种对执行一些与测试相关的活动是更好的。例如，一个人可能对比较几个测试标准的故障检测效果感兴趣，用来导出测试用例，一个人求助于实验来达到这个目的。测试实验通常需要一组具有已知故障的主题程序。这些课题程序应该大到符合实际，但又不能大到让实验无法进行。至于错误，一种技术处理给定错误的能力应该是该技术在实际程序中的性能的准确预测器。

2005 年 5 月 15-21 日，美国密苏里州圣路易斯市，ICSE '05。

测试实验设计中的一个问题是，具有真实缺陷的适当大小的真实程序很难被发现，也很难进行适当的准备(例如，通过准备正确的和错误的版本)。即使有带有实际故障的实际程序可用，通常这些故障也不足以使实验结果达到统计显著性。因此，许多研究人员采取了在正确的程序中引入错误的方法，以产生错误的版本。

这些错误可以手工引入(例如，实验者可以请有经验的工程师来做)，或者通过自动生成代码的变体来引入。一般来说，我们把自动生成的变体看作是对代码应用运算符的结果。以这种方式使用的运算符称为突变运算符，产生的错误版本称为突变体，一般的技术称为突变或突变生成。

突变生成的主要潜在优势在于，突变算子可以被精确地描述，从而提供一个定义明确的、故障播种的过程。这有助于研究人员复制他人的实验，这是良好的实验科学的必要条件。虽然手工引入的错误可以被认为更现实，但最终它是一个主观判断，一个给定的错误是否现实。突变体生成的另一个重要优势是，可以产生潜在的大量突变体，增加获得的结果的统计显著性。

然而，一个重要的问题仍然存在。我们如何知道检测(或“杀死”)突变体的能力是否能准确预测实际表现;即在评估测试技术的故障检测效果时，突变体的外部效度是多少?这个问题的答案将对我们如何进行测试实验，从而对实验结果的有效性产生重要的影响。

本文的主要贡献是比较测试套件对手工播种、自动生成和真实世界的故障的检测能力。我们的主题程序是一组被广泛使用的带有手工种子故障的程序，以及一个被广泛使用的带有真实故障的程序。我们使用一组来自突变测试文献的标准突变算子，从主题程序中生成突变。我们的结果表明，生成的突变体与真实故障相似，但与手种故障不同，而且手种故障比真实故障更难检测。

2.相关工作

使用突变体来衡量测试套件充分性的想法最初由 DeMillo 等人[6]和 Hamlet[12]提出，并被 Offutt 等人[20]广泛探索。Offutt[17]展示了对突变测试的一个基本前提的经验支持，即检测简单故障(如由突变引入的故障)的测试数据集将检测复杂故障，即。，几个简单故障的组合。

Frankl 和 Weiss[10]、Thévenod-Fosse 等人[24]、Hutchins 等人[14]等人使用程序的错误变体进行了实验，此后也有许多研究人员进行了实验。Frankl 和 Weiss 使用了 9 个 Pascal 程序，每个程序都有一个现有的错误，而 Hutchins 等人则在使用的 7 个程序中手工播种了 130 个错误。Thévenod-Fosse 等人使用变异算子在四个小 C 程序中自动播种故障。一般来说，这些实验遵循这样的模式:生成一个测试用例的大型“测试池”，在测试池中的所有测试用例上运行所有有缺陷的版本，观察哪些测试用例检测到哪些故障，并使用这些数据推断从池中抽取的给定测试套件的故障检测能力(例如，满足特定覆盖标准的测试套件)。

尽管突变生成最初是作为测试策略的一部分提出的，但请注意 Thévenod-Fosse 等人将其作为一种为实验生成错误版本的方法。其他做过这方面工作的研究人员包括 Kim et al.[15]、Memon et al.[16]、Andrews and Zhang[1]以及 Briand and Labiche[2]。

最后，Chen et al.[5]在他们的实验中同时使用了手工播种的断层和生成的突变体。他们指出，手播故障只是可能故障的一个子集，并提出了故障是否具有代表性的问题，但由于这不是他们研究的重点，他们没有进一步探索。

综上所述，据我们所知，还没有任何实证研究通过与真实断层的结果进行比较，直接评估了突变体或手种断层的使用情况。

3.实验材料和过程

在这一节中，我们描述了我们执行的实验，主题程序和与它们相关的工件(错误版本和测试用例)，以及我们用来产生突变的突变操作符。

在本文的其余部分，我们使用术语手工播种错误(例如，由经验丰富的工程师产生)和真实错误(例如，在软件开发过程中发现的)，或者只是在没有歧义的情况下出现的错误，而不是由被称为突变体的突变操作产生的错误。

表 1 中总结了主题程序的属性。这里的“NLOC”是代码的净行，即。，删除注释后非空白的代码行。“# conditionals”是 C 条件结构(if, while, case 等)和二进制逻辑操作符(&&和||)的数量，作为代码复杂度的粗略指标。

3.1 实验的定义

为了实现引言中提到的目标，我们分析了测试套件的检出率。我们使用八个主题程序(表 1)，其中故障和大型测试用例池是可用的:注意那些测试池是可比性的，因为它们确保几个结构标准得到满足(第 3.2 节)。然后通过对每个主题程序的大型测试池的随机抽样形成测试套件(第 3.4 节)。然后我们创建这些程序的突变版本(章节 3.3)，并在可用的错误版本和突变上执行这些测试套件，记录被杀死的故障，以计算测试套件的故障检测比率。这些比率在突变体和故障之间进行比较，以确定它们是否相似。因为每个测试套件可能会产生不同的检测比，我们为每个主题程序获得两个分布，分别为故障和突变体。通过统计推断检验来比较这些分布的中心趋势。零假设(H_0因此，可以这样表述:在一组故障和一组突变体之间，检测率没有差异。备择假设(H_a(h .)是错误和突变体之间的检测率存在差异。断层和突变分布之间的差异必须根据现有的数据来解释，而且各学科项目之间结果的可能差异也需要进行调查。

3.2 课题项目

我们使用了一套 8 个用 C 语言编写的知名主题程序(表 1)。第一个是通常被称为 Space 的程序，由欧洲航天局开发，最初由 Frankl 等人用于测试战略评估目的[9,25]。后 7 个是所谓的西门子手播故障程序套件，最早由 Hutchins 等人[14]使用，用于比较基于控制流和基于数据流的覆盖标准。所有 8 个程序的工件(包括有缺陷的版本和测试用例)随后被其他研究人员修改和扩展，特别是 Rothermel 和 Harrold[23]和 Graves 等人[11]。我们之所以选择这些程序，是因为它们的成熟度

表 1。学科项目描述

	主题项目							
标准	空间	Printtokens	Printtokens2	取代	时间表	Schedule2	见面会上,	Totinfo
NLOC	5905	343	355	513	296	263	137	281
#条件	635	94	81	99	37	51	25	46
测试池大小(#测试用例)	13585	4130	4115	5542	2650	2710	1608	1052
故障个数(版本号)	38	7	10	32	9	10	41	23
编译突变体数量	11379	582	375	666	253	299	291	516

相关的文物，并因为它们的历史意义:Do et al.[8]报告称，在过去十年中，17 篇备受瞩目的实验软件工程论文使用了西门子套件和/或空间。

Space 与 38 个有缺陷的版本(真正的缺陷)有关，33 个来自最初的 Vokolos 和 Frankl 研究，5 个后来被其他研究人员检测到。每个有故障的版本对应一个“在测试和程序的操作使用中”被纠正的故障[25]。修正了所有错误的“黄金”版本被视为产生正确的输出。相比之下，西门子套件程序的错误版本是手工制作的，“由十个不同的人制作，大多数人不知道彼此的工作;他们的目标是生产出尽可能逼真的错误“[14]”。

每个主题程序的测试池(所有测试用例的集合)是由最初的研究人员和后续的研究人员按照各种功能(黑盒)测试技术和结构测试覆盖标准构建的。Harder 等人[13]给出了这些测试池构建的全面历史。西门子程序的测试池是这样开发的:黑盒技术分类-划分[21]用来创建第一组测试用例，然后根据几个结构覆盖标准(所有语句、所有边、所有定义-使用对)对测试用例进行扩展。程序 Space 测试池的构建遵循类似的过程，只是原始测试集是通过随机输入选择生成的。

3.3 变异算子

为了生成主题程序的突变体，我们使用了 Andrews 和 Zhang[1]首先使用的突变体生成程序，为用 c 编写的代码生成突变体。要从源文件生成突变体，每一行代码都按顺序考虑，并分别应用四类“突变算子”(只要可能)。换句话说，突变算子对一行代码的每一次有效应用都会导致产生另一个突变体。突变算子的四类分别是:

-用 0,1，-1，((C)+1)或((C)-1)替换整数常量 C。-将一个算术、关系、逻辑、位逻辑、递增/递减或算术赋值操作符替换为来自同一类的另一个操作符。

-对 if 或 while 语句中的判定求反。

-删除语句。

前三个操作符类取自 Offutt 等人关于识别一组“充分”突变操作符的[18]研究，即。，一组 S 的操作符，这样的测试套件杀死由 S 形成的突变体，往往会杀死由非常广泛的操作符类形成的突变体。它们经过了调整，以便它们可以用于 C 程序，而不是原始研究中的 Fortran。第四个操作符也出现在[18]中，之所以加入，是因为在突变生成程序首次使用的原始研究[1]的主题程序中，包含了大量的指针操作和字段赋值语句，这些语句不会受到任何足够的突变操作符的攻击。

由此产生的突变体中，约有 8.4%没有编译。每个被编译的主体程序的突变体数量出现在表 1 中。对于 Space 项目来说，产生的突变体太多了，要在测试套件上全部运行是不可行的。

因此，我们每 10 个就运行一次测试套件 th 产生的突变体。因为每行产生的突变体数量并没有遵循任何会与每 10 个选择相互作用的模式 th 突变体(这只取决于系谱上的构造)，这相当于随机选择 10%的突变体，从所有可能突变体的均匀分布中取出。此外，这确保了整个源代码都被植入了错误(而不仅仅是几个函数/过程)。

3.4 分析过程

在本节中，我们对所使用的分析程序进行了简要的描述和论证。更多的细节将在我们报告结果的下一节中呈现。

第一步是生成并编译突变体，并在整个测试池中运行所有突变体和错误版本。然后，所有未被任何测试用例杀死的突变体被视为与原始程序等价。虽然这可能不是每一个突变体的情况，但它被认为是一个足够好的近似，在任何情况下，当处理大量突变体时，它是唯一的选择，因为自动识别等效突变体是一个不可判定的问题[3,6,19]。

对于每个主题程序，通过对可用的测试池进行随机抽样(无重复)，随机形成 5000 个测试套件。我们需要生成大量的测试套件，以获得能够很好地近似底层理论分布的故障/突变检出率的样本分布。正如下文进一步描述的那样，大量的样本也增加了统计测试的能力，并促进了它们的使用。随机选择而不是由覆盖标准(例如，结构)驱动的选择被认为提供了足够的可变性。要做的一个决定与测试套件的大小有关。我们希望这在我们的分析中是一个常数，这样就不会模糊我们所分析的趋势:例如，更好的故障/突变检测效率可以简单地归因于测试集的大小，而不是(可能的)故障和突变之间的差异。测试集中太少的测试用例会导致小的故障/突变检出率，并会阻止我们观察差异，而测试集中太多的测试用例会减少可变性。我们以不同的规模执行我们的分析，以确定它将如何影响结果，范围从 10 个到 100 个测试用例。由于对不同的测试套件大小没有观察到主要的差异，并且 100 的大小导致了在测试套件中具有良好可变性的合理的故障/突变检出率(100 对应于最小测试池的 10% -表 1)，我们决定只报告规模为 100 的测试套件的结果。

然后，我们确定池中的哪个测试用例检测到哪个突变和故障。接下来，我们计算所有测试套件的故障检测率，绘制出每个主题程序的突变体和故障检测率分布，然后比较它们的中心趋势。

总结一下，对于每个测试套件 S，程序产生了两条汇总数据:S 检测到的突变体数量 Dm(S)和 S 检测到的故障数量 Df(S)。给定主题程序的非等效突变体数量 Nm 和非等效故障数量 Nf，我们计算每个测试套件 S 的突变检测比 Am(S)为 Dm(S)/Nm，故障检测比 Af(S)为 Df(S)/Nf。

我们想先检验零假设 $H_0(P)$对每个主题程序 P:即 Am 和 Af 比率的均值

P 是一样的。为了做到这一点，我们采用了标准的统计检验:配对 t 检验[22]，原因将在第 4 节中进一步详细说明[1]。如果我们拒绝 H_0 因为断层分布和突变分布之间的平均值差异是统计上和实际上的[2]意义重大，那么接下来的问题是为什么?此外，如果不同项目的结果不一致，我们还需要确定最合理的解释是什么。有许多可能的原因，包括主题程序的特征差异、测试套件和错误产生的方式。所有主题程序唯一的共同点是我们生成突变体的方式。

为了解释我们的结果，以及在第 4 节中进一步解释的原因，我们还查看了杀死每个错误和突变的测试用例的百分比。对于每个突变体 M 和错误版本 F，我们计算了 K(M)(对应。K(F))，即杀死它的测试用例的数量。然后我们计算，对于每个突变体 M，杀死突变体的容易程度 E(M)为比率 E(M) = K(M)/T，其中 T 为程序的测试用例总数。(我们以类似的方式计算每个错误版本的 E(F)。)然后，我们得到了每个观测值对应一个突变或故障的分布，我们可以在各个主题程序中比较这些分布的平均值。

在分析方面，我们首先进行了方差分析(ANOVA)，以评估程序平均值之间差异的总体统计显著性。如果显著，然后我们使用独立样本的 t 检验来比较每个(Space, Siemens Program)对的分布平均值。为此，我们求助于 Bonferroni 程序[22]，以确保在执行大量比较时 I 型错误的低风险:在我们的案例中为 28 个比较[3]。因为我们这里处理的是较小的样本(样本大小是由数量决定的

[1] 当处理大数据集时(根据经验，超过 100 次观察)，这个检验对于配对 t 检验(假设配对之间的差异是正态分布的)下的正态假设的偏离是非常稳健的，即使是在显示出极端分布偏态的情况下。这一点很重要，因为这种违反假设的情况在现实世界的数据集中很常见。还要注意的是，在我们的情况下，我们将不会在我们的观察中有极端的异常值，因为我们将处理 0 和 1 之间的比率，我们将有 5000 个观察值可供我们处理。然而，为了安全起见，我们使用了另一个等价的非参数检验(Wilcoxon match - pairs Signed Ranks test)来再次检查结果。一般来说，当使用大样本时，配对 t 检验也特别适合，因为它考虑了差异的大小，因此比等价的非参数检验更强大。

[2] 在处理大样本时，即使均值上的微小差异也能导致统计上显著的结果。然而，确定一个统计上显著的结果是否会有任何实际意义也很重要，即均值差异是否足够大，值得考虑。这就是通常所说的“实际”意义，有时也被称为“临床”意义。

[3] 我们对每对(Space, Siemens Program)进行 Am(S)和 Af(S)比较，从而得到(7*8)/2=14 个比较。同样，比较 E(M)和 E(F)会产生 14 个额外的比较。

突变体和错误，与上面的测试套件相反)，我们需要用一个等价的非参数检验，即曼-惠特尼检验(Mann-Whitney test)来检查不显著的结果[20,22]。的确，在小样本上，违反 t 检验假设会导致 II 型错误的增加，即错误地推断平均值的差异不显著。

在我们所有的统计检验中，我们都使用显著性水平 $\alpha = 0.05$，这意味着出现 I 型错误的概率为 5%，或者换句话说，在差异不显著时，将其识别为统计显著性的风险为 5%。

3.5 对有效性的威胁

本节讨论了对我们实验有效性的不同类型的威胁，按照优先级递减的顺序:内部效度、外部效度、结构效度和结论效度[4,26]。

与内部有效性相关的一个问题是，我们在没有任何修改的情况下重复使用了在以前的实验中广泛使用的程序。我们正在使用的程序、测试池和故障并不是基于任何特定的标准进行选择的，除了它们是准备充分的和历史上重要的。然而，我们不能保证测试池具有相同的检测能力和覆盖范围，尽管关于它们的文献报告似乎表明它们是以类似的方式生成的，而且是相当全面的，因为它们覆盖了一些主要的结构覆盖标准，如第 3.2 节所报告的。

对于 Space，我们有真正的错误，但对于其他的，我们只有经验丰富的工程师手工播种的“现实的”错误。我们也不能指望程序具有类似的复杂性，不管我们如何定义和衡量它。它们实际上在大小(表 1)和控制流复杂性上有很大差异。也可以预期，我们的研究结果会因所选择的变异算子而有所不同;我们所使用的算子，如 3.3 节中所讨论的，是基于可用的文献进行选择的，以便成为一个最小但充分的集合。

另一个可能的威胁涉及我们对所有程序的独特测试套件大小的报告，不管它们的复杂性，以及从测试池中随机选择的测试用例。然而，100 的大小结合随机选择被认为是足够的(足够高的检测比率和测试套件足够的可变性)。此外，使用了其他测试套件大小，并没有导致不同的结果，因此这里不做报告。

外部有效性与我们将实验结果推广到工业实践的能力有关。虽然我们只有程序空间的真实故障，但实验中使用的仪器都是真实的工业程序，具有真实故障或由经验丰富的工程师手工播种的故障。尽管如此，由于只使用了一个带有真实故障的程序，因此在其他已识别出真实故障的主题程序上复制这项研究将是非常重要的。

构造效度关系到我们定义测量的方式，以及它是否测量了我们真正想要捕获的属性:测试集的检测能力和故障的可检测性。这在上面讨论测试集的大小和它们的构造时得到了充分的证明。

结论有效性涉及主题选择，数据收集，测量可靠性，和统计检验的有效性。这些问题已经在实验设计中得到了解决，

表 2。描述性统计-突变体和故障的检测比率(测试套件大小为 100)

主题项目								
Statistic	Space	Replace	Printtokens	Printtokens2	Schedule	表 2	柠檬酸年代	Totinfo
突变体是(S)								
中位数	0.75	0.93	0.98	0.99	0.96	0.96	0.91	0.99
的意思是	0.75	0.93	0.97	0.99	0.96	0.96	0.90	0.99
90%	0.78	0.95	0.99	0.99	0.98	0.97	0.94	0.99
75%	0.77	0.94	0.98	0.99	0.97	0.97	0.93	0.99
25%	0.74	0.93	0.97	0.98	0.94	0.95	0.89	0.98
10%	0.72	0.92	0.96	0.98	0.93	0.95	0.86	0.98
最小值	0.65	0.88	0.94	0.93	0.91	0.91	0.77	0.94
马克斯	0.82	0.98	1	1	0.99	0.99	0.97	1
缺点- Af (S)								
中位数	0.76	0.68	0.57	0.90	0.67	0.67	0.76	0.65
的意思是	0.77	0.67	0.63	0.93	0.66	0.63	0.79	0.89
90%	0.82	0.77	0.86	1	0.78	0.78	1	0.96
75%	0.79	0.74	0.71	1	0.78	0.78	0.95	0.91
25%	0.74	0.61	0.57	0.90	0.56	0.56	0.68	0.87
10%	0.71	0.55	0.29	0.80	0.44	0.44	0.61	0.83
最小值	0.53	0.35	0.14	0.60	0.33	0	0.34	0.65
马克斯	0.97	0.93	1	1	1	1	1	1

虽然如果所有的程序都有手工播种和真实错误会更好。

我们的分析和解释必须考虑到上述所有因素，以确保所有似是而非的解释都被考虑在内。

4.分析结果

按照第 3.4 节中描述的过程，我们首先比较测试套件在突变和故障方面的检测比率(第 4.1 节)。在第 4.2 节中，我们确定了可能的现象，可以解释第 4.1 节中观察到的趋势。为了研究最合理的解释，我们然后比较跨程序杀死错误和突变的测试用例的百分比，从而观察它们的可检测性，即。，它们的可检测性(第 4.3 节和 4.4 节)。我们以讨论我们的结果的含义来结束这一节(第 4.5 节)。

的检测分布比较 突变体和错误

我们分析的第一步是比较每个主题程序的突变和故障检出率 Am 和 Af 的分布。目的是确定它们之间是否有统计上和实际上的显著差异。我们在表 2 中提供了每个主题程序的这些分布的描述性统计(具有两位数的精度)。请注意，这些是我们在选定测试套件时获得的结果

大小是 100。其他测试套件大小(10,20,50)也得到了类似的结果，因此这里没有报告。

从表 2 中，我们可以很容易地观察到，一般来说，突变往往比故障更容易检测。这一趋势的唯一例外是 Space，在这里，突变体和断层分布之间并不能观察到实际的显著差异(例如，它们的中位数分别为 0.75 和 0.76)。虽然由于空间的限制，不能在这里显示所有的分布，图 1 和图 2 显示了 space and Replace 程序的分布直方图，以及标准的分位数框图，以可视化分布的主要描述性统计。箱线图显示了中位数、平均值、各种分位数(1%、10%、25%、75%、90%、99%)，以及最小值和最大值。

我们现在要检查差异在统计上是否显著。为此，我们进行配对 t 检验[7,22]。回想一下，在这些分布中，我们比较了 5000 个测试套件的突变(Am)和故障(Af)检测率。在每个分布中，当观测结果配对时，它们很可能是相关的，即。，每个 Am 和 Af 分布中的一个观察结果对应于相同的测试套件。这就是为什么需要这样的配对测试而不是独立样本测试的原因。

表 3 显示了每个主题程序的 t 检验结果，即故障/突变检出率在配对之间的平均差异、t 比和 p 值(这是一个双向检验，因为我们事先无法确定差异的方向)。结果显示，由于所有的 p 值都接近或等于 0，所以所有的差异在统计上都是显著的。甚至 Space 的情况也是如此

虽然有 1%的平均差异，但这实际上并不显著，因为没有测试技术评估会基于如此小的差异得出不同的结论。对于其余的科目项目，平均差异在 6%到 34%之间，平均差异为 22%。与太空项目相反，这实际上意义重大，因为基于这样的范围，如果一个人使用突变体来评估一项测试技术，它可能会比使用种子错误更有效地检测故障。

然而，重要的是要记住，只有太空故障才是在测试和操作中检测到的真正故障。因此，Space 和其他程序观察到的差异的一个看似合理的原因是，后者中播下的故障比 Space 的真正故障更难检测。如果这被认为是最可信的原因，那么对 Space 故障的测试结果因此更可信，结论应该只基于这个程序的结果得出。

表 3。配对 t 检验结果-测试套件大小= 100

	配对结果		
主题项目	Mean Af(S) - Am(S)	*t-ratio*	*假定值*
空间	0.014	16.87	< 0.0001
取代	-0.266	-233.96	0.0000
Printtokens	-0.344	-158.2	0.0000
Printtokens2	-0.061	-59.39	0.0000
时间表	-0.298	-161.33	0.0000
Schedule2	-0.327	-152.19	0.0000
见面会上,	-0.1128	-57.56	0.0000
Totinfo	-0.1037	-145.78	0.0000

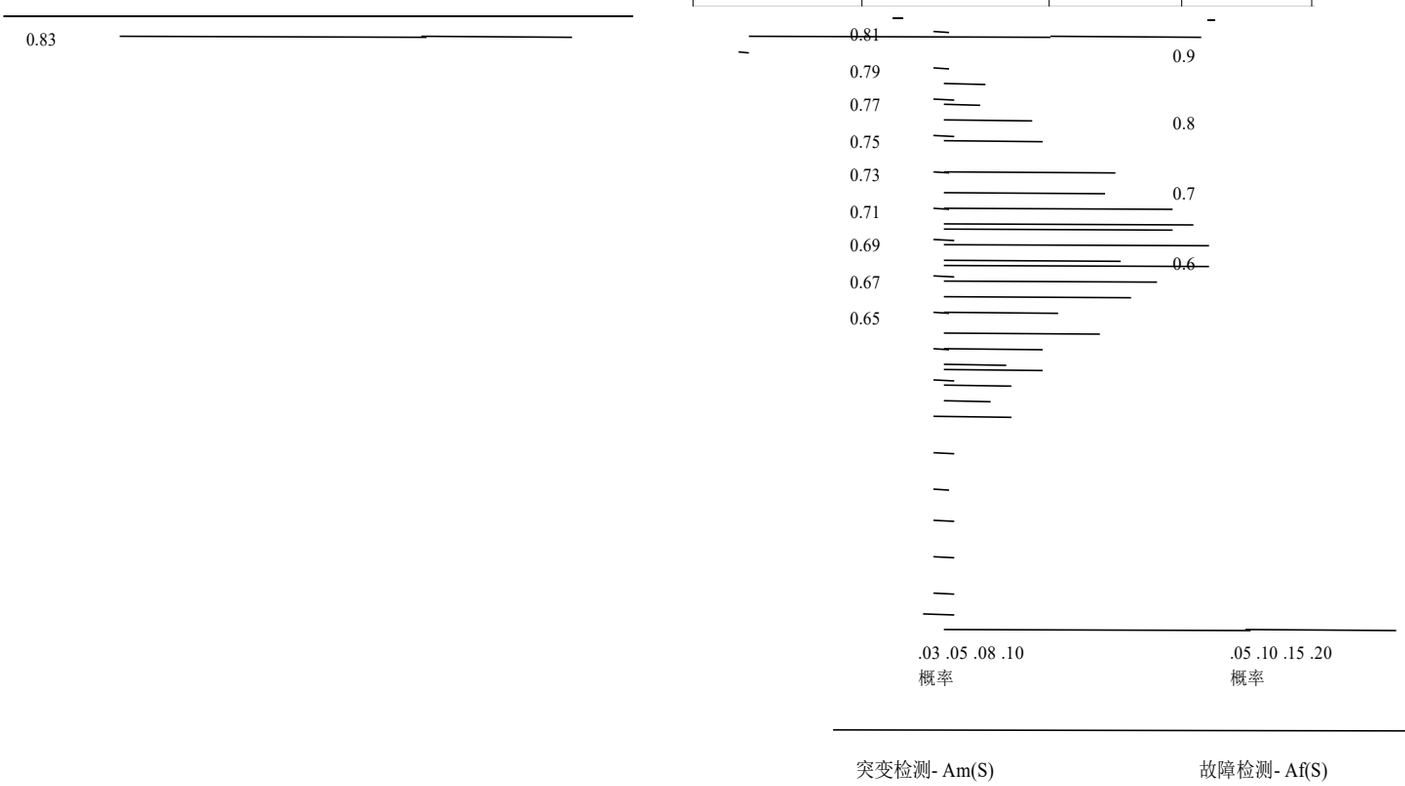

图 1 所示。主题程序空间:检测比分布-测试套件大小= 100

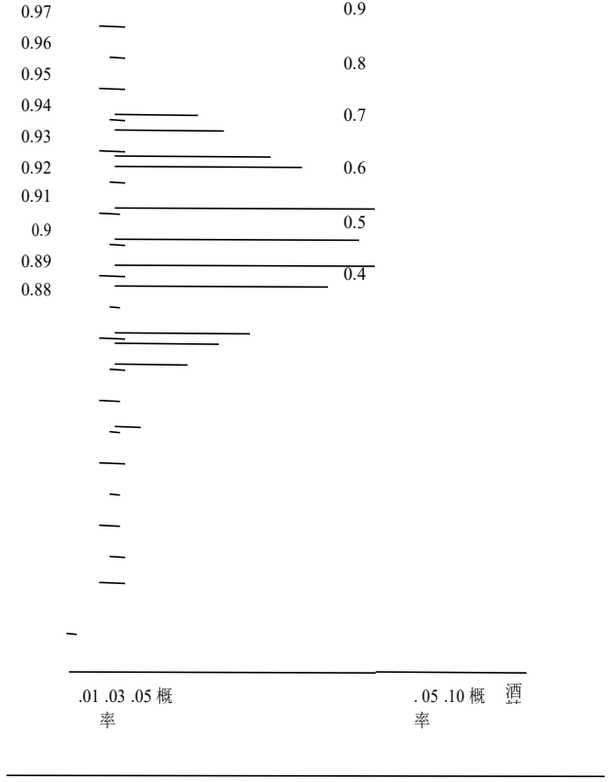

图 2。主题程序替换:检测比分布-测试套件大小= 100

4.2其他解释

除了真实的和种子故障的相对可检测性之外，其他几个因素也可以解释上一节中给出的数据:测试池、程序内部特征(例如，大小)、与测试池和程序的大小相比的测试大小套件，以及突变过程。为了评估我们的第一种解释是否确实是最可信的，依次考虑它们每一个都是很重要的。

测试池可以包含测试用例，平均而言，这些测试用例具有不同的检测能力。然而，关于这些程序[13]的文献表明，池是按照类似的过程和相同的结构标准形成的:所有节点，所有边，所有定义-使用对。此外，为了解释我们的结果，这将意味着平均测试用例检测能力将根据是否考虑突变体或故障而变化，当与西门子程序相比，对故障和突变体有一个中等排名的检测比率。

另一种可能性是，Am 和 Af 之间的差异随着测试套件规模的增加而增加，与测试池或程序的规模相比。(Space 要大得多，测试池大小也比其他程序大。)如果是这样的话，那么随着测试套件尺寸的减少，我们应该会看到西门子程序的 Am-Af 更小。然而，对于较小的 10、20 和 50 的测试套件，结果是相似的:Space 的平均(Am-Af)保持相似，甚至随着我们使用的较小的测试套件尺寸，其他程序的平均(Am-Af)甚至有增加的趋势。附录中的图表比较了四个测试套件尺寸的 Am 和 Af，对于 Space 和 Replace;西门子其他程序的图表也是类似的。

突变过程可能会有一定的偏差，所以与西门子的程序相比，Space 上的突变人更难被杀死。然而，我们在《太空》中遵循了完全相同的突变过程，并没有明确的理由说明为什么结果会有所不同。因此，Space 比 Siemens 的程序大得多这一事实更有可能是突变检测率较低的原因，因为众所周知，较大的程序更难测试。然而，这一解释既不能证实，也不能证伪，因为我们拥有的主题程序数量很少，因为所有西门子程序都很小，大小相似，我们没有在一个连续的大小范围内观察到空间的大小。

Space 比西门子程序具有更高的故障可检测性，这是我们在上面已经讨论过的因素，也是最后一个需要考虑的因素。为了进一步研究这是否是最合理的解释，我们接下来看看故障和突变的可检测性，即它们被跨程序测试池随机抽取的测试用例检测到的可能性。

假设西门子手工种子故障比真正的空间故障更难检测——而且这不是由于测试用例跨测试池的检测能力的变化——我们应该观察以下趋势:(1)空间故障应该比任何单个测试用例更容易检测到西门子故障，(2)空间突变体不应该比任何单个测试用例更容易杀死西门子程序突变体。在接下来的两个部分中，我们将依次探索这些预测。

4.3 比较故障的可检测性在项目

图 3 显示了比率 E(F)的分布，检测每个故障 F 的容易程度，对于每个主题程序(第 3.4 节)。y 轴是每个故障的计算比率，而 x 轴是简单的主题程序。x 轴上每个主题程序的水平程度与其相对故障样本大小成比例。例如，我们可以看到，Space 和 Replace 的样本量较大，而 Schedule 和 Schedule2 的样本量相当小(表 1)。这些不相等的样本量需要在我们的分析中加以考虑。此外，图中显示了每个程序的观察结果、总体平均值(跨程序的横线)、每个程序特定平均值(跨每个菱形的横线)以及它们对平均值的 95%置信区间(每个菱形的垂直跨度)。

图 3 清楚地显示，正如我们所预期的那样，空间比率往往比其他程序更高。单向方差分析(ANOVA)[7,22]表明，总体而言，跨程序平均值存在统计学显著差异(f 检验，p 值< 0.0001)。Siemens 程序和 Space 故障的 E(F)平均值分别为 0.035 和 0.14，因此明显显示出显著性差异。

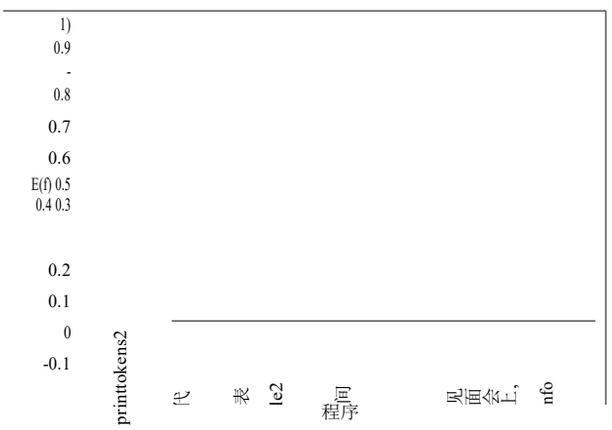

图 3。故障易检测度分布- E(F)

将每个 Siemens 程序与 Space 进行进一步的统计检验(使用 Bonferroni 程序 4，使用 t 检验[22]对平均值进行多重比较)，表明 Space 仅与 Tcas 和 Replace 显著不同(α = 0.05)，尽管 Space 与所有其他程序之间的差异在图形上是可见的[5]。西门子其他程序缺乏统计显著性的一个可能原因是，我们处理的是小样本，因为故障数量从 7 (printtoken) 到 23 (Totinfo) 不等。这也可能是由于 Bonferroni 程序往往是保守的[7]。这增加了 II 型错误的概率，并可能导致合法的显著结果无法被检测到。换句话说，它保证了第 I 类错误率最多为 α (在我们的例子中为 0.05)，但在现实中可能要小得多。

4.4 跨程序比较突变体的可检测性

图 4 显示了 E(M)的分布，即杀死每个突变体的测试用例的比率，对于每个主题程序(第 3.4 节)。它的结构与图 3 相同，只是它关注的是突变体而不是故障。

至于上一节中的错误，方差分析显示，不同程序之间的差异在统计上总体上是显著的(F-检验，p 值< 0.0001)。

当再次使用 Bonferroni 过程和 t 检验更仔细地观察每个程序对时，我们观察到，除了 Tcas (α = 0.05)外，Space 的平均值显著低于所有程序的平均值，这证实了图 4 中可见的结果。因此，我们可以得出这样的结论:不仅突变体不容易杀死，而且个别测试用例显示，Space 中检测突变体的难度更大，在较小程度上，Tcas 比 Siemens 程序中检测突变体的难度更大。对于 Space 来说，如上所述，这很可能是由于该程序的规模大得多(见表 1)。

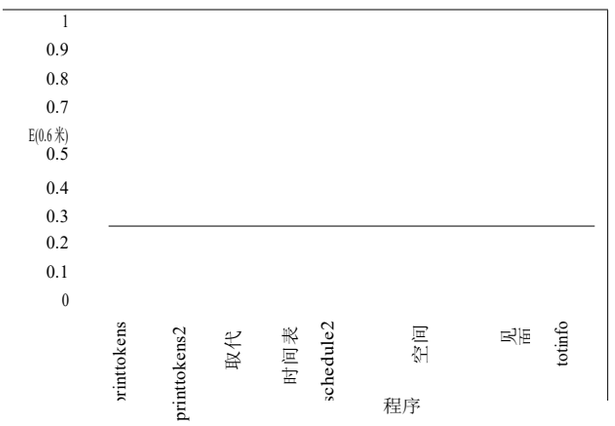

图 4。突变体易检测度分布- E(M)

[4] 这需要用 α 分割的比值作为显著性阈值
除以平均比较的次数，在我们的例子中是 28 次。

[5] 一个等价的非参数曼-惠特尼检验[7,22]得出了相同的结果。其他用于多个平均值比较的程序，如 Tukey 的方法，不能使用，因为我们在程序[22]之间有不等的样本大小。

这些突变体和错误的反向趋势支持我们的解释，即跨程序的差异不是由于不同的测试用例检测能力，因为这将相反地显示突变体和错误的一致趋势，即。，空间测试用例检测更多的突变和故障。因此，图 3 和图 4 以及它们相关的统计测试结果支持我们的猜想，即对于第 4.1 节中的趋势最合理的解释是，Space 中的故障往往更容易检测，因为它们是真实的故障，而不是像西门子程序那样由人类设计的故障。

4.5 讨论

基于上述结果，我们可以推断，测试套件突变检测率与 Space 中的故障检测率相当，而它们在其他程序中却有很大不同，这是因为在其他程序中播种的故障可能在检测的易用性方面不具有代表性。毕竟，它们并不是真正的故障，人类很可能很难衡量种子故障的可检测性。

第一次报道这些程序的论文[14]实际上包含了对这一结论的确证。Hutchins 等人明确指出，从开发人员建议的一组初始大故障中，被 350 个或更多测试用例(来自其原始测试池)检测到的故障被丢弃 [6]。这支持了我们上面的猜想，即《太空》的结果应该被赋予更多的可信性。如果是这样的话，那么我们可以得出这样的结论:基于这里提出的突变算子，突变体确实提供了代表真实故障的测试有效性结果。

它还意味着，任何关于人类种子故障的研究都应该仔细解释，因为我们看到，通过使用这里使用的示例程序——在以前的研究[8]中广泛使用——在评估测试技术的故障检出率时，人们会得到保守的结果。换句话说，这将导致低估测试技术的有效性。在衡量不同测试技术的成本效益时，这一点尤为重要。如果技术 A 比技术 B 检测到更大比例的难以检测的西门子故障，但 A 比 B 更昂贵，我们可能实际上没有证明 A 的相对成本效益。

在一些实验设置中，我们可能想把精力集中在难以检测的故障上，因为这些是程序员最头疼的故障。在这方面，西门子套件中的缺陷种子比我们在这里使用的全套突变更可取。然而，我们认为，研究人员应该可以选择困难的突变体，就像选择困难的西门子故障一样，即。，通过排除那些被大量测试用例杀死的突变体。如果研究人员做到了这一点，那么他们就有可能以一种允许其他研究人员复制的精确方式，描述整个突变选择过程，从突变算子到容易的突变排除。

[6] 他们报告说，大约 38%的错误因为这个原因而被丢弃，18%的错误因为补充性的原因被丢弃，即它们太难被检测到。

5.结论

这篇论文的主要贡献是对迄今为止许多实验软件测试研究的基本假设的彻底调查:手工(例如，由经验丰富的工程师)或从突变算子生成的故障是真实故障的代表。事实上，大多数实验结果依赖于种子突变体[1,2,15,16,24]-使用突变算子-或由经验丰富的开发人员选择的错误[10,14]。此外，这一问题不太可能在未来的研究中消失，因为真正的故障通常太少，不适合进行允许统计分析的实验。因此，即使一个特定的课题程序有真实的故障，人们也经常被迫播种额外的故障。

因此，一个基本的问题是，突变体或人工播种的故障是否有可能产生结果，例如，在检测能力方面，这些结果是人们在真实故障上所能获得的代表性结果。我们的分析表明，突变体在使用精心选择的突变算子并去除等效突变体后，可以很好地表明测试套件的故障检测能力。此外，它显示了使用人类选择的故障的危险，因为在本文研究的系统中，它导致低估测试套件的故障检测能力。人类选择的故障的另一个问题更多地与随着时间的推移建立一个关于测试技术效率的实验结果体的必要性有关。为了使这成为可能，研究人员必须重复进行研究。当错误的选择是基于主观的、不确定的过程时，这是很难实现的。

我们还应该注意到，虽然我们在这篇论文中的重点是表明，我们生成的突变体并不比真正的故障更容易检测，但我们的数据也表明，在任何实际程度上，它们并不比真正的故障更难检测。这一事实支持了作为突变测试理论基础的“称职的程序员假设”;也就是说，程序员犯的错误会被杀死变种人的测试套件检测到的假设。

未来的研究当然需要复制我们在本文中报道的研究。我们这样做的方式应该有助于我们的分析的精确复制，从而使未来的结果与我们的比较更容易。同样重要的是，在面向对象系统的背景下，用合适的突变算子进行类似的研究，因为这些系统代表着工业系统中越来越大的份额。最后，由于本文中的结果报告了随机选择的相同大小的测试套件(尽管构建它们的测试池满足结构覆盖标准)，研究根据各种标准(如代码覆盖标准或操作概要标准)选择的测试套件的结果是否相同也将是重要的。

6.致谢

非常感谢 Gregg Rothermel 和 Hyunsook Do 的宝贵讨论和建议，并向我们发送西门子和空间项目以及所有相关的文物。也感谢所有研究人员，这些年来，他们在这些课题项目和人工制品上进行了工作和改进。我们也要感谢 Mike Sowka 审阅了这篇论文的草稿。这项工作得到了加拿大研究主席(CRC)拨款的部分支持。杰米

Andrews、Lionel Briand 和 Yvan Labiche 进一步得到了 NSERC 运作拨款的支持。

7.参考文献

[1] J. H. Andrews 和 Y. Zhang，“用日志文件分析检查一般测试结果”，IEEE Transactions on Software Engineering, vol. 29 (7)， pp. 634-648, 2003。

[2] L. Briand, Y. Labiche 和 Y. Wang，“使用模拟实证调查测试覆盖标准”，Proc. IEEE/ACM 软件工程国际会议，爱丁堡，86-95 页，2004 年 5 月。

[3] t.a Budd 和 D. Angluin，“正确性的两种概念及其与测试的关系”，《信息学报》，第 18(1)卷，第 31-45 页，1982 年。

[4] D. T. Campbell 和 J. C. Stanley，研究的实验和准实验设计，霍顿米夫林公司，1990 年。

[5] W. Chen, R. H. Untch, G. Rothermel, S. Elbaum 和 J. von Ronne，“故障暴露-潜在估计能提高测试套件的故障检测能力吗?，“软件测试，验证和可靠性，vol. 12 (4)， pp. 197-218, 2002。

[6] R. A. DeMillo, R. J. Lipton 和 F. G. Sayward，“测试数据选择的提示:对实践程序员的帮助”，《IEEE 计算机》，vol. 11 (4)， pp. 34-41, 1978。

[7] J. L. Devore，《工程与科学的概率与统计》，达克斯伯里出版社，5 页 th 版,1999 年版。

[8] H. Do、G. Rothermel 和 S. Elbaum，“用软件测试和回归测试技术对控制实验的基础设施支持”，俄勒冈州立大学科瓦利斯，OR，美国，04-06-01 技术报告，2004 年 1 月。

[9] P. G. Frankl 和 O. Iakounenko，《测试有效性的进一步实证研究》，Proc. 6th *ACM SIGSOFT 软件工程基础国际研讨会，Orlando (FL, USA)， 153-162 页，1998 年 11 月 1-5 日。*

[10] P. G. Frankl 和 S. N. Weiss，“全用途和全边充分性标准的有效性的实验比较”，Proc. 4th *《测试、分析和验证研讨会》，纽约，第 154-164 页，1991 年。*

[11] T. L. Graves, M. J. Harrold, J.-M.;Kim, A. Porter 和 G. Rothermel，“回归测试选择技术的实证研究”，ACM 软件工程和方法论学报，第 10 卷(2)，第 184-208,2001。

[12] R. G. Hamlet，“在编译器的帮助下测试程序”，《IEEE 软件工程汇刊》，vol. 3 (4)， pp. 279- 290,1977。

[13] M. Harder, J. Mellen 和 M. D. Ernst，“通过操作抽象改进测试套件”，Proc.第 25 届软件工程国际会议，波特兰，OR，美国，第 60-71 页，2003 年 5 月。

[14] M. Hutchins, H. Froster, T. Goradia 和 T. Ostrand，“基于数据流和控制流的测试充分性标准有效性的实验”，Proc. 第 16 届 IEEE 软件工程国际会议，索伦托(意大利)，第 91-200 页，1994 年 5 月 16-21 日。

[15] S. Kim, J. A. Clark 和 J. A. McDermid，“用突变方法调查面向对象测试策略的有效性”，《软件测试，验证与可靠性》，vol. 11 (3)， pp. 207- 225,2001。

[16] a.m. M. Memon, I. Banerjee 和 A. Nagarajan，“为了有效的 GUI 测试，我应该使用什么样的测试 Oracle ?”， Proc. IEEE 自动化软件工程国际会议(ASE’03)，加拿大魁北克蒙特利尔，第 164- 173 页，2003 年 10 月。

[17] A. J. Offutt，“软件测试耦合效应的调查”，ACM Transactions on Software Engineering and Methodology, vol. 1 (1)， pp. 3-18, 1992。

[18] A. J. Offutt, A. Lee, G. Rothermel, R. H. Untch 和 C. Zapf，“充分突变算子的实验确定”，ACM 软件工程与方法论学报，vol. 5 (2)， pp. 99- 118.1996。

[19] A. J. Offutt 和 J. Pan，“检测等效突变体和可行路径问题”，《软件测试、验证和可靠性》，vol. 7 (3)， pp. 165-192, 1997。

[20] A. J. Offutt 和 R. H. Untch，“突变 2000:联合正交”，Proc. Mutation, San Jose, CA, USA, 45- 55 页，2000 年 10 月。

[21] T. J. Ostrand 和 M. J. Balcer，“指定和生成功能测试的类别划分方法”，《ACM 通信》，vol. 31 (6)， pp. 676-686, 1988 年。

[22] J. Rice，《数理统计与数据分析》，达克斯伯里出版社，第 2 期 nd 版,1995 年版。

罗泽明和哈罗德，“一种安全回归测试选择技术的实证研究”，IEEE 译。*《软件工程》，第 24 卷(6)，第 401-419 页，1998*。

[24] P. Thévenod-Fosse, H. Waeselynck 和 Y. Crouzet，“关于软件结构测试的实验研究:确定性与随机输入生成”，Proc. 21 国际容错计算研讨会，加拿大蒙特利尔，1991 年 6 月，第 410-417 页。

[25] F. I. Vokolos 和 P. G. Frankl，“文本差异回归测试技术的实证评估”，Proc. IEEE 软件维护国际会议，Bethesda, MD, USA, 1998 年 3 月，第 44-53 页。

[26] C. Wohlin, P. Runeson, M. Host, M. C. Ohlsson, B. Regnell 和 A. Wesslen，《软件工程实验-导论》，Kluwer, 2000 年。

8.附录

图 5 和图 6 显示了 Am 和 Af 的平均值如何随着测试套件大小的变化而变化。我们可以观察到，不管大小如何，对于 Space 来说，Af 和 Am 是非常相似的。然而，对于 Replace 来说，对于剩下的程序来说，差异是实质性的，并且随着测试套件大小的减少，这种差异往往会增加。

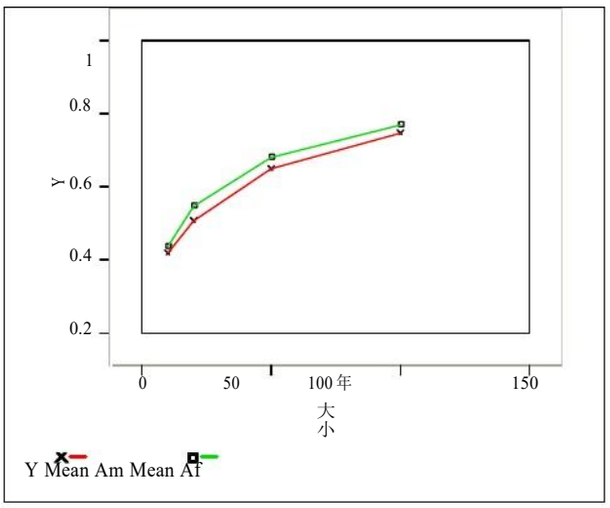

图 5。平均值(Am)和平均值(Af)对比测试套件大小-空间

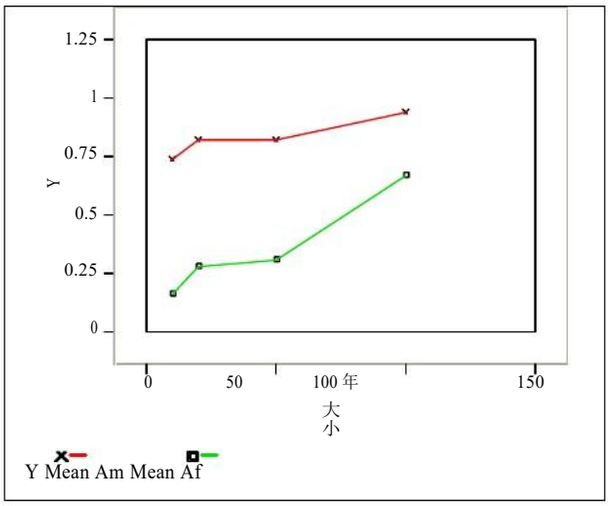

图 6。平均值(Am)和平均值(Af)对比测试套件大小-