**1背景介绍**

随机测试[1]以及分区测试[2,3,4]是两个非常著名的测试策略。在传统的随机测试中，按照统一或者不统一的概率分布随机地从软件输入域中选择测试用例。分区测试涉及到一簇的测试技术：状态测试、数据流测试、分枝测试、变异测试等，任何的一个输入域的子域，都需要从中挑选至少一个测试用例。

Cai[11,12]等人将随机测试与分区测试结合，提出了随机分区测试策略。该策略假设待测软件的输入域被分为个子分区。随机分区测试策略首先根据测试剖面选择一个分区。然后在中随机地选择一个测试用例执行。在整个的测试过程中测试剖面的大小不变。

在随机分区测试策略中，一个分区对应的选择概率在整个的测试过程中是不变的，这一点可能不总是好的。因为引起故障的输入在输入域中趋向于聚簇在连续的区域[5-7]，也就是说存在一些分区更可能揭示软件中的故障。Cai 等人依据这一想法，利用软件的控制理论[9]提出了动态随机测试策略（DRT）[8]以改进传统的随机测试与随机分区测试。软件的控制理论探索软件工程理论与控制理论相互作用的关系，被用来解决软件工程中的问题。DRT策略的主要特点是在测试的过程中根据每一次测试用例的执行结果动态改变测试剖面：假设存在一个分区，若该分区中的一个测试用例揭示了软件中的故障，那么认为该分区具有更大的可能性再次检测出软件中的故障，因此增大该分区被选择的概率，即，。如果这个测试用例没有检测出故障，减小该分区被选择的概率，即，。

但是该方法仍然存在一些不足。

1. 由于在测试过程中参数的取值很小，具有更高检测能力的分区很难在短时间内突显出来。特别是软件输入域的失效率很小的情况下。本文提出一种基于奖励惩罚机制的动态随机测试策略提高DRT的测试效率。
2. 某个分区被选择的概率受其它分区测试结果的影响,使得该分区的概率无法准确地反映该分区真正的故障检测能力。将软件的输入域按照一定的方式划分为若干分区之后，每个分区的故障检测能力是独立的。但是在传统的DRT算法中每个分区的故障检测能力，随着其它分区的测试用例检测结果发生改变。本文利用Markov链的状态转移矩阵，提出一种DRT的改进策略MDRT,该策略将测试过程中每一次选择分区当成状态转移的过程，每一个状态根据自身情况调整，不受其它状态的改变而发生改变。
3. 直觉上，每次调整分区的概率幅度应当根据当前分区的概率大小，而不应该是一致的：如果当前分区的概率比较大说明该分区在理论上具有更高的故障检测能力，但是并不能保证该分区的每一个测试用例都能揭示软件的故障，因此该分区的测试用例没有揭示故障时的调整幅度应当比较小分区没有检测出故障时的调整幅度要小。本文在DRT策略的算法上进行优化提出了更合理的MDRT测试策略。
4. 在黑盒测试中，同一个项目的不同分区策略以及同一个项目的相同分区策略，分区的数目也可能不同，不同数目的分区导致算法的检测效率发生改变。本文通过五个实验研究相同分区策略下不同分区数目对实验结果的影响。
5. 一般情况下，初始测试剖面中，即初始条件下每一个分区被选择的概率是均等的。这种做法在某些情况下可能不是最好的选择。由于产生故障的输入在输入域中趋向于聚簇在连续的区域，当软件输入域的失效率较低时，意味着输入域中有很少的测试用例能够揭示软件中的故障并且测试用例数目多的分区更有可能包含能够揭示故障的测试用例。因此本文提出将分区内的测试用例数目占输入域中所有测试用例数目的百分比作为初始剖面的概率分布。然后通过实验对比初始剖面为均等概率分布和不均等概率分布不同策略的测试效率。

接下里的paper...

**2相关工作**

很多的工作对随机测试与分区测试做了研究。Myers[13]认为：“随机测试是所有测试策略中效率最低的”。然而Duran[14]认为：“随机测试在很多程序中表现良好并且有时候可以用较小的代价揭示相对难发现的缺陷”。Weyuker[15]经过研究之后发现：分区测试可能是一个卓越的测试策略也可能是一个低效率的测试策略，分区测试的效率很大程度上取决于如何将产生错误输出的输入集中在某个或者某些分区中。Hamlet[3]认为成功的分区测试不能激发测试人员对软件质量的信心。Chen[4]认为当存在较高失效率的分区时，分区测试具有较高的检测能力。[Gutjahr](http://xueshu.baidu.com/s?wd=author%3A%28Gutjahr%2C%20W.J%29%20Dept.%20of%20Stat.%20Oper.%20Res.%20%26%20Comput.%20Sci.%2C%20Wien%20Univ.%2C%20Austria&tn=SE_baiduxueshu_c1gjeupa&ie=utf-8&sc_f_para=sc_hilight%3Dperson) [17]认为在不确定条件下，分区测试比随机测试效率更高。

Cai结合了随机测试和分区测试的特点提出了分区随机测试（RPT）策略。RPT策略首先根据测试剖面选择分区，然后在中随机选择测试用例。Cai[19]利用软件控制理论提出了适应性测试策略（AT），该策略的测试效率相对于随机测试、分区测试有很高的改进[11,19]。但是AT策略在实际中需要消耗大量的时间。为了解决这一问题Cai[8]提出了动态随机测试策略（DRT）。在DRT策略中，测试剖面根据测试的反馈信息动态地改变。

**3基于Markov链的动态随机测试和基于奖惩机制的动态随机测试**

这个章节，先介绍动态随机测试(DRT)策略，然后介绍Markov基本理论，最后介绍基于Markov链的动态随机测试(MDRT)策略。

1. *DRT策略*

Cai最先在[8]中提出了动态随机测试，这里将完整的算法策略展示如下。

1. 将待测软件的输入域划分为个不相交的分区：，每一个分区中有个测试用例。
2. 初始化参数，并且。
3. 根据每一个分区所对应的概率随机选取一个分区,在这里。
4. 随机地从中挑选一个测试用例。

如果测试用例揭示了软件中的故障，就增大测试用例所在分区被选择的概率，同时减小其它分区被选择的概率，并把缺陷移除。



如果测试用例没有找到缺陷，就减少被选中的概率，同时增大其它分区被选中的概率。



1. 检查停止条件，如果不满足，则跳转执行步骤3，如果满足则停止测试。

根据上面的DRT策略的描述，可以看出划分分区的个数，初始剖面的大小、这两个参数的取值，这些因素都影响DRT策略的测试效率。其中分区的个数以及初始剖面的取值是属于外部因素，参数的取值是内部因素。探究这些因素对算法测试效率的影响大都集中对参数的研究，很少的工作对分区的个数以及初始剖面的大小进行研究。Lv在[10]中假设软件输入域的失效率已知、各个分区的失效率已知、测试过程中各个分区失效率保持不变以及测试用例执行之后放回原来的分区之中，通过理论分析的方式得到了的最佳取值范围。然而实际中很难知道输入域的失效率以及各个分区的失效率大小。Yang在[22]中通过在实验的过程中统计每一个分区的成功检测率，然后调整的取值，在软件的输入域失效率比较大的情况下，效果比较好。在实际的项目中简单有效地确定参数的取值仍然是一个困难的事情，特别在软件输入域的失效率较小的情况下，即软件中的缺陷不容易被发现。本文对5个真实的软件项目通过设置，其中

。在试验中发现当处于区间时，DRT算法具有较高的检测效率。初始剖面的取值有两种可能，一种是的均等初始剖面，另一种是不均等的初始剖面。在以往的研究中[10][20]，均采用均等的初始剖面。确定不均等的测试剖面的取值是一件较为困难的事情，或者是不知道依据什么确定初始剖面的取值。直觉上，当软件输入域的失效率比较小时，划分分区之后，分区内的测试用例越多，该分区越有可能检测出软件中的缺陷。由此，本文根据分区内测试用例的数目占软件输入域的测试用例总数的百分比作为初始剖面取值的一种方式。另一种方式采用均等的初始剖面，并将这两种不同的初始剖面取值对测试策略的效率进行对比。当软件输入域的失效率较低时，测试策略在不均等的初始剖面下比均等的初始剖面具有更高的检测效率。将软件的输入域划分分区的方式有很多，例如边界值分析、功能分区、等价类、决策表、CPM[]等。一方面，同一个测试项目的不同分区策略得到的分区数目不一定相同并且很难确定某一个分区策略对于一个特定的测试项目是最佳的选择；另一方面，同一个分区策略，分区的数目也不一定相同。例如：。。。。。。。

1. *Markov链理论概述*

Markov随机过程称为Markov链，具备“无后效应”，即要确定过程将来的状态，知道它此刻的情况就够了，并不需要对它以往状况的认识。对于有限个或可列个值，以来标记并称它们为过程的状态，对于任意的及状态有：。可见，一旦Markov链的初始分布给定，其统计特性完全由条件概率决定。

假设状态空间。

定义1（转移概率）：条件概率为Markov链的一步转移概率，简称转移概率。

定义2（时齐Markov链）：当Markov链的转移概率只与状态有关，而与无关时，称Markov链为时齐的，并记为。

由定义2知道我们可以将排成一个矩阵的形式，令



则称为转移矩阵。

转移矩阵P具有如下性质：



1. *MDRT策略*

MDRT算法结合了传统随机算法与分区算法的特点，并引入软件的控制理论。假设软件的输入域有个测试用例，被划分到个不相交的分区中并且每一个分区有个测试用例，。如果将每个时间点测试用例所在的分区作为时刻t测试系统所处的状态，则整个状态空间为。根据在状态下测试用例的执行结果可以计算调整到状态的转移概率。如果状态下检测出软件中存在缺陷，那么就增大下一时刻转移到的概率，同时减小转移到其它状态的概率；反之，如果在状态下没有检测出软件中存在缺陷那么就减少下一步转移到状态的概率，同时增大转移到其它状态的概率。另外，根据当前状态下测试用例的执行结果每次增大或者减少的转移概率的幅度应该是不同的。一般情况下，一方面即便某一个分区具有较强的检测出软件存在缺陷的能力，也不可能每一次在该分区中选择测试用例都会检测出缺陷。另一方面如果某一个分区被选择的概率比较大说明在以往的测试过程中较多的检测出了软件中的缺陷或者理论上有较大的可能揭示软件中的缺陷。在检测过程中增大或者减少该分区被选择的概率幅度时应当与该分区当前概率有关：如果当前分区被选择的概率较大，那么增大该分区被选择的概率幅度就越大，减小该分区被选择的幅度就越小。整个过程的状态可以用转移矩阵表示。在整个软件测试过程中，将每一个时间点选取测试用例并执行作为一次决策行动，则行动全体组成整个行动空间，并且每个时间点的状态和所采取的行动都会影响到下一个时间点的状态。因此，整个测试过程形成一个Markov决策过程。

开始时刻初始化测试剖面，则初始转移矩阵其中。

整个软件测试的执行可以分为如下步骤：

步骤1：根据当前分区到其它分区所对应的转移概率随机选取一个分区(第一次根据测试剖面选择分区)，在这里。转步骤2。

步骤2：等概率随机地从分区中选取一个测试用例TC,转步骤3。

步骤3：执行选中的测试用例TC：

如果测试用例TC找出了缺陷，就增大测试用例TC所在状态转移到本身的概率，同时减小转移到其它状态的概率，并把缺陷移除：

如果测试用例TC没有找到缺陷，就减少测试用例TC所在状态转移到本身的概率，同时增大转移到其它状态的概率：

更新转移矩阵：

步骤4：检查测试停止条件，如果不满足则转步骤1；如果满足则停止测试。

1. *RDRT策略*

在软件中不存在难检测的故障时，Yang在[22]中提出的A-DRT策略的测试效率比传统的DRT策略有明显的提高；但是软件存在难检测的故障时，效果不理想。当软件的失效率高时，软件内的缺陷很多测试策略都能用较小的代价揭示出来。但是当软件的失效率低时，不同检测策略的效率差别很大。在以往的测试活动中发现，当失效率很低时DRT策略的测试效率相对于RT策略没有提高或者提高不明显。直觉地，引起故障的输入在输入域中趋向于聚簇在连续的区域，即存在一个或者少数分区具有较高的检测能力。因此在软件输入域的失效率较低时，往往一些分区内不具备揭示软件中缺陷的能力或者具备较小的检测能力。另一方面由于每次调整概率的幅度很小，并且某一个分区被选择的概率易受到其它分区的测试结果的影响，使得那些不具备或者具备很小的检测能力的分区仍然被不断的选择，最终具有较高检测能力的分区不容易突显出来.因此DRT策略在软件输入域的失效率低时，测试效率不高。为了缓解这一问题，本文提出了基于奖惩机制的动态随机测试策略(RDRT)，该策略旨在加速测试的过程：如果分区内的测试用例揭示了软件中的缺陷，下一次仍在该分区内选择测试用例，那么该分区的其它测试用例具有较大的可能揭示另一个软件中的缺陷，因此下一次仍在该分区选择测试用例，并且该分区绑定的奖励因子自增一次，对应的惩罚因子清0，直到该分区中的测试用例没有检测出软件中的缺陷，奖励因子清0，惩罚因子+1。奖励因子越大该分区对应的概率增加的越多。相反地，如果存在这样一个分区：累计n次选中该分区，但是该分区中的测试用例均没有揭示出软件中存在缺陷，那么就认为该分区具有较低的检测能力，甚至不具备检测能力，让该分区对应的选择概率为0。

假定软件测试的输入域中的测试用例划分到m个不相交的分区中，输入域中共有个测试用例，用来表示这m个分区，每一个分区有 个测试用例。初始每个分区的奖励因子，惩罚因子，惩罚上限。

步骤1：根据当前各个分区所对应的概率选取分区,其中。

步骤2：等概率随机地从分区中选取一个测试用例TC。

步骤3：执行选中的测试用例TC。如果测试用例TC揭示了软件中的故障转步骤4，反之转步骤5。

步骤4：分区的奖励因子，惩罚因子，并移除缺陷。转步骤2。

步骤5：分区的惩罚因子。如果就增大测试用例TC所在分区对应的概率，同时减小其它分区被选择的概率：

如果，就减少该分区对应的概率，同时增大其它分区对应的概率，如果该分区的惩罚因子则该分区对应的概率：

步骤6：检查测试停止条件，如果不满足则转步骤1；如果满足则停止测试。

1. *Remarks*
2. 为了方便研究本实验中RDRT策略以及DRT策略的参数。为了研究是否存在一个区间D，参数在区间D时，对于不同的实验对象DRT，RDRT仍有较高的故障检测效率。为此本实验设置了一系列的参数取值：

。对于MDRT策略中的参数也设置了一系列的取值:

。

当MDRT策略的参数为1时，如果一个分区中的测试用例没有揭示软件中的故障，那么该分区对应的概率就会降为0，这样明显不恰当，因此将参数的最大值设为0.9。

1. RDRT策略的惩罚上限根据具体实验对象的不同设置的具体值也可能不同。在grep实验中，由于测试用例的数目比较多并且本实验中的故障较难检测因此惩罚上限设置为50。基于相似地考虑gzip实验的惩罚上限为30；flex实验的惩罚上限为10；bash实验的惩罚上限为50；make实验的惩罚上限为10。
2. 初始的测试剖面应该有测试工程师根据以往的测试经验设定。本文采取两种方式设定初始测试剖面。第一种方式为；第二种设置方式为，表示分区内的测试用例数目，表示SUT输入域的测试用例总数。这两种分区方式应用于RPT、DRT、MDRT、RDRT四种测试策略中。
3. 在MDRT策略中通过将当前测试用例所在的分区当成此时所处的状态，根据该测试用例的执行结果计算调整到下一个状态的转移概率。通过转移矩阵使得某一个分区的故障检测能力仅由该分区内的测试用力执行情况决定，不受其它分区测试结果的影响，从而加速了找到具有高故障检测能力分区的进程。通过步骤3使得每一次根据某一分区内的测试用例执行情况调整分区概率的幅度由当前分区被选择的概率大小决定：如果当前分区被选择的概率大，增加的幅度就比较大，减少的幅度就比较小；如果当前分区被选择的概率小，增加的幅度相对较小，增加的幅度相对较大。

**4实验设置**

1. *实验对象*

为了避免测试策略在特定的程序中具有更高的故障检测能力，本文在Software artifact

Infrastructure Repository(SIR)网站下载了五个真实的程序，每一个程序附带测试用例集以及故障。为了模拟实际的软件测试，本文选取的程序代码行数均大于5K。

在SIR网站中，每一个程序都有不同的版本以及对应的测试用例集和故障。例如bash程

序有6个可测的版本，本文将V1作为SUT。实验对象的基础信息展示在表1。

表1. 实验对象

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **源程序** | **代码行数** | **测试版本** | **测试用例数目** | **故障数目** | **测试的故障数目** | **分区数目** |
| **bash** | 59846 | V1 | 1061 | 6 | 6 | 6，4 |
| **flex** | 10459 | V1 | 567 | 19 | 3 | 6，3 |
| **grep** | 10068 | V1 | 809 | 19 | 5 | 6，4 |
| **gzip** | 5680 | V1 | 217 | 16 | 5 | 4，3 |
| **make** | 35545 | V1 | 793 | 19 | 2 | 5，3 |

1. *测试用例和分区*

TSL[24]是一个用来书写测试规格说明书的语言。基于测试规格说明书中的信息产生大量

的测试帧。为每一个测试帧中的choice指定一个具体的值，可以得到一个具体的测试用例。根据等价类划分策略将规格说明书中规定的相同处理方式组成一个分区，按照粗细两种粒度得到不同的分区数目。根据每一个测试用例的具体情况，将测试用例划分到不同的分区之中

。具体的划分情况展示在表1的最后一列。

1. *测试的故障选择*

对于每一个实验对象，本实验先用随机测试对每一个故障进行测试，重复50遍，得到

每一个故障被检测到时用的测试用例数目。比较每一个故障用到的测试用例数目然后取相对难杀死的故障(检测到该故障用到的测试用例数目较多)，每一个实验对象测试的故障数目展示在表1的倒数第二列。

1. *测试策略*

本实验检测了五个策略：传统的随机测试策略(RT)，随机分区测试策略(RPT)，动态随机

测试策略(DRT)，以及本文提到的基于Markov链的动态随机测试策略和基于奖惩机制的动态随机测试策略。在测试过程中，如果一个故障被检测到就立即移除该故障。测试的停止条件为所有的故障都被揭示和移除。本实验对于DRT,MDRT,RDRT的不同参数、不同的初始测试剖面和不同的分区数目重复20次。DRT,RDRT策略的参数取值情况在Ⅲ.B中，MDRT的参数取值情况在Ⅲ.C中。

1. *指定随机数种子*

计算机产生的随机数是伪随机,如果不指定随机数种子，它将以当前时间为种子随机产生

随机数。如果这样做一方面导致实验不可重复，另一方面也使得不同测试策略的差异是随机数产生的还是策略本身产生的无法确定。因此对于同一个实验对象的不同重复次数指定的随机数种子不同，同一重复次数的不同测试策略的随机数种子相同。

1. *度量方法*

为了反映五个测试策略的故障检测效率，在实验中运用了4个度量标准：

1. ，揭示第一个故障需要的测试用例数目。的均值用表示
2. ，揭示第一个故障之后到揭示第二个故障需要的测试用例数目。的均值用表示。
3. ，揭示软件中所有的故障需要的测试用例数目。的均值用表示
4. D，为以上三个度量标准差。

接下来的章节展示了不同实验的各个度量标准的值。本文没有考虑执行每一个测试用例花费的代价，假设同一个实验中，每一个测试用例花费的代价是相同的。在这种情况下，F、NF、T三种度量标准可以用来比较不同测试策略检测故障的效率。

1. *实验结果*

表2(a)，(b)，(c)分别展示了在F、NF、T度量标准下，RT策略的测试结果以及RPT、

DRT、MDRT、RDRT策略的相对于RT策略的提升率。

表2 实验结果 (a)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 实验项目 | 分区数目及初始概率分布 | RT | RPT | DRT | MDRT | RDRT |
| bash | 4个分区均等初始概率分布 | 36.15 | -24.48% | -96.13% | -82.85% | -16.60% |
| 4个分区不均等初始概率分布 | 36.15 | -45.37% | -20.19% | -18.12% | -32.50% |
| 6个分区均等初始概率分布 | 36.15 | 26.28% | 30.43% | 35.41% | 36.38% |
| 6个分区不均等初始概率分布 | 36.15 | -29.60% | 7.19% | -62.10% | -56.02% |
| flex | 3个分区均等初始概率分布 | 3.75 | -110.67% | -109.33% | -104.00% | -77.33% |
| 3个分区不均等初始概率分布 | 3.75 | -13.33% | -69.33% | -9.33% | -10.67% |
| 6个分区均等初始概率分布 | 3.75 | -37.33% | -12.00% | -33.33% | -41.33% |
| 6个分区不均等初始概率分布 | 3.75 | -6.67% | 4.00% | 4.00% | -10.67% |
| grep | 4个分区均等初始概率分布 | 97.95 | 12.05% | 14.50% | 32.01% | 36.50% |
| 4个分区不均等初始概率分布 | 97.95 | -35.78% | 22.26% | 27.41% | 20.62% |
| 6个分区均等初始概率分布 | 97.95 | 41.21% | 34.05% | 38.74% | 53.34% |
| 6个分区不均等初始概率分布 | 97.95 | 48.65% | 23.99% | 18.12% | 36.24% |
| gzip | 3个分区均等初始概率分布 | 38.65 | -8.67% | -6.47% | -4.79% | -1.42% |
| 3个分区不均等初始概率分布 | 38.65 | -8.15% | 8.54% | -3.36% | 34.54% |
| 4个分区均等初始概率分布 | 38.65 | 57.31% | 60.80% | 65.46% | 72.32% |
| 4个分区不均等初始概率分布 | 38.65 | 8.15% | 41.53% | 23.54% | 50.84% |
| make | 3个分区均等初始概率分布 | 90.80 | 89.92% | 90.31% | 90.86% | 90.86% |
| 3个分区不均等初始概率分布 | 90.80 | -21.04% | 1.76% | 36.56% | 75.83% |
| 5个分区均等初始概率分布 | 90.80 | 81.50% | 81.94% | 85.68% | 85.19% |
| 5个分区不均等初始概率分布 | 90.80 | -22.41% | -11.56% | 41.30% | 65.25% |

表2 实验结果 (b)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 实验项目 | 分区数目及初始概率分布 | RT | RPT | DRT | MDRT | RDRT |
| bash | 4个分区均等初始概率分布 | 60.35 | -93.29% | 0.66% | -39.93% | -44.16% |
| 4个分区不均等初始概率分布 | 60.35 | -33.89% | -14.17% | -45.40% | 5.72% |
| 6个分区均等初始概率分布 | 60.35 | -19.97% | -28.83% | 17.56% | 1.99% |
| 6个分区不均等初始概率分布 | 60.35 | -44.74% | -37.20% | -5.39% | 54.85% |
| flex | 3个分区均等初始概率分布 | 4.00 | -111.25% | -103.75% | -93.75% | -65.00% |
| 3个分区不均等初始概率分布 | 4.00 | -16.25% | -83.75% | -2.50% | -75.00% |
| 6个分区均等初始概率分布 | 4.00 | -120.00% | -45.00% | -43.75% | -33.75% |
| 6个分区不均等初始概率分布 | 4.00 | -25.00% | -20.00% | -1.25% | -17.50% |
| grep | 4个分区均等初始概率分布 | 132.50 | 17.47% | 28.04% | 32.75% | 42.72% |
| 4个分区不均等初始概率分布 | 132.50 | -10.45% | -5.92% | -47.25% | 40.04% |
| 6个分区均等初始概率分布 | 132.50 | 31.13% | 46.08% | 47.81% | 53.09% |
| 6个分区不均等初始概率分布 | 132.50 | 44.60% | -38.38% | 11.02% | 33.74% |
| gzip | 3个分区均等初始概率分布 | 86.15 | -73.24% | -130.59% | -108.13% | -34.30% |
| 3个分区不均等初始概率分布 | 86.15 | 24.26% | -9.00% | 2.79% | 5.05% |
| 4个分区均等初始概率分布 | 86.15 | 69.41% | 64.94% | 78.87% | 73.88% |
| 4个分区不均等初始概率分布 | 86.15 | 7.72% | 71.62% | 69.53% | 72.72% |

表2 实验结果 (c)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 实验项目 | 分区数目及初始概率分布 | RT | RPT | DRT | MDRT | RDRT |
| bash | 4个分区均等初始概率分布 | 1582.95 | -19.76% | -9.56% | 2.35% | 10.67% |
| 4个分区不均等初始概率分布 | 1582.95 | 0.17% | 2.07% | 5.04% | 8.27% |
| 6个分区均等初始概率分布 | 1582.95 | 3.96% | 8.39% | 10.81% | 12.06% |
| 6个分区不均等初始概率分布 | 1582.95 | 23.44% | 21.87% | 26.79% | 31.64% |
| flex | 3个分区均等初始概率分布 | 26.65 | -3.38% | -18.57% | -6.00% | -16.51% |
| 3个分区不均等初始概率分布 | 26.65 | -0.38% | -20.08% | 18.20% | 13.13% |
| 6个分区均等初始概率分布 | 26.65 | 12.95% | -12.76% | 1.31% | 24.77% |
| 6个分区不均等初始概率分布 | 26.65 | -14.45% | -23.83% | 21.76% | 15.95% |
| grep | 4个分区均等初始概率分布 | 1486.95 | -67.03% | -28.20% | -12.82% | -3.28% |
| 4个分区不均等初始概率分布 | 1486.95 | 11.43% | -24.37% | -8.81% | -24.32% |
| 6个分区均等初始概率分布 | 1486.95 | -58.00% | -57.36% | -29.11% | -52.43% |
| 6个分区不均等初始概率分布 | 1486.95 | -48.21% | -71.04% | -35.34% | -60.66% |
| gzip | 3个分区均等初始概率分布 | 332.15 | -66.88% | -66.28% | -39.88% | -65.24% |
| 3个分区不均等初始概率分布 | 332.15 | -0.41% | -25.55% | -20.40% | -21.80% |
| 4个分区均等初始概率分布 | 332.15 | 65.87% | 65.08% | 73.34% | 78.19% |
| 4个分区不均等初始概率分布 | 332.15 | 8.35% | 69.02% | 73.16% | 73.91% |
| make | 3个分区均等初始概率分布 | 278.70 | 91.51% | 91.84% | 92.21% | 93.04% |
| 3个分区不均等初始概率分布 | 278.70 | -8.72% | 49.10% | 50.79% | 88.12% |
| 5个分区均等初始概率分布 | 278.70 | 85.77% | 86.51% | 88.30% | 88.57% |
| 5个分区不均等初始概率分布 | 278.70 | 5.87% | 36.58% | 39.68% | 83.42% |

5数据分析和讨论

1. *数据分析*
2. 除了make实验，DRT、MDRT、RDRT在其它测试对象上几乎所有的测试结果显示数目较多的分区方式比数目较少的分区方式具有更高的故障检测效率。由于make实验能够检测出软件故障的测试用例集中在一个分区之中，因此分区数目少时有更高的概率选中该分区。
3. 通过比较不同的测试策略相对于RT策略的提升率，几乎所有的测试结果表明：DRT、MDRT、RDRT策略在均等的初始概率分布下，具有更高的故障检测效率。make实验中能够检测出故障的测试用例所在的分区具有较少的测试用例，即该分区在不均等的初始概率分布下被选中的概率很小，因此DRT、MDRT、RDRT策略在均等的初始概率分布下的测试效率比不均等的初始概率分布的测试效率高的多。从make实验的T度量标准分析，RDRT策略在不均等的初始概率分布下，比其它策略的测试效率都高。这一现象说明当初始剖面的设置与分区的故障检测能力相差较大时，RDRT策略能够更加迅速地找到故障检测能力强的分区。
4. 对于除了flex实验的每一个实验在均等的初始概率分布以及分区数目较多的情况下，几乎所有的测试结果表明：MDRT、RDRT策略揭示第一个和第二个故障的测试效率比RT、RPT、DRT策略的测试效率高。在flex实验中，DRT策略的测试效率在F、NF、T三个度量指标下均不如RT策略，MDRT策略以及RDRT策略在F、NF度量指标下不如RT策略，但是T度量相对RT策略表现更好。可能有一下几个原因：1，正如第一部分提到的引起故障的输入趋向于集簇在连续的区域。DRT、MDRT、RDRT策略都是根据这一个思想改进RT测试策略。然而在对flex对象测试时，能够揭示软件故障的测试用例数目很多，导致引起故障的输入分散到整体输入域中。这就意味着每一个分区的故障检测能力几乎相同。此时根据历史信息更新测试剖面很有可能不能提高软件的测试效率。2，根据表2(a)，(b)中的数据显示揭示软件中的前两个故障RT策略平均需要7.75个测试用例。这表明flex程序中的故障“很容易”被揭示。在对这样很容易揭示软件中故障的程序DRT、MDRT、RDRT策略没有更多的机会调整测试剖面，测试过程就已经结束了。MDRT策略不受其它分区测试结果的影响，并且根据所处分区当前被选择概率大小调整概率的幅度，使得该策略可以更快速的识别每一个分区的故障检测能力，因此MDRT策略在揭示flex程序所有的故障时比DRT、RPT、RT策略具有更高的测试效率。RDRT策略由于具有奖惩机制，因此在对这种“很容易”被检测出故障的程序在经过短暂的测试剖面调整之后更快的辨别每一个分区的故障检测能力，从而在T度量指标下比DRT、RPT、RT具有更高的故障检测效率。
5. 对于除了grep实验的每一个实验在均等的初始概率分布以及分区数目较多的情况下，MDRT、RDRT策略揭示所有故障的测试效率均比RT、RPT、DRT策略的测试效率高。在grep实验中，RPT、DRT、MDRT、RDRT策略的测试效率不如RT策略。原因可能如下：根据表2(a)，(b)，(c)中的数据，杀死前两个故障RT策略平均需要270.45个测试用例，杀死所有的故障平均需要1486.95个测试用例。表1中可以看出grep实验的测试用例数目为809个，因此grep实验中的故障很难被揭示，也就是说有很少的测试用例能够揭示软件中的故障，很有可能使得只有在个别分区中才能检测出软件中的故障，并且这些分区的失效率很低。如果引起故障的输入分散到整个输入域中，那么能够揭示软件故障的分区的失效率更低。当每一个分区的失效率都很低时，即便选中有可能揭示故障的分区，由于失效率很低，揭示软件中故障的概率也很小。因此当存在“不容易”揭示的故障时，RPT、DRT、MDRT、RDRT策略不如RT策略。但是MDRT、RDRT策略的测试效率仍然高于RPT、DRT策略。

6结论和将来的工作

动态随机测试是一个旨在利用历史的测试信息动态改变测试剖面的测试策略。DRT策略的主要优点是测试剖面不断变化，使得较高失效率的分区具有更高的被选择概率。但是DRT策略的测试效率受分区数目、初始剖面这些外部因素的影响。同时DRT策略的测试效率也受内部机制的影响：该策略根据某一个分区的执行结果调整所有分区被选择的概率并且所有的分区调整概率的幅度都相同。本文结合Markov链的状态转移矩阵提出了MDRT策略解决DRT策略的内部机制的不恰当问题。由于传统的DRT策略的参数取值普遍很小，并且分区被选择的概率容易受其它分区测试结果的影响使得找出具有较高故障检测能力的分区的速度较慢。本文提出基于奖惩机制的RDRT策略解决这一问题。针对DRT策略的两个外部影响因素本文为每一个实验设置了不同数目的分区，并且为每一种分区方式设置均等的初始概率分布和不均等的初始概率分布作为初始剖面。通过对5个真实的程序进行测试，均等的初始概率分布作为初始剖面以及数目较多的分区方式DRT、MDRT、RDRT具有更高的故障检测效率。在均等的初始概率分布作为初始剖面以及分区数目较多的分区方式下各个策略的测试效率结果分为三个方面：1.当软件中存在“很难”检测到的故障时，MDRT、RDRT策略的测试效率相对较低，DRT策略具有很高的提升空间。2.当软件中的故障都很容易被检测出来时，DRT、MDRT、RDRT的测试效率收到限制，但是MDRT、RDRT策略的测试效率仍然比DRT、RPT高。3.当软件中的故障“不是很难”被检测到时，DRT、MDRT、RDRT策略比RT、RPT策略具有更高的故障检测效率，并且MDRT、RDRT策略比DRT、RPT策略的测试效率更高。因此可以总结出MDRT、RDRT策略比DRT策略具有更高的故障检测能力。

但是MDRT策略中参数应当满足，因为在实际情况中输入造成的故障要比输入没有造成故障少。RDRT策略中惩罚上限的设置不同的实验可能取值不同，对策略的测试效率也有影响。将来的重点工作是研究MDRT策略中的参数以及RDRT策略中的惩罚上限进一步提高MDRT、RDRT策略的测试效率。

**7 参考文献**

[1] R. Hamlet “Random Testing”

[2] W. J. Gutjahr “Partition Testing vs Random Testing”

[3] R. Hamlet “Partition Testing Does Not Inspire Cinfidence”

[4] T. Y. Chen “A more general sufficient condition for partition testing to be better than testing”

[5] A. G. koru “Theory of relative defect proeness”

[6] data diversity: an approach to software fault tolerance

[7] G. B. Finelli “NASA Software Failure Characterization Experiments”

[8] K. Y. Cai “Random testing with dynamically updated test profile”

[9] K. Y. Cai “Optimal software testing and adaptive software testing in the context of software cybermetics”

[10] Junpeng Lv “A Sufficient Condition for Parameters Estimation in Dynamic Random Testing”

[11] K. Y. Cai “Adaptive Software Testing with Fixed-Memory Feedback”

[12] K. Y. Cai “Partition Testing with Dynamic Partitionint”

[13] The art of software testing

[14] An evaluation of random testing

[15] Analyzing partition testing stragegies

[16] Partition testing does not inspire confidence跟[3]是一样的

[16] A more general sufficient condition for partition testing to be better than testing跟【4】一样。

[16] on some reliability estimation problems in random and partition testing

[17] partition testing vs random testing

[18] comparing partition testing and random testing via majorization and Schur functions

[19] Optimal software testing and adaptive software testing in the context of software cybernetics.

[20] A History-Based Dynamic Random Software Testing

[21] Approach for Test Profile Optimization in Dynamic Random Testing

[22] Dynamic Random Testing with Parameter Adjustment

[23] A Sufficient Condition for Parameters Estimation in Dynamic Random Testing

[24]CPM