基于马尔科夫链的动态随机测试与奖惩机制的动态随机测试研究

技术报告

代贺鹏

# 1问题阐述

软件测试是保证软件质量的重要而有效的手段，已成为软件工程中重要且不可或缺的一个环节。随机测试（RT）[1]和分区测试（PT）[2]以其无偏性，自动化高，速度快和易于实现等优点使其成为近年来软件测试领域的一个研究热点。然而，RT、PT没有充分利用当前测试用例信息以及之前的测试用例信息，在某些情况下会降低测试效率。基于这一点提出了基于Markov链的动态随机算法(MDRT),利用上一步的测试结果指导下一步测试用例的选择。以往的动态随机算法每一次根据上一步的测试结果同等幅度的调整分区对应的概率，是不公平的（一般情况下初始概率为均等分布）。分区对应的概率大，说明该分区更能有可能发现软件中的缺陷，但是即便是失效率很大的分区也不可能每次挑选的测试用例都检测出缺陷。因此每次改变分区对应的概率应该考虑到自身当前概率的大小(以往等幅度更改)。将这一机制加入MDRT算法之中。

动态随机测试（DRT）[3]相较于传统的测试方法：随机测试、分区测试，有了很大的提高。DRT的核心思想是：根据当前的测试信息，更新测试剖面，以便使具有更高检测错误能力的分区被选择的概率增大，反之被选择的概率减少。当一个测试用例检测到错误时，那么这个测试用例附近的测试用例具有较高的可能检测出错误[4,5]，但是在DRT中，较高检测错误能力的分区被选择的概率增大的速度比较慢，同样具有较低检测能力甚至没有检测能力的分区被选择的概率减少的太慢，并且在根据其它分区的测试结果调整概率时还会增大具有较低检测能力的分区被选择的概率，导致具有较高检测能力的分区不容易凸显出来。为了弥补这方面的缺陷，提出了基于奖励、惩罚机制的Reward Dynamic Random Software Testing (RDRT)算法。在RDRT算法中，当某个分区中的测试用例找到了缺陷，与该分区绑定的奖励因子增加，惩罚因子降为0，并且下一次的测试用例还在这个分区之中随机挑选直到没有找到缺陷为止，然后根据该分区对应的奖励因子调整该分区被选择的概率；当分区中的测试用例没有找到缺陷，那么惩罚因子增加，如果该分区对应的惩罚因子累积到某一个值时，该分区的概率为0。通过奖励惩罚机制，希望尽可能快的找出具有较高检测能力的分区，同时大大减少没有检测能力或者检测能力很弱的分区被选择的概率。

有很多的因素影响测试算法在软件测试中的测试效率。在诸如DRT、MDRT、RDRT等算法中，都包含一个或者多个参数，不同的参数使得测试算法在同一个软件测试过程中具有不同的测试效率。在一般情况下，怎么样设置各个算法的参数大小本文通过5个实验进行研究。另一方面，在分区的基础上提出的一系列测试算法例如：RPT、DRT、MDRT、RDRT等，针对同一个测试对对象，对同一个输入域，不同数目的分区对测试有着显著的影响。怎么划分分区使得测试算法具有较好的测试效率，本文通过5个实验进行研究。对软件的输入域划分分区之后，怎么样设置对应分区的初始概率使得测试算法具有较好的测试效率，本文通过5个实验进行研究。

# 2相关理论

## 2.1 随机测试(RT)与随机分区测试（RPT）

随机测试和分区测试是两个主要的软件测试方法。随机测试在选取测试用例时一般按照均等的概率或者根据功能剖面来随机选取测试用例集中的测试用例。所谓功能剖面，是根据待测软件不同输入数据的使用频率人为设定概率-使用频率高的输入数据对应的测试用例则有较高的选取概率，使用频率低的输入数据赋予较低的选择概率。随机测试选取测试用例的方法较为简单，有助于估计待测软件的稳定性与可信性。分区测试则试图把测试用例集分为若干个不交叉的分区，每一个分区都包含若干个测试用例，一个接一个从不同的分区中选择测试用例进行测试。

RandomPartition(RPT)[6,7]算法是一种结合了随机测试和分区测试特点的新的测试策略。RPT将整个软件的测试用例集按照一定的方法分成m个的分区，每一个分区里面有个测试用例。RPT选择测试用例分成两个步骤：一，按照测试剖面选择分区；二，从被选择的分区中按照等可能概率随机选择测试用例。并且在测试过程中测试剖面不变。



## 2.2 动态随机测试（DRT）

不同的分区方法，在软件测试过程中，结果也不同。但是通过分析不同分区之前的检测情况，可以发现在特定的分区方式下各个分区检测失效的能力大小。动态随机测试的提出是为了提高随机测试（RT）的执行效率，K.Y.Cai等人引入了控制理论[8]，把软件测试看成一个控制问题，软件作为控制对象，执行策略当做控制者。在这种情况下，测试中的软件和预置对应的策略形成了一个紧密的反馈环。DRT算法[10]就是由此思想提出来的，主要的优点是基于当前测试信息动态地改变测试剖面。正如[3]中描述的，如果在一个分区中检测到错误那么在这个分区中有较高的可能性再次检测出错误。DRT算法描述如下。

假定测试用例集分为m个分区，测试用例集中共有k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。

步骤1 给参数赋一个初值，初值的取值范围为。

步骤2 根据每一个分区所对应的概率随机选取一个测试子集，在这里，。

步骤3 等概率随机地从测试子集中选取一个测试用例t。

步骤4 执行选中的测试用例t。

如果测试用例t找出了缺陷，就增大测试用例t所在分区对应的概率，同时减小其他分区被选择的概率，并把缺陷移除。



如果测试用例t没有找到缺陷，就减少被选中的概率，同时增大其它子集被选中的概率。



步骤5 检查停止条件，如果不满足，则跳转执行步骤2，如果满足则停止测试。

DRT算法通过在测试的过程中，动态调整各个分区对应的概率，使得具有较高检测能力的分区更有机会被选择，提高了软件测试的测试效率。

## 2.3 基于历史信息的动态随机测试（DRT-h）

一般情况下，DRT比RT、RPT有更高的测试效率，它的核心思想是根据测试信息动态改变测试剖面，以便使具有更高检测能力的分区更容易被选择。但是DRT根据当前测试信息改变测试剖面并不是严格的。假设一个分区之中只有一个测试用例可以揭示软件之中失效，执行这个测试用例后，这个分区对应的概率增加了，但是这个分区中的测试用例并不能再揭示软件之中的失效。为了弥补DRT算法中的这一缺陷K.Y. Cai提出了基于历史的动态随机测试（DRT-h）[9]。

在原来DRT的测试理论下，执行测试，并且记录结果然后计算各个分区的失效率，其中为在h次实验中分区被挑选次中检测出错误的次数。DRT-h记录每一个分区的失效率，然后比较每一个分区的失效率，增加最大失效率对应的分区的测试剖面，减少其它的分区的测试剖面。

DRT-h算法不仅仅考虑了当期执行测试用例的结果信息，还考虑了之前执行测试的结果信息。由此可以比较全面的、合理的调整各个分区对应的概率。但是在整个测试过程中由于记录了所有的测试信息，系统的开销太过于庞大，执行测试过程中所需要的时间也较之RT、PT大的多。

## 2.4 变异测试

变异测试[9]是一种基于错误的软件测试技术。它的基本思想是：给定一个程序和一个测试用例集，通过变异算子为产生一组变异体（合乎语法的变更），对于和都是用测试用例集中的测试用例进行测试运行，如果某在某个测试输入上与产生了不同的结果，则该被杀死；若某在所有的测试数据集上都与P产生相同的结果，则称其为活的变异体。接下来对活的变异体进行分析，检查是否等价于；对不等价于的变异体进一步测试，直到充分性度量达到满意程度。变异测试的基本流程如图2-1：

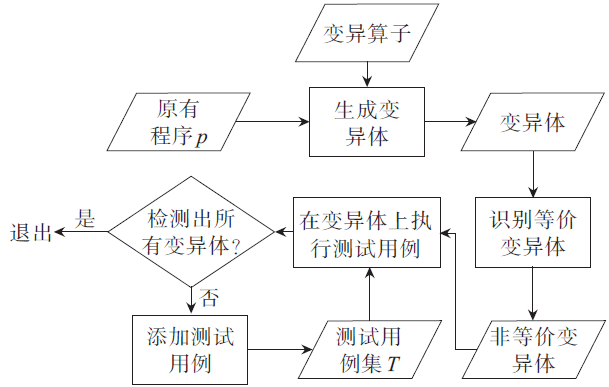


图2-1 变异测试流程图

# 3基于Markov的动态随机测试与奖惩机制的动态随机测试研究

## 3.1 Markov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我们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i,j有关，而与n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P为转移矩阵。

转移矩阵P具有如下性质：



## 3.2 基于Markov的动态随机测试算法（MDRT）

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。MDRT算法的思想：将每一次选中的分区当成当时所处的状态，根据该分区测试用例的执行结果调整到下一个状态即分区的概率。如果当前选中的分区之中测试用例检测出软件中存在失效，那么就增大下一步依然在该分区选择测试用例的可能，减少在其它分区选择测试用例的可能；反之，如果没有检测出软件中存在失效那么就减少该分区下一步扔在此分区选择测试用例的可能，增大在其它分区选择测试用例的可能。另外，每一个分区每次增大或者减少的力度应该是不同的。一般情况下，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。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缺陷，那么以后的检测过程中在增大或者减少该分区被选择的概率时应该考虑这种情况，即增大或减少某一分区被选中的概率时以当时分区被选中的概率相关。整个过程的状态可以用转移矩阵表示。假设软件测试输入域中的测试用例分到m个分区之中，输入域中共有k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。在整个软件测试过程中，如果将每个时间点t（t=0,1,2,…）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，将每一个时间点测试用例的执行状况按照某种测试策略调整对应分区的转移概率作为一次决策行动，各个时刻按照某种测试策略选取测试用例的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t+1的状态。因此，整个测试过程形成一个Markov决策过程，如图1所示。



图2 Markov决策过程

开始时刻设置每一个状态转移到其它状态的概率时相同的，（）为转移矩阵，初始矩阵如下：



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(第一次选择分区以作为转移概率)随机选取下一个分区，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t,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所在分区转移到本身的概率，同时减小转移到其它分区的概率，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所在子集转移到本身的概率，同时增大转移到其它分区的概率：



更新转移矩阵：



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。

## 3.3 奖惩机制的动态随机测试算法（RDRT）

基于奖惩机制的动态随机测试算法是在动态随机测试（DRT）的基础上提出来的。传统的DRT算法引入了软件的控制理论，如果某一分区中的测试用例检测出软件之中存在缺陷，那么该分区具有较高的可能性再次检查出缺陷，因此增大该分区被选择的概率。反之，减少该分区被选择的概率。在实际的测试中比RT和RPT具有更高的检测效率。但是该算法也存在一些不足之处，例如，每个分区被选择的概率根据每一次的测试结果进行改变，这种做法是不严格的。在另一方面，具有较高检测能力的分区被选择的概率容易受到其它分区的测试用例执行结果的影响，这样有可能使得具有较高检测能力的分区对应的概率减少。具有较低检测能力或者没有检测能力的分区也容易受到别的分区测试用例执行情况的影响，使得这样的分区对应得得概率增大或者维持比较高的水平。为了加速在软件测试的过程中挑选出具有较高的检测能力的分区和具有较低或者没有检测能力的分区的过程，本文提出了基于奖励惩罚机制的动态随机算法。该算法的思想是：如果某一个分区对应的测试用例检测出软件中存在失效，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失效为止。奖励因子越大该分区对应的倍选择的概率增加的就越多。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。

RDRT的算法过程如下：

假定软件测试的输入域中的测试用例分到m个分区中，输入域中共有k个测试用例，用来表示这m个子集，每一个子集有个测试用例。起初每个子集的奖励因子，惩罚因子。

步骤1：根据当前各个子集所对应的测试剖面选取子集，在这里。转步骤2。

步骤2：等概率随机地从子集中选取一个测试用例t,转步骤3。

步骤3：执行选中的测试用例t。如果测试用例t找出了缺陷转步骤4，反之转步骤5。

步骤4：子集的奖励因子，惩罚因子，并移除缺陷。转步骤2。

步骤5：子集的惩罚因子。如果就增大测试用例t所在子集对应的概率，同时减小其他子集被选择的概率：



如果，就减少该子集对应的概率，同时增大其它子集对应的概率，如果该分区的惩罚因子(为认为设定的惩罚上界)：



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。

## 3.4 停止性分析

在软件测试的过程中，很大程度上都是依赖某种执行测试用例的策略进行测试，但是什么时候停止测试，却是一直让人思考的问题。Chen在[10]向我们列举了几种软件测试的停止条件。

在实际的工程中，往往是测试用例发现了软件之中的缺陷，就停下来进行调试，在找到缺陷并且排除以后继续运行测试用例，对比测试用例的执行结果与预期的结果是否一致，检测软件中是否存在其它的缺陷。因此，在利用某种策略选择测试用例的时候，可以发现一个软件缺陷就停下来。

在实际的工程中，由于项目往往比较大执行一个测试用例所花费的时间比较久，并且之前经过了严格的把关，软件中的存在的缺陷并不多，或者当前的测试用例集并不能确定可以揭示出目前软件中存在的缺陷。如果一直运行下去或者等测试用例揭示软件中的缺陷往往是恰当的做法。这样的情况下，就可以预先设定一个时间点，到达规定的时间停止测试。

另一方面，软件的测试不可能达到穷尽测试，因此在实际的工程中，往往也可以采取测试用例揭示了一定数目的软件中的缺陷，或者整个软件测试输入域中的测试用例集数目执行到某一个临界值作为停止条件。

根据实际中工程的不同、具体情况的不同，要采取不同的测试停止方案。

## 3.3复杂性分析（空间、时间）

算法的复杂性的高低体现在运行该算法所需要的计算机资源的多少上，所需要的资源越多，该算法的复杂性越高；反之，所需要的资源越少，该算法的复杂性越低。计算机的资源最重要的是时间和空间资源。这个部分我们主要研究MDRT与RDRT的时间复杂度和空间复杂度。

### 3.3.1 时间复杂度

一个算法执行所用的时间，理论上讲是不能通过计算得出来的，因为它受多方面的影响，比如说不同的硬件，相同的算法在不同的硬件机器上执行，所消耗的时间是不同的。即使是在同一台机器上，一个算法在不同的时间执行，所消耗的时间也是不同的(当某个时刻计算机系统待处理的任务比较多时，这个时刻算法执行消耗的时间相对于计算机系统待处理任务较少时，所用的时间可能会多些)。“时间复杂度”并不是为了计算算法执行所消耗的时间，而是用于评定不同的算法之间在时间成本上，哪个消耗的时间理论上少些，那个多些。一个算法执行所消耗的时间等于算法中所有语句执行的时间之和。如果独立机器的软，硬件。假定语句执行一次所消耗的时间一样，并把语句执行一次所消耗的时间定义为单位时间。由于算法所消耗的时间与所有语句的执行时间有关，那么要先确定各个语句的执行时间。算法中每条语句的执行时间等于该语句的执行次数(语句频度)与单位时间的乘积。所以一个算法执行所消耗的时间等于算法中所有语句的语句频度与单位时间的乘积。

### 3.3.2 渐进时间复杂度

不妨设一个算法的语句频度为T(n)。如果能找到一个函数f(n),使当n趋于无穷大时，T(n)/f(n)的极限值为一个不等于零的常数，则称f(n)是T(n)的同数量级函数.并记T(n)=O(f(n))为算法的渐进时间复杂度。

### 3.3.3 MDRT算法和RDRT算法的时间复杂度分析

对于MDRT算法我们不妨设在一次软件测试过程中将软件的输入域分为m个分区。在理想情况下，一个测试用例就可以找出所有的缺陷，这时算法的语句频度为T(n)=m+1。由于在实际的情况中分区的个数总是有限的，因此这时算法的渐进时间复杂的为。如果不能一个测试用例就找出所有的缺陷，那么不妨设在整个测试的过程中共执行了n个测试用例，这时的语句频度为。当h很大时，这时的时间复杂度为。

相似地，对RDRT，在理想情况下，一个测试用就找出所有的缺陷，这时的时间复杂度为。如果执行的测定用例的数目比较多时，时间复杂度为。

### 3.3.4 空间复杂度

空间复杂度(Space Complexity)是对一个算法在运行过程中临时占用存储空间大小的量度。一个算法在计算机存储器上所占用的存储空间，包括存储算法本身所占用的存储空间，算法的输入输出数据所占用的存储空间和算法在运行过程中临时占用的存储空间这三个方面。算法的输入输出数据所占用的存储空间主要是由要解决的问题决定的，是通过参数表由调用函数传递而来的，它不随本算法的不同而改变。存储算法本身所占用的存储空间与算法书写的长短成正比，要压缩这方面的存储空间，就必须编写出较短的算法。算法在运行过程中临时占用的存储空间随算法的不同而异。

算法的空间复杂度通过计算算法所需的存储空间实现，算法空间复杂度的计算公式记作：S(n)= O(f(n))，其中，n为问题的规模，f(n)为语句关于n所占存储空间的函数。一般情况下，一个程序在机器上执行时，除了需要存储程序本身的指令、常数、变量和输入数据外，还需要存储对数据操作的存储单元。若输入数据所占空间只取决于问题本身，和算法无关，这样只需要分析该算法在实现时所需的辅助单元即可。若算法执行时所需的辅助空间相对于输入数据量而言是个常数，则称此算法为原地工作，空间复杂度为O(1)。

### 3.3.5 MDRT算法与RDRT算法空间复杂度分析

对于MDRT算法，该算法的输入与输出只与问题本身有关，与算法本身无关。因为，对不同的软件进行测试所需要的测试用例一般是不同的。因此这里主要分析算法的辅助空间。不妨设在一次软件测试过程中，将软件的输入域划分为m个分区，那么该算法的辅助空间主要为m\*m的矩阵以及每次执行测试用例时，记录的执行结果。因此该算法的空间复杂度为。

相似地，对RDRT算法该算法的输入与输出只与问题本身有关，与算法本身无关。因为，对不同的软件进行测试所需要的测试用例一般是不同的。因此这里主要分析算法的辅助空间。不妨设在一次软件测试过程中，将软件的输入域划分为m个分区，那么该算法的辅助空间主要为长度为m的数组以及每次执行测试用例时，记录的执行结果。因此该算法的空间复杂度为。

# 4实验工具的设计与实现

本章介绍我们开发的工具原型MT4W，支持不同算法的变异测试技术。首先分析与讨论MT4W的需求、系统架构与系统设计，然后利用一个java程序实例来演示工具的使用。

## 4.1需求分析

为了方便使用提出的RDRT、MDRT算法或者已有的算法，例如：RT、RPT、DRT等对软件进行测试，开发了一个工具原型，该工具提供以下功能：

* 获取测试用例
* 分区
* 选取测试用例
* 执行测试用例
* 生成报告

根据对上述功能需求的分析，绘制了PMRT的用例图，如图4-1所示。其中，各用例描述如下：

* 获取测试用例：获取测试用例是进行软件测试的第一步。该工具支持两种方式的测试用例获取方式。
* 自动生成测试用例：该方式需要用户知道软件的输入与输出的具体形式与取值范围，将每个参数的类型以及取值范围输入该工具中，并且输入要指明每个参数之间的关系，以便得到预期的正确结果。生成的测试用例会附带每个测试用例的预期输出，以便在测试的时候将程序执行的结果与预期结果对比，判断是否揭示了软件失效。
* 人工导入测试用例：该方式需要用户提前准备好测试用例集，并将所有的测试用例写入XML文档中，将该XML文档导入系统。人工导入测试用例需要用户判断每次的输出结果是否是正确的。

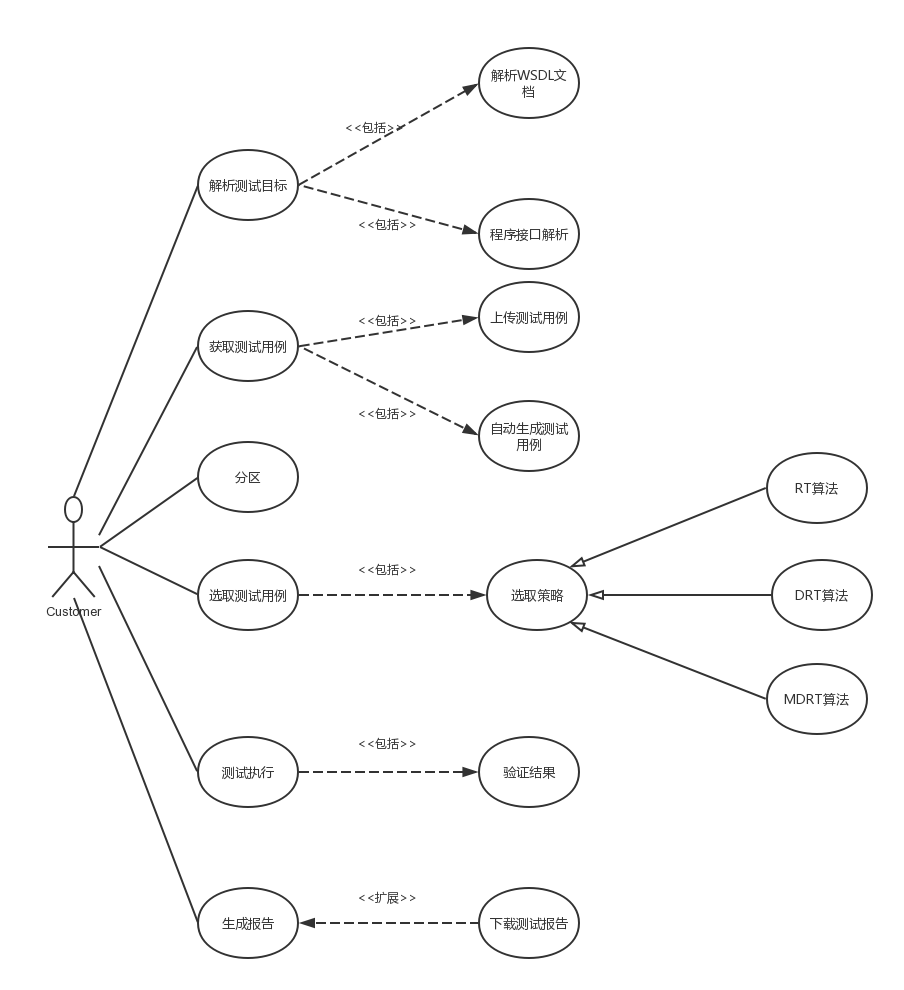


图4-1 MT4W测试工具用例图

* 分区：该功能的实现，需要用户编辑分区规则，将每一个分区各个参数的取值范围输入系统之中。每一个分区规则对应一个分区。系统将用户编辑的分区规则对应到上一步得到的测试用例集中。每一个分区对应一个XML文件。所有的XML文件由数字标识。
* 选取测试用例选取：测试用例的选取有两个因素决定。
* 不同的测试算法：根据用户选择的算法不同，选取测试用例的规则不同，RT算法不需要上一步的测试用例执行的结果信息。但是基于控制理论的算法，例如：DRT、RDRT、MDRT算法需要利用上一步测试用例的执行信息。
* RT算法：该算法是在上一步得到测试用例集中选取随机选取测试用例。
* DRT算法：将软件测试的过程看成一种控制过程，利用上一个测试用例执行的结果指导下一次选择测试用例。
* MDRT算法：该算法是本文提出的算法，利用了Markov链将来的状态只与当前状态有关这一思想，并结合状态转移矩阵，根据上一个测试用例的执行结果动态改变转移矩阵。根据当前所处的状态（分区）由转台转移矩阵选出下一个状态（分区）。
* RDRT算法：该算法是本文提出的算法，利用DRT每一次根据当前测试用例的执行结果调整分区对应的概率不严格，导致选出具有较强检测能力的分区比较慢这一不足，提出了一种基于奖励惩罚机制的动态随机算法。
* 对于基于分区思想的算法，例如RPT、DRT等算法，在选取测试用例之前必须先确定分区，然后在该分区中按照一定的方式选取测试用例。
* 执行测试用例：将上一步选出的测试用例，放在要检测的程序上执行，并每个测试用例判断输出的结果是否与预期的一致。如果一致则在测试报告中，该测试用例的测试结果标记为0；如果不一致，则说明该测试用例揭示了一个软件存在的缺陷，在测试报告中标记为1。
* 生成报告：当软件测试满足停止条件时，将自动生成整个过程的测试报告，以供用户下载。

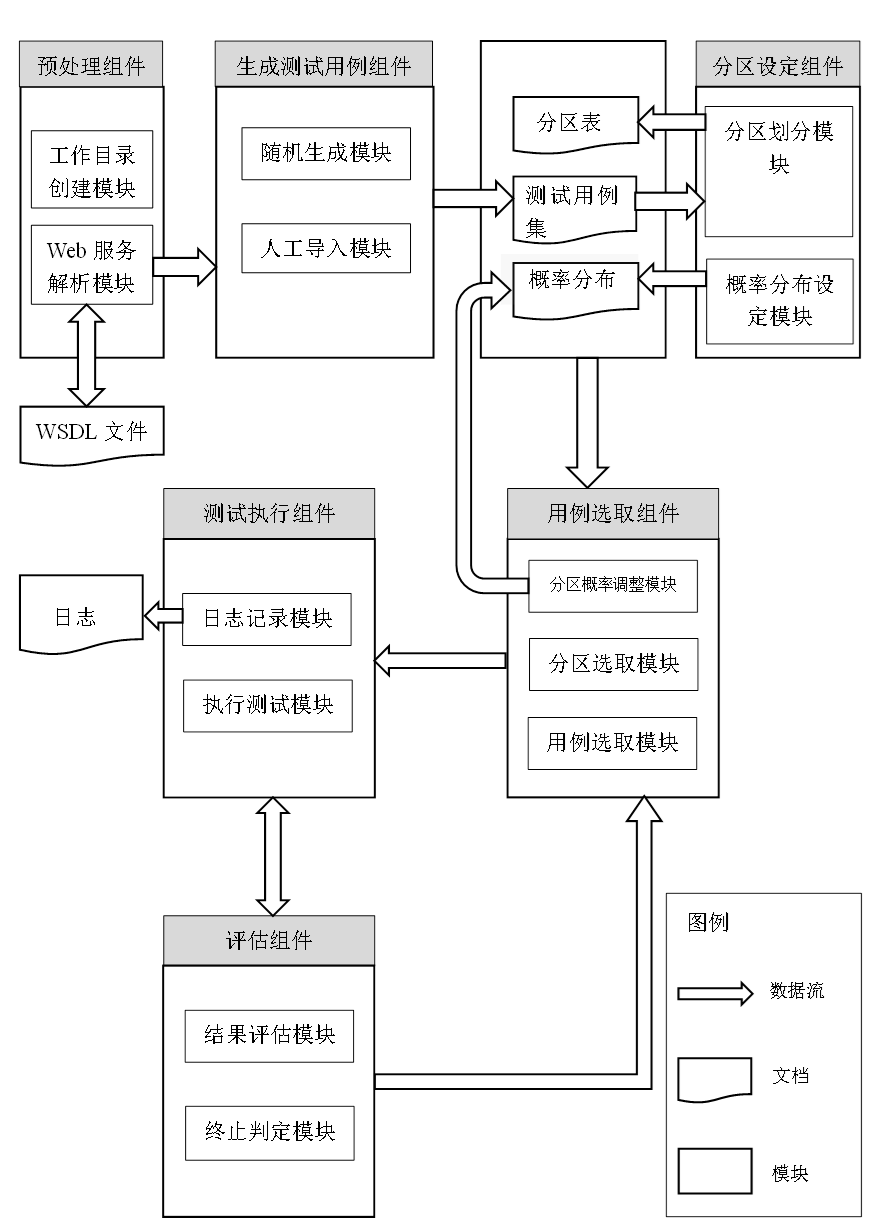
## 4.2 系统设计

根据上述需求分析，设计了基于控制流的变异体精简工具MT4W。本节介绍MT4W的系统架构以及各组件的设计。

### 4.2.1 系统架构

图4-2描述了MT4W工具的系统架构。使用灰色矩形框表示基本组件。每个组件有多个模块构成。本节对主要组件的功能及设计做简要概述。

图4-2 MT4W系统架构



### 4.2.2 预处理组件

预处理组件负责为待测Web服务创建测试项目目录以及解析待测服务的WSDL文件。

工作目录创建模块根据测试者提供的URI为待测Web服务创建测试目录。工具在测试过程中所产生的所有文档都存放在测试目录中。

Web服务解析模块根据URI获取Web服务的WSDL文件，解析服务描述中定义的所有操作的信息。操作信息包括：操作名，操作输入接口所包含的参数个数、参数名称、各参数的数据类型，操作输出接口的数据类型。

### 4.2.3 测试用例生成组件

生成测试用例组件负责生成测试用例并将测试用例集以文件的形式保存下来。

随机生成模块根据操作的信息，以随机的方式生成输入接口各个参数的值，并按照操作输入接口的格式封装成测试用例。所有测试用例组成测试用例集，以XML文件的形式保存在测试项目目录中。

人工导入模块接收XML格式的测试用例集文件，并根据待测Web服务的操作信息对文件内容进行验证。文件符合服务操作的接口格式时，人工导入模块将此文件复制到测试项目目录下；文件不符合接口格式时，导入模块将错误信息显示给测试者。对于其他格式的测试用例集文件的解析与验证，工具预留了扩展接口。

### 4.2.4 分区设定组件

分区设定组件负责对测试用例集进行划分并设定初始概率分布。

分区划分模块根据测试者提供的分区划分规则将测试用例生成组件产生的测试用例集划分成若干个分区。分区划分的结果是分区表，分区表中记录分区情况。

概率分布设定模块根据测试者的选择设定分区的初始概率分布。目前工具三种设定初始概率分布方式：测试者手动输入，平均分布和根据分区容量设定初始概率分布。分区的概率分布满足以下条件：N个分区，每个分区都不为空，则对应概率分布为，其中且。

### 4.2.5用例选取组件

用例选取组件负责根据测试者设定的测试策略，在测试过程中自动选取测试用例交给测试执行组件。用例选取组件是工具的关键组件。

用例选取组件选取测试用例的过程分为两步：

1）第一步，按照一定测试策略，从测试用例集中挑选符合策略标准的测试用例构成测试用例集合的子集；

2）第二步，从测试用例集合的子集中随机挑选一个测试用例，子集中每个测试用例被选中的概率均等。

分区选取模块是用例选取组件的核心模块，它负责从测试用例集合中挑选子集。它是具体测试策略在工具中的体现。目前工具提供两种分区选取模块——随机模块与动态随机模块。随机模块直接将整个测试用例集作为子集传递给用例选取模块；动态随机模块则根据概率分布选取一个分区，再查询分区表，选出对应分区中的测试用例形成子集传递给用例选取模块。

动态随机测试技术在用例选取组件中对应两个模块。动态随机模块负责根据分区概率分布和分区分布表挑选测试用例子集合；分区概率调整模块则根据每个测试用例是否发现错误来调整概率分布。

分区选取模块也是工具扩展的关键。测试者可以编写自己的分区选取模块，并在工具配置文件中“注册”此模块，就可以在工具中使用自己定义的测试策略。3.2节讨论具体分区选取模块的扩展方式。

### 4.2.6测试执行组件

测试执行组件负责执行测试用例，截取输出和记录日志。主要由两个模块组成：执行测试模块和日志记录模块。

执行测试模块将测试用例中的数据按照待服务的接口信息封装成服务请求发送给待测服务，然后接收服务响应，解析服务请求的结果并将结果传递给评估组件进行判断是否发现错误。本工具使用开源软件soapUI的API实现Web服务的调用。

日志记录模块负责在测试过程中的采集测试信息并记录。最终的测试报告从测试日志中提取并分析。

### 4.2.7评估组件

评估组件由两部分组成：结果评估模块和终止判定模块。

结果评估模块负责根据每个测试用例的实际执行结果与预期结果判断是否发现待测服务中的错误。

终止判定模块根据测试者选择的测试终止条件来判断测试过程是否完成。目前工具提供两种测试终止条件供测试者选择：一种是执行若干条测试用例后测试停止，另一种是发现若干错误后停止（具体多少个测试用例或多少个错误由测试者决定）。

## 4.3 MT4W工具的实现及演示

### 4.3.1 工具功能

MT4W系统实现了对Web服务的测试，并集成了5种选择测试用例的算法。该系统的测试用例可以人工导入也可以根据用户输入的参数类型以及范围随机生成。在形成测试用例集之后根据用户选择不同的算法确定是否需要分区，如果需要分区，由用户编辑分区规则，系统将根据用户编辑的分区规则将测试用例进行分区。然后按照某一个随机数种子产生的随机数序列，依次读取随机序列，直到读取的随机数对应的测试用例在选择的分区之中。接着执行测试用例，进行结果分析，形成测试报告。

### 4.3.2界面演示

界面设计主要通过对Java swing中控件的继承来实现，Swing使用了高度模块化的架构，可以通过接口的方式使用各种定制框架来扩展Swing。选择测试用例的算法部分用了设计模式-策略模式，因此用户可以根据自己的需求重写算法的父类定义自己的算法。工具的初步设计思想如下所示：

首先进入系统的登录界面图4-3，输入要解析的Web服务的地址，点击“解析”按钮按钮，进入系统；选择“重置”则清空服务地址重新输入。

点击“解析”按钮之后进入操作选择界面图4-4，用户可以选择要测试的Web服务的功能，然后填写测试的功能的参数信息。再由用户选择不同的算法，用户可以选择该工具自带的5中算法，也可以根据实际情况自己重写算法的实现类扩展算法。最后点击下一步，进入生成测试用例阶段。

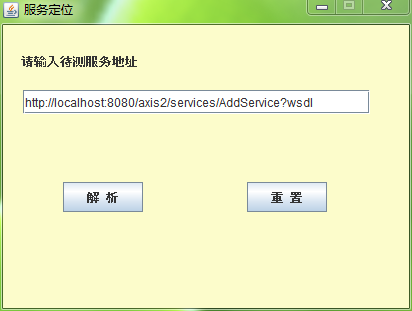


图4-3 MT4W系统登录界面

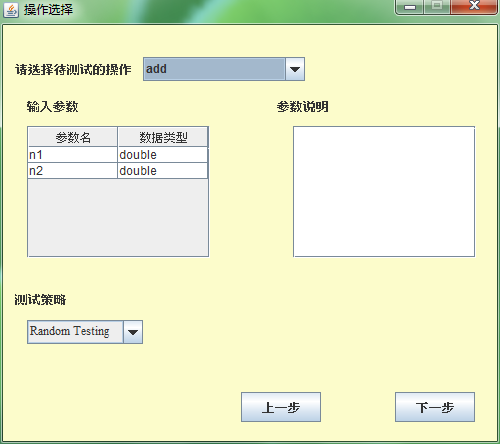


图4-4 MT4W系统操作选择界面

该工具实现了两种方式生成可以测试的测试用例集图4-5。一，人工导入测试用例，由用户提前准备好测试用例集，该测试用例集必须是XML文件，然后导入系统之中。二，随机生成测试用例，该方法需要用户和工具交互，用户编辑每一个参数的类型以及取值范围最后设定生成测试用例的数目，系统将根据用户的输入随机生成测试用例。

如果用户选择了基于分区的算法，用户需要编辑分区规则图4-6。点击“添加分区”按



图4-5测试用例生成以及分区界面

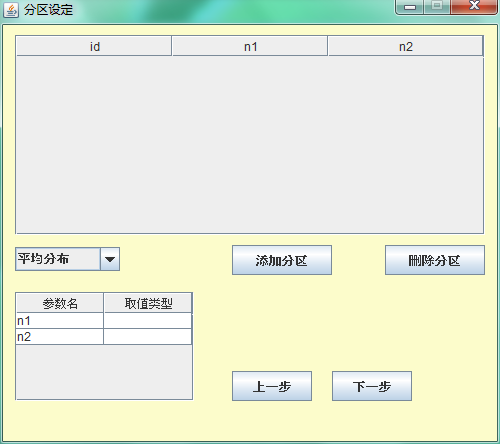


图4-6 编辑分区界面

钮，系统将给出所选择功能的参数列表，用户对每一个参数编辑分区规则，系统将根据用户的规则进行分区图4-7。



图4-7 编辑分区规则

用户编辑完分区规则之后。就可以点击“下一步”进入测试界面图4-8。该界面的

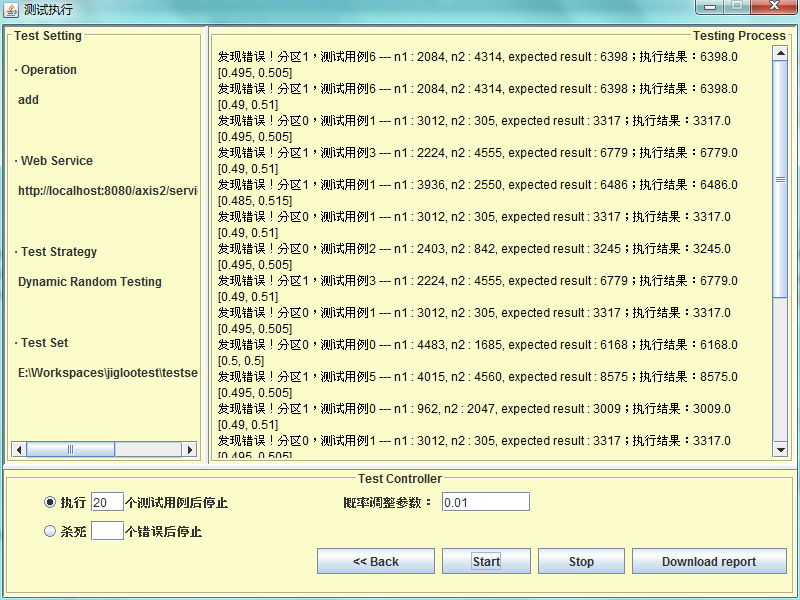


图4-8 测试界面

左边将显示有关测试对象的基本信息。在执行测试之前用户需要设定停止条件，该工具提供了两种方案供用户选择：一，执行一定数目的测试用例；二杀死一定数目的变异体。如果用户有不同的要求可以实现控制停止方案接口，实现满足需求的停止条件。右边部分将显示测试过程的一些基本信息，同时记录在后台，执行完测试之后，用户可以下载测试日志。

# 5实验评估

## 5.1实验目的

(1)评估与比较不同算法的性能

通过上面的讨论，介绍了MDRT与RDRT算法的思想与步骤，接下来通过5个真实的工程验证在一些情况下MDRT比RT、RPT具有更高的测试效率；RDRT比RT、RPT、DRT具有更高的测试效率。

在验证算法的有效性之外，算法本身存在一些影响其表现的因素：参数、初始概率分布、分区的多少。另外测试用例个数一定的情况下，各个算法的表现是不同的哪一种算法表现的更具有优势，也是讨论的主要目标。

实际的工作中，如果要把所有的软件测试输入域中的测试用例全部执行，是不太现实的，一方面很有可能执行到一定数目的测试用例就发现了一定数目的软件缺陷，另一方面执行所有的测试用例所需要的时间太多。因此本文通过每一个实验取一系列数目的测试用例比较各个算法杀死变异体的情况。

(2)影响性能的算法因素(算法参数的影响)

在MDRT算法与RDRT算法中均存在参数，参数不同，每次根据测试用例的执行结果调整各个分区对应概率的力度也不同。参数较大的情况下，每次调整的幅度就越大；参数较小的情况下，每次调整的幅度也会比较小。如何确定比较合适的参数范围使得不同的实验，测试效率都比较高，是本文讨论的目标之一。

(3)实验对象的影响(初始概率分布的影响)

由于软件测试所需要执行的测试用例数目往往比较大，因此根据每次执行测试用例的结果调整各个分区对应的选择概率相对于分区概率本身是很小的，因此初始概率的分布不同会对测试的结果产生很大的影响。在工业界一般采用均等分布和不均等分布两种方式。哪一种方式更恰当，在下文中会进一步讨论。

(4)实验对象的影响(分区大小影响)

从最开始的与路径测试将结合的PT测试到RPT测试，再到引入软件控制理论的DRT测试与ART测试。不管哪一种测试策略都需要对软件测试的输入域中的测试用例按照一定的方式进行分区。分区的方法有很多种，JOE在[11]中阐述了可以按照软件的不同的功能，将输入域中的测试用例划分到不同的分区之中。但是，在一个实验中既使是按照功能划分，但是也可以有不同数目的分区。比如银行储蓄系统，广泛的功能可以有存款、取款。但是仔细地划分又可以分为：定期储蓄、活期储蓄等等。分区的程度不同，对实验的影响也不同。那么在一般情况下，什么程度的分区算法都具有比较好的表现，在下文会进一步讨论。

## 5.2实验策略

### 5.2.1 待测源程序介绍

SIR网站[12]是一个提供Java、C、C#等多种语言编写的真实项目源代码的网站。该网站为软件测试与分析提供了帮助。SIR收录了大量典型的中小型软件产品线（SPL），并且每一种SPL都含有多个版本可供选择。每个版本都带有特定的变异体以及与之配套的测试用例。

为了尽量模拟实际工程上的情况，本文选取了SIR网站中提供的由C语言编写的5个真是存在的源程序：GREP、GZIP、FLEX、BASH以及MAKE。这5个待测源程序的LOC都在5000行以上，并且由于都是LINUX内核的一部分因此比较普及，选取这5个程序作为待测程序具有就较高的可信度。每个待测程序的基本信息如表5-1：

表5-1 待测程序的基本信息

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 源程序 | 代码行数 | 版本数 | 测试用例数目 | 变异体数目 |
| bash | 59846 | 6 | 1061 | 6 |
| flex | 10459 | 6 | 567 | 19 |
| grep | 10068 | 6 | 809 | 19 |
| gzip | 5680 | 6 | 217 | 16 |
| make | 35545 | 5 | 793 | 19 |

在选择要做的实验之后，从表1可以看出每一个程序都有不同的版本，这些版本一般都对应实际情况下项目的实际发行版本顺序。本文所有的实验都是选择的第二个版本作为实验对象。每个实验不同的版本都会带有\*.universe文件，这是生成该实验对应版本的测试用例脚本。通过在SIR网站上下载MTS工具对universe文件进行解析，可以生成能够执行的.sh文件。该shell脚本包含了所选择实验对应版本的大量可供执行的脚本语句。即，生成的测试用例脚本文件中若干行脚本语句就是一个测试用例，本文在做实验的时候将每一个测试用例都从脚本文件中取出来，单独形成一个可以执行的脚本文件，再根据每一个实验的具体分区，将所有的测试用例按照某些规则划分到不同的分区之中。再按照不同的策略选择分区，进而选择测试用例。

### 5.2.2 测试方法

本实验通过变异测试，在源程序的基础上植入变异体形成新的程序，然后选择测试用例在原始程序以及变异体上执行，对比执行的结果如果执行结果一直，说明该测试用例没有揭示程序中变异体的能力；如果输出结果不一致说明该测试用例发现了程序中的变异体。这是一个成功的测试。

本实验的变异体采用的是程序之中自带的变异体。每一个源程序的不同版本之中都会存在两个文件：version.seed文件、version.orig文件，这两个文件分别放置植入变异体的源程序以及原始源程序。在version.seed文件中放开对应变异体的权限，再经过编译之后形成特定的可执行变异体。逐步放开所有变异体的权限并进行编译之后，就可以得到所有变异体的可执行文件。然后选择测试用例，如果杀死了某一个变异体，就取消该变异体的执行权限，直到所有的变异体全部杀死。

### 5.2.3 评估策略

由于本实验MDRT算法的提出原因是为了改进RT、RPT算法实验没有考虑历史的实验信息，并且结合了软件的控制理论。RDRT算法的提出是在DRT算法的基础上，由于DRT算法每次根据测试用例的执行结果调整各个分区对应的选择概率并不是严格的，导致不能快速地揭示哪些个分区具有较强的检测能力，哪些分区具有很弱的检测能力或者没有检测能力。为了加速这个进程而提出了RDRT算法。因此，本文将实验分成两组：RT、RPT和MDRT算法是一组，MDRT算法与RT、RPT形成对照试验；RT、DRT和RDRT算法是另外一组，RDRT实验与RT、DRT实验形成对照试验。通过对比的形式说明那一种算法在软件测试中表现的更好。

通过对比的形式可以得出那种算法具有更高的检测能力，但是这种方式不直观，并且不能具体的表现出具有更高检测能力的算法相较于另一种算法提升的百分率。为了量化以上六种算法的测试效率，我们用到了三种度量标准：T-Measure,即，杀死所有变异体需要的测试用例；F-Measure，即，杀死第一个变异体所需要的测试用例；NF-Measure，即，杀死第一个变异体之后距离杀死第二个测试用例用的测试用例数目。在实际实验中，由于变异体较多往往少数几个个测试用例就能够找到变异体，不能显示不同算法之间的测试效率。为了模拟实际的测试过程我们挑选了相对不容易检测出来的变异体。假如一个变异体这里挑选出不容易检测出来的变异体时相对的。假设，一个程序的输入域中的测试用例比较少，大多数变异体只需要10以下的测试用例就可以检测出来，少数变异体需要10几个或者20几个测试用例才能够检测出来，那么就可以挑选出需要10几个或者20几个测试用例才能杀死的变异体。另一方面，如果一个程序的输入域中的测试用例比较少，大多数变异体需要10几个测试用例检测出来，少数变异体需要20几个甚至更多的测试用例才能检测出来，那么就选择需要20几个或者更多的测试用例才能检测出来的变异体。

在量化算法的优劣性时，我们采用(1)相对于RT算法的效率提升率来作为衡量的指标。





表示RT算法对应的度量标准大小，表示所选算法对应的度量标准大小。

通过观察不同度量标准的值，验证在不同的条件下MDRT比RT、RPT有较高的测试效率，RDRT比DRT与RT有较高的测试效率。下文提到的不同方法的提升率均是相对于RT算法的提升率。

## 5.3实验方案

前面已经介绍了5个待测程序的一些基本信息，本节将详细介绍每个程序的实验细节：检测测试用例、筛选变异体、分区方案、执行测试等等。

整个实验使用C、Java和shell脚本相结合的方法实现。算法的控制过程使用Java编写，环境配置、测试用例选择、测试用例执行以及统计结果使用shell脚本，随机数的产生使用C语言编写的。

所有的实验均均重复20次，每次实验的随机数种子是相同的，20次重复实验的随机数种子分别是1-20。由于每一个实验的测试用例个数都不一样，随机数的范围也不一样，因此不同的实验即便是随机数种子相同，产生的随机序列也是不同的。由于计算机中的随机数都是伪随机，如果不设定随机数种子，每一个随机数的选择都是依照当前的时间随机产生。这样做不仅仅导致了实验不可重复另外也导致算法之间的差异不能确定是由算法本身的机制产生的还是随机数的不同造成的。因此每一个实验不同重复次数可以选择不同的随机数种子，在特定次数的实验下，随机数应该是由相同的随机数产生的。

每一次的实验中，RT策略按照特定的随机数种子产生的随机数依次选择测试用例进行测试直到杀死所有的变异体。DRT、MDRT、RDRT策略在同一个试验下的特定次数的实验中，选择测试用例是和RT策略一样的随机序列。不同在于分区的选择是根据当时不同分区对应的概率随机选择。先由DRT、MDRT、RDRT策略根据上一步的执行结果调整各个分区对应的概率，然后随机选择分区（不设定随机数种子）。读取由随机数种子产生的序列，判断该测试用例是否在选择的分区之中，如果对应的测试用例在被选中的分区之中，就在源程序以及所有的变异体上执行该测试用例，然后对照结果；如果该测试用例不在被选择的分区之中，继续读取下一个测试用例，直到读取测试用例在被选择的分区之中。该过程的基本流程如图5-1：

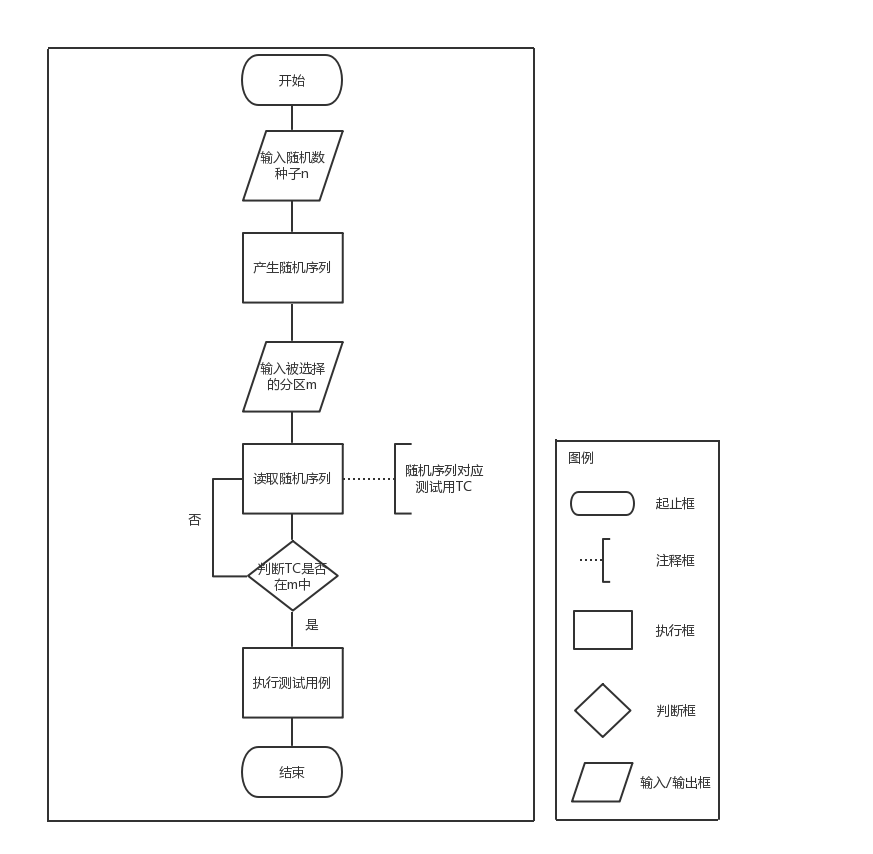


图5-1 DRT、MDRT、RDRT选择测试用例过程

在设置DRT、RDRT参数时，对于算法本身而言参数应当（为分区中的测试用例发现缺陷时的参数，为分区中的测试用例没有发现缺陷时的参数），因为测试用例发现缺陷的频率要低于没有发现缺陷的频率。但是对于不同的实验怎么设置、，以及它们的相对大小是一个很不好决定的事情，为了研究方便本文实验都是基于

### 5.3.1 grep实验细节

#### 5.3.1.1 grep简介

Linux内核命令grep的全称为global search regular expression and print out the line,即，全面搜索正则表达式并把行打印出来是一种强大的文本搜索工具，它能使用正则表达式搜索文本，并把匹配的行打印出来。它是按照在一个或多个文件中搜索字符串模板。如果模板包括空格，则必须被引用，模板后的所有字符串被看作文件名。搜索的结果被送到屏幕，不影响原文件内容的形式工作的。它的执行格式如（2）所示：



在grep的specification中对grep的功能有比较详细的介绍，grep不同功能的实现主要是依靠设定不同的options来决定的。不同功能的options如表5-2所示。

表5-2 grep工具功能介绍

|  |  |  |
| --- | --- | --- |
| 功能模块 | 具体可选项 | 具体功能描述 |
| 正则表达式控制 | -E | 扩展正则表达式 |
| -F | 一个换行符分隔的字符串的集合 |
| -G | 基本正则 |
| 输出控制 | -m | 匹配最大数 |
| -b | 在打印的匹配行前面加上行号 |
| -n | 显示匹配行的行号 |
| -H | 当搜索多个文件时，显示匹配文件名的前缀 |
| -q | 不显示任何东西 |
| -s | 不显示错误信息 |
| 文本行控制 | -B | 打印匹配行本身以及前面几行 |
| -A | 打印匹配行本身以及后面几行 |
| 匹配控制 | -i | 匹配忽略大小写 |
| -v | 取相反的项 |

实际情况下，在Linux中grep的可选项远不止上面提到的几种，这里由于篇幅限制，列举了不同功能模块的个别可选项。

#### 5.3.1.2 grep测试用例检测

TSL[12]是一个测试规格说明语言，用来生成基于规格说明书的测试用例。由TSL规格说明可以形成一定数目的测试帧，然后再由生成的测试帧得到可以执行的测试用例。

本文选取grep源代码中的第二个版本进行试验，在实验之前测试用例的生成与检查是应该首先要做的事情。SIR网站提供的项目源代码中附带了不同版本的各自的测试用例脚本，这些测试用例是由TSL规格说明生成的。用该网站提供的测试用例解析工具mts对测试用例脚本进行解析后得到可以执行的测试用例。

#### 5.3.1.3 grep实验分区介绍

JOE在[11]中阐述了可以根据程序中实现的不同的功能进行分区。前面的章节本文对grep工具实现的功能进行了简单介绍，不成程度的归纳，可以得出不同数目的功能模块。为了探究分区数目对各个算法在测试中的影响本文设计了两种分区方案：一，对功能进行详细的划分表5-3；二，在详细划分功能的基础上对，对有些功能进行合并从而得到一种较为粗略的分区方式表5-4。

表5-3 grep实验分区方案一

|  |  |
| --- | --- |
| 分区号 | 功能描述 |
| 1 | 匹配控制 |
| 2 | 输出控制 |
| 3 | 输出行控制 |
| 4 | 文本行控制 |
| 5 | 正则表达式控制 |
| 6 | 没有可选项 |

表5-4 grep实验分区方案二

|  |  |
| --- | --- |
| 分区号 | 功能描述 |
| 1 | 匹配控制 |
| 2 | 输出控制 |
| 3 | 正则表达式控制 |
| 4 | 没有可选项 |

#### 5.3.1.4 grep 变异体选择

在源程序中，一共有20个变异体，对于每一变异体用随机测试的方法进行测试，重复20遍取均值结果如表5-5所示。

表5-5 grep实验杀死变异体信息

|  |  |
| --- | --- |
| 变异体编号 | 杀死变异体所需要的测试用例数目 |
| Muant\_1 | 825.05 |
| Muant\_2 | 杀不死 |
| Muant\_3 | 3.30 |
| Muant\_4 | 杀不死 |
| Muant\_5 | 杀不死 |
| Muant\_6 | 杀不死 |
| Muant\_7 | 341.65 |
| Muant\_8 | 268.55 |
| Muant\_9 | 杀不死 |
| Muant\_10 | 杀不死 |
| Muant\_11 | 1.10 |
| Muant\_12 | 杀不死 |
| Muant\_13 | 杀不死 |
| Muant\_14 | 748.45 |
| Muant\_15 | 杀不死 |
| Muant\_16 | 834.65 |
| Muant\_17 | 杀不死 |
| Muant\_18 | 杀不死 |
| Muant\_19 | 杀不死 |
| Muant\_20 | 杀不死 |

从表5-5中可以看出对于能够杀死的变异体，有些变异体很容易就能够被杀死，这样的变异体不能显现出不同算法之间的检测效率的差异，例如Mutant\_11、Mutant\_3。因此本文选取比较难杀死的变异体Mutant\_1、Mutant\_7、Mutant\_8、Mutant\_14以及Mutant\_16作为变异体。

#### 5.3.1.5 grep实验参数设定

由于DRT、RDRT、MDRT三种算法之中都带有参数，参数不同导致每次根据测试用的执行结果调整各个分区的选择概率力度不同。参数较大时，每次更改分区对应的概率的幅度就越大；参数较小时，每次更改分区对应的概率的幅度就比较小。由此会对分区的选择造成很大的差异性。为了探究算法中参数在哪一个范围内具有较高的检测能力，本文设置了一系列的梯度：DRT、RDRT参数如表5-6，MDRT参数如表5-7。

表5-6 DRT、RDRT算法参数列表

|  |  |
| --- | --- |
| 参数 | 取值 |
|  | 0.00001 |
|  | 0.0001 |
|  | 0.001 |
|  | 0.01 |
|  | 0.1 |
|  | 0.2 |
|  | 0.25 |
|  | 0.3 |
|  | 0.35 |
|  | 0.4 |
|  | 0.45 |
|  | 0.5 |
|  | 0.55 |
|  | 0.6 |
|  | 0.65 |
|  | 0.8 |
|  | 1 |

表5-7 MDRT参数设置列表

|  |  |
| --- | --- |
| 参数 | 取值 |
|  | 0.00001 |
|  | 0.0001 |
|  | 0.001 |
|  | 0.01 |
|  | 0.1 |
|  | 0.2 |
|  | 0.25 |
|  | 0.3 |
|  | 0.35 |
|  | 0.4 |
|  | 0.45 |
|  | 0.5 |
|  | 0.55 |
|  | 0.6 |
|  | 0.65 |
|  | 0.7 |
|  | 0.9 |
| punishment | 50 |

DRT、与RDRT算法的参数从1开始调整，根据以往的测试经验发现DRT与RDRT在[0.2,0.6]之间具有更高的检测能力。因此这个范围内设置的参数相对密集。当参数小于0.1之后，由于参数的数值相对较小，本文一次降低一个数量级直到0.00001。由于参数的数值到达0.00001之后已经特别小，每次根据测试用例的执行结果调整概率的幅度已经特别小，因此如果参数再比0.00001小就没有必要了。

MDRT算法的参数没有涉及到1，因为参数为1时如果一个分区内的测试用例没有检测出缺陷，那么该分区下次被选中的概率就为0。另一方面，如果参数比1小但是很接近1，那么如果一个分区中的测试用例没有检测出软件中缺陷，那么这个分区对应的概率相当小接近0，这显然不合理，因此本文在设置MDRT参数的上限时最高到0.9。根据以往的测试经验发现当参数取值在0.5附近是，具有较高的检测效率，因此本文在[0.2,0.7]之间设置的参数相对密集。由于参数的数值到达0.00001之后已经特别小，每次根据测试用例的执行结果调整概率的幅度已经特别小，因此如果参数再比0.00001小就没有必要了。由于该实验选择的测试用例都比较难杀死，因此惩罚因子取50。

#### 5.3.1.6 grep实验初始概率设定

将grep实验的测试用例按照前面阐述的分区规则分区之后，在测试之前并不知道哪些分区具有较高的检测能力，怎么设置分区对应的初始概率成为一个不容易解决的难题。一方面，一般来说测试用例多的分区，由于测试用例的基数大，更有可能发现软件中的缺陷；另一方面，测试用例少的分区也存在发现软件中缺陷的可能。针对以上两个方面，本文提出了两种设定初始概率的方式：一，按照每个分区的测试用例个数占所有测试用例的百分比作为该分区对应的初始概率，DRT、RDRT如（3）(4)，MDRT的初始状态转移矩阵每一行的状态转移概率和DRT的相同；二，由于含有较少测试用例的分区也有可能具有较高的检测能力，因此将初始概率设定为均等分布，DRT、RDRT如（5）、(6)，MDRT的初始状态转移矩阵每一行的状态转移概率和DRT的相同。



### 5.3.2 gzip实验细节

#### 5.3.2.1 gzip简介

Linux内核命令gzip是GNUzip的缩写，它在Linux系统中经常使用的一个对文件进行压缩和解压缩的命令，既方便又好用。gzip不仅可以用来压缩大的、较少使用的文件以节省磁盘空间，还可以和tar命令一起构成Linux操作系统中比较流行的压缩文件格式。据统计，gzip命令对文本文件有60%～70%的压缩率。减少文件大小有两个明显的好处，一是可以减少存储空间，二是通过网络传输文件时，可以减少传输的时间。它的执行格式如（7）所示。



在gzip的specification中对gzip的功能有比较详细的介绍，gzip不同功能的实现主要是依靠设定不同的options来决定的。不同功能的options如下表5-8所示。

表5-8 gzip工具功能介绍

|  |  |  |
| --- | --- | --- |
| 功能模块 | 具体可选项 | 具体功能描述 |
| 压缩文件 | -n | 压缩文件时不保存原来文件名称及时间 |
| -N | 压缩文件时保存原来文件名称及时间 |
| 解压缩文件 | -d | 解开压缩文件 |
| 获取信息 | -l | 列出文件的相关信息 |
| -h | 获取帮助信息 |
| -L | 获取版本信息 |
| 控制执行过程 | -t | 不执行，检查压缩文件是否完整 |
| -q | 不显示警告信息 |
| -v | 显示指令执行过程 |

实际情况下，在Linux中gzip的可选项远不止上面提到的几种，这里由于篇幅限制，列举了不同功能模块的个别可选项。

#### 5.3.2.2 gzip测试用例检测

TSL是一个测试规格说明语言，用来生成基于规格说明书的测试用例。由TSL规格说明可以形成一定数目的测试帧，然后再由生成的测试帧得到可以执行的测试用例。

本文选取gzip源代码中的第二个版本进行试验，在实验之前测试用例的生成与检查是应该首先要做的事情。SIR网站提供的项目源代码中附带了不同版本的各自的测试用例脚本，这些测试用例是由TSL规格说明生成的。用该网站提供的测试用例解析工具mts对测试用例脚本进行解析后得到可以执行的测试用例。

#### 5.3.2.3 gzip实验分区介绍

前面的章节已经介绍了可以根据程序中实现的不同的功能进行分区。之前本文对gzip工具实现的功能进行了简单介绍，不成程度的归纳，可以得出不同数目的功能模块。为了探究分区数目对各个算法在测试中的影响本文设计了两种分区方案：一，对功能进行详细的划分表5-9；二，在详细划分功能的基础上对，对有些功能进行合并从而得到一种较为粗略的分区方式表5-10。

表5-9 gzip实验分区方案一

|  |  |
| --- | --- |
| 分区号 | 功能介绍 |
| 1 | 压缩文件 |
| 2 | 解压文件 |
| 3 | 获取信息 |
| 4 | 执行过程控制 |

表5-10 gzip实验分区方案二

|  |  |
| --- | --- |
| 分区号 | 功能介绍 |
| 1 | 输出控制 |
| 2 | 获取信息 |
| 3 | 执行过程控制 |

#### 5.3.2.4 gzip 变异体选择

在源程序中，一共有16个变异体，对于每一变异体用随机测试的方法进行测试，重复20遍取均值结果如表5-11所示。

表5-11 gzip实验杀死变异体信息

|  |  |
| --- | --- |
| 变异体编号 | 杀死变异体所需要的测试用例数目 |
| Mutant\_1 | 杀不死 |
| Mutant\_2 | 杀不死 |
| Mutant\_3 | 18.15 |
| Mutant\_4 | 281.65 |
| Mutant\_5 | 杀不死 |
| Mutant\_6 | 杀不死 |
| Mutant\_7 | 15.95 |
| Mutant\_8 | 杀不死 |
| Mutant\_9 | 杀不死 |
| Mutant\_10 | 杀不死 |
| Mutant\_11 | 282.35 |
| Mutant\_12 | 杀不死 |
| Mutant\_13 | 杀不死 |
| Mutant\_14 | 2.30 |
| Mutant\_15 | 杀不死 |
| Mutant\_16 | 74.35 |

与之前的实验相似，挑选出很容易杀死的变异体Mutant\_14，因为这样的变异体很容易被杀死不能够体现出不同算法检测能力的不同。由此得到gzip实际做实验的变异体：Mutant\_3、Mutant\_4、Mutant\_7、Mutant\_16、Mutant\_11。

#### 5.3.2.5 gzip实验参数设定

本实验的DRT、RDRT参数与grep实验的相同。但是MDRT实验的参数不一致，参数信息如表5-12。

表5-12 gzip实验的MDRT参数列表

|  |  |
| --- | --- |
| 参数 | 取值 |
|  | 0.00001 |
|  | 0.0001 |
|  | 0.001 |
|  | 0.01 |
|  | 0.1 |
|  | 0.2 |
|  | 0.25 |
|  | 0.3 |
|  | 0.35 |
|  | 0.4 |
|  | 0.45 |
|  | 0.5 |
|  | 0.55 |
|  | 0.6 |
|  | 0.65 |
|  | 0.7 |
| punishment | 30 |

从表5-12可以看出，惩罚因子与grep实验不同，由于本实验的测试用例相对较少，但是存在个别测试用例需要200多个测试用例才能杀死的情况，因此本实验将惩罚因子取值为30。

#### 5.3.3.6 gzip实验初始概率设定

将grep实验的测试用例按照前面阐述的分区规则分区之后，在测试之前并不知道哪些分区具有较高的检测能力，怎么设置分区对应的初始概率成为一个不容易解决的难题。一方面，一般来说测试用例多的分区，由于测试用例的基数大，更有可能发现软件中的缺陷；另一方面，测试用例少的分区也存在发现软件中缺陷的可能。针对以上两个方面，本文提出了两种设定初始概率的方式：一，按照每个分区的测试用例个数占所有测试用例的百分比作为该分区对应的初始概率，DRT、RDRT如（8）(9)，MDRT的初始状态转移矩阵每一行的状态转移概率和DRT的相同；二，由于含有较少测试用例的分区也有可能具有较高的检测能力，因此将初始概率设定为均等分布，DRT、RDRT如（10）(11)，MDRT的初始状态转移矩阵每一行的状态转移概率和DRT的相同。





### 5.3.3 flex实验细节

#### 5.3.3.1 flex简介

Linux下的Flex是一个生成扫描器(scanner)的工具，生成的扫描器能够识别文本中的词法模式(lexical pattern)。Flex接受文本格式的Flex文件（扩展名可以为.l、.flx、.lex或者.flex）作为输入，生成一个c源文件：lex.yy.c，其中定义了一个函数yylex()，该函数就是扫描器。它根据Flex文件中定义的模式(pattern)对输入的文本串进行分析，然后执行对应的动作(Action)，该模式和对应的动作叫做规则。例如，可以定义一个模式识别自定义标识符，并在对应的动作中规定，如果遇到自定义标识符，将该标识符写入某个数组。另外，lex.yy.c可以编译后执行，也可以被其他源文件中的函数调用。它的执行格式如（12）所示。



在flex的specification中对它的功能有比较详细的介绍，flex不同功能的实现主要是依靠设定不同的options来决定的。不同功能的options如下表5-13所示。

表5-13 flex工具功能介绍

|  |  |  |
| --- | --- | --- |
| 功能块 | 具体可选项 | 具体功能描述 |
| 输出控制 | -b | 在lex.backup中生成备份信息 |
| -w | 禁止警告信息 |
| 扫描器产生控制 | -i | 生成一个不区分大小写的扫描器 |
| -B | 生成一个不交互的扫描器 |
| -7 | 生成一个7-bit的扫描器 |
| 扫描器运行控制 | -d | 在调试模式下运行 |
| -f | 快速地扫描 |
| -l | 扫描器的兼容性最大 |
| 获取信息 | -h | 获取帮助信息 |
| -p | 生成报告 |
| -V | 打印版本信息 |
| 仅扫描或者什么都不做 | -c | 什么都不做 |
| -n | 什么都不做 |
| 表压缩控制 | -C[aefFMr] | 控制表压缩 |

#### 5.3.3.2 flex测试用例检测

TSL是一个测试规格说明语言，用来生成基于规格说明书的测试用例。由TSL规格说明可以形成一定数目的测试帧，然后再由生成的测试帧得到可以执行的测试用例。

本文选取flex源代码中的第二个版本进行试验，在实验之前测试用例的生成与检查是应该首先要做的事情。SIR网站提供的项目源代码中附带了不同版本的各自的测试用例脚本，这些测试用例是由TSL规格说明生成的。用该网站提供的测试用例解析工具mts对测试用例脚本进行解析后得到可以执行的测试用例。

#### 5.3.3.3 flex实验分区介绍

前面的章节已经介绍了可以根据程序中实现的不同的功能进行分区。之前本文对flex工具实现的功能进行了简单介绍，不成程度的归纳，可以得出不同数目的功能模块。为了探究分区数目对各个算法在测试中的影响本文设计了两种分区方案：一，对功能进行详细的划分表5-14；二，在详细划分功能的基础上对，对有些功能进行合并从而得到一种较为粗略的分区方式表5-15。

表5-14 flex分区方案一

|  |  |
| --- | --- |
| 分区号 | 功能块描述 |
| 1 | 输出控制 |
| 2 | 扫描器产生控制 |
| 3 | 扫描运行控制 |
| 4 | 获取信息 |
| 5 | 什么都不做 |
| 6 | 表压缩控制 |

表5-15 flex分区方案二

|  |  |
| --- | --- |
| 分区号 | 功能块描述 |
| 1 | 输出 |
| 2 | 运行 |
| 3 | 什么都不做 |

#### 5.3.3.4 flex变异体选择

在源程序中，一共有16个变异体，对于每一变异体用随机测试的方法进行测试，重复20遍取均值结果如表5-16所示。

表16 flex实验杀死变异体信息

|  |  |
| --- | --- |
| 变异体编号 | 杀死变异体所需要的测试用例数目 |
| Mutant\_1 | 3.70 |
| Mutant\_2 | 杀不死 |
| Mutant\_3 | 4.35 |
| Mutant\_4 | 6.30 |
| Mutant\_5 | 3.70 |
| Mutant\_6 | 1.00 |
| Mutant\_7 | 6.45 |
| Mutant\_8 | 1.00 |
| Mutant\_9 | 1.05 |
| Mutant\_10 | 4.35 |
| Mutant\_11 | 1.50 |
| Mutant\_12 | 杀不死 |
| Mutant\_13 | 杀不死 |
| Mutant\_14 | 25.40 |
| Mutant\_15 | 1.05 |
| Mutant\_16 | 1.40 |
| Mutant\_17 | 1.70 |
| Mutant\_18 | 4.25 |
| Mutant\_19 | 2.15 |

与之前的实验相似，挑选出很容易杀死的变异体Mutant\_2、Mutant\_3、Mutant\_6等，因为这样的变异体很容易被杀死不能够体现出不同算法检测能力的不同。由于本实验的能够杀死的变异体普遍比较容易杀死，取相对难杀死的变异体，由此得到flex实际做实验的变异体：Mutant\_4、Mutant\_7、Mutant\_16。

#### 5.3.3.5 flex试验参数设定

本实验的DRT、RDRT参数与grep、gzip实验的相同。但是MDRT实验的参数不一致，参数信息如表5-17。

表5-17 flex实验MDRT参数列表

|  |  |
| --- | --- |
| 参数 | 取值 |
|  | 0.00001 |
|  | 0.0001 |
|  | 0.001 |
|  | 0.01 |
|  | 0.1 |
|  | 0.2 |
|  | 0.25 |
|  | 0.3 |
|  | 0.35 |
|  | 0.4 |
|  | 0.45 |
|  | 0.5 |
|  | 0.55 |
|  | 0.6 |
|  | 0.65 |
|  | 0.7 |
| punishment | 10 |

从表5-16可以看出，各个变异体被杀死需要的测试用例相对都不多，因此将惩罚因子的上限设置的相对较小，取值为10。

#### 5.3.3.6 flex试验初始概率设定

将flex实验的测试用例按照前面阐述的分区规则分区之后，在测试之前并不知道哪些分区具有较高的检测能力，怎么设置分区对应的初始概率成为一个不容易解决的难题。一方面，一般来说测试用例多的分区，由于测试用例的基数大，更有可能发现软件中的缺陷；另一方面，测试用例少的分区也存在发现软件中缺陷的可能。针对以上两个方面，本文提出了两种设定初始概率的方式：一，按照每个分区的测试用例个数占所有测试用例的百分比作为该分区对应的初始概率，DRT、RDRT如（13）(14)，MDRT的初始状态转移矩阵每一行的状态转移概率和DRT的相同；二，由于含有较少测试用例的分区也有可能具有较高的检测能力，因此将初始概率设定为均等分布，DRT、RDRT如（15）(16)，MDRT的初始状态转移矩阵每一行的状态转移概率和DRT的相同。

### 5.3.4 bash实验细节

#### 5.3.4.1 bash简介

bash是一个为GNU计划编写的Unix shell，或者说是一个命令解释器，它的名字是一系列缩写：Bourne-Again Shell。bash大体上兼容并且集成了ksh以及csh，成为符合了IEEE POSIX规范的一种命令解释器。在几乎所有的Unix和Linux版本上运行。

在SIR中下载的bash源代码中的测试用例，对bash的80个方面做了测试，测试信息如表5-18所示。

#### 5.3.4.2 bash测试用例检测

由于bash是一个命令解释工具，比较复杂，有的测试用例甚至需要配置环境才能够执行，本实验的测试用例不是写在.universe脚本之中，而是一个测试用例一个文本文件，一共1061个测试用例。所有的测试用例的生成方式为随机生成。

经进一步分析，所有的测试用例测试了有关bash的80个方面，每一个方面由若干个测试用例组成。具体情况如表5-18，由于每一个方面都是由若干个测试用例组成，因此表中只列举每一个方面的前缀。

表5-18 bash实验测试用例介绍

|  |  |
| --- | --- |
| 测试用例前缀 | 具体功能 |
| arith | 算术 |
| array | 数组 |
| bg | 控制脚本前台后台运行 |
| braces | 花括号测试 |
| builtins | 内建命令 |
| cd | 切换目录 |
| child\_trap | 在脚本中接受信号 |
| command | 输出一些信息 |
| countlines | 输出有多少行的内容 |
| disown | 后台进程 |
| dottest | 用于测试圆点的输出 |
| enable | 用于启动或者关闭内建命令 |
| env | 用来显示当前用户的环境变量 |
| environ | 环境变量 |
| exit | 退出当前的脚本 |
| exp | 导出工具 |
| fc | 显示文件内容或者调用一个编辑器修改 |
| file\_attrib | 修改文件属性 |
| function | 函数（shell脚本） |
| getopts | 获取选项参数 |
| glob | 用于linux系统的路径名称匹配 |
| hash | 记录一些操作信息 |
| myreadline | 读入文件内容 |
| myread | 读入文件内容 |
| myexpr | 求表达式变量的值 |
| myescapes | 输出 |
| mycontrol\_flow | 串口流控制 |
| mybash\_invoke | 系统调用和 |
| myalias | 设置指令的别名 |
| more-exp | 导出工具 |
| ls | 列出文件 |
| loop | 循环 |
| jobs | 用于查看当前终端后台运行的任务 |
| ifs | shell脚本字段分隔符 |
| histvars | 系统保存命令的数目 |
| heredoc | shell中一种特殊的重定向方式 |
| help | 显示shell内部命令的帮助信息 |
| run | 运行脚本文件 |
| rhs-exp | 系统输出 |
| redir | 系统端口输出 |
| read | 从标准输出一行到shell |
| quote | 管理用户对磁盘的操作 |
| prompt | 修改命令提示符 |
| printf | 格式化并输出 |
| precedence:shell | 脚本的运行 |
| posix | 系统消息列队 |
| pipe | 管道 |
| perf-script | 脚本 |
| nquote | 跟rhs-exp差不多 |
| new-exp | 新建一些变量 |
| mytrap | 在脚本中处理信号 |
| mytest | shell脚本环境测试条件表达式 |
| mytab | shell中脚本补全命令行 |
| myalias | 别名 |
| function | shell脚本函数 |
| builtin | shell内建命令 |
| time | 用于测试一个命令的运行时间 |
| tilde-test | 输出波浪字符 |
| test | shell脚本环境中测试条件表达式 |
| stty | 命令用于显示和修改终端行设置 |
| strip | 从特定文件中剥掉一些符号信息和调试信息 |
| string | 输出多字符情况 |
| special | 跟shell脚本有关 |
| spece | 打印空格 |
| shopt | 用于显示和设置shell中的行为选项 |
| set | 显示系统中已经存在的shell变量，以及设置shell变量的值 |
| builtin | 是有关声明的 |
| enable | 有关路径的输出路径 |
| getopts | case程序选择检查 |
| special | 声明 |
| exit | 退出系统 |
| file | 根据内容判断文件类型，使用权限是所有用户 |
| ls | 显示目录内容 |
| wildcard | 扩展通配符 |
| vars | 输出变量值 |
| varenv | 输出环境变量 |
| uname | 用来获取电脑和操作系统的相关信息 |
| umask | 设置文件权限 |
| ulimit | 用于限制shell启动进程所占有的资源 |
| type | 用来显示指定命令的类型 |

#### 5.3.4.3 bash实验分区介绍

本实验的分区方式是按照bash工具实现的不同功能分区的。从表5-18中可以看出1061个测试用例在80个方面进行了测试。对以上80组测试用例进行分析：可以概括为6个方面：文本操作控制、输出信息控制、shell脚本控制、系统环境探测与修改、逻辑运算、bash的内置命令。按照以上6个功能模块进行分区结果如表5-19。还可以对以上6个功能模块联合，例如shell脚本控制与逻辑运算这两个功能模块，在实际的shell脚本中一定会带有逻辑运算，因此可以将shell脚本模块与逻辑运算模块结合。系统环境的探测与修改也是通过bash命令操作的因此可以将这两个模块进行合并，得到另一种分区方案如表5-20。

表5-19 bash实验分区方案一

|  |  |
| --- | --- |
| 分区号 | 功能介绍 |
| 1 | 文本操作 |
| 2 | 输出信息 |
| 3 | Shell脚本 |
| 4 | 系统环境 |
| 5 | 逻辑运算 |
| 6 | bash内置命令 |

表5-20 bash实验分区方案二

|  |  |
| --- | --- |
| 分区号 | 功能介绍 |
| 1 | 文本操作 |
| 2 | 输出信息 |
| 3 | Shell脚本环境 |
| 4 | 系统bash环境 |

#### 5.3.4.4 bash变异体选择

在源程序中，一共有6个变异体，对于每一变异体用随机测试的方法进行测试，重复20遍取均值结果如表5-21所示。

表5-21 bash实验杀死变异体信息

|  |  |
| --- | --- |
| 变异体编号 | 杀死变异体所需要的测试用例数目 |
| Mutant\_1 | 86.65 |
| Mutant\_2 | 334.35 |
| Mutant\_3 | 509.50 |
| Mutant\_4 | 502.45 |
| Mutant\_5 | 742.30 |
| Mutant\_6 | 1064.20 |

由表5-21可以知道本实验中可以杀死的变异体均需要较多的测试用例才能杀死。因此不需要进行筛选本实验的变异体为：Mutant\_1、Mutant\_2、Mutant\_3、Mutant\_4、Mutant\_5、Mutant\_6。

#### 5.3.4.5 bash试验参数设定

本实验的DRT、RDRT参数与grep、gzip、flex实验的相同。但是MDRT实验的参数不一致，参数信息如表5-22。

表5-22 bash实验MDRT参数列表

|  |  |
| --- | --- |
| 参数 | 取值 |
|  | 0.00001 |
|  | 0.0001 |
|  | 0.001 |
|  | 0.01 |
|  | 0.1 |
|  | 0.2 |
|  | 0.25 |
|  | 0.3 |
|  | 0.35 |
|  | 0.4 |
|  | 0.45 |
|  | 0.5 |
|  | 0.55 |
|  | 0.6 |
|  | 0.65 |
|  | 0.7 |
| punishment | 50 |

由于本实验各个变异体相对难杀死，因此将惩罚因子的上限设置为50。

#### 5.3.4.6 bash试验初始概率设定

将bash实验的测试用例按照前面阐述的分区规则分区之后，在测试之前并不知道哪些分区具有较高的检测能力，怎么设置分区对应的初始概率成为一个不容易解决的难题。一方面，一般来说测试用例多的分区，由于测试用例的基数大，更有可能发现软件中的缺陷；另一方面，测试用例少的分区也存在发现软件中缺陷的可能。针对以上两个方面，本文提出了两种设定初始概率的方式：一，按照每个分区的测试用例个数占所有测试用例的百分比作为该分区对应的初始概率，DRT、RDRT如（17）(18)，MDRT的初始状态转移矩阵每一行的状态转移概率和DRT的相同；二，由于含有较少测试用例的分区也有可能具有较高的检测能力，因此将初始概率设定为均等分布，如（19）、（20）。





### 5.3.5 make实验细节

#### 5.3.5.1 make简介

在Linux（Unix）环境下使用GNU 的make工具能够比较容易的构建一个属于自己的工程，整个工程的编译只需要一个命令就可以完成编译、连接以至于最后的执行。不过投入一些时间去完成一个或者多个称之为Makefile文件的编写。Makefile文件描述了整个工程的编译、连接等规则。其中包括：工程中的哪些源文件需要编译以及如何编译、需要创建那些库文件以及如何创建这些库文件、如何最后产生想要得可执行文件。一旦提供一个（也可能会是多个）正确的Makefile。编译整个工程所要做的唯一的一件事就是在shell 提示符下输入make命令。整个工程完全自动编译，极大提高了效率。它的执行格式如（21）。



该工具在按照既定的规则进行编译时，不同的功能的实现是根据选择不同的可选项实现的，基本功能如表5-23所示。

表5-23 make工具功能列表

|  |  |
| --- | --- |
| 具体可选项 | 具体功能描述 |
| -f | 指定文件为makefile |
| -i | 忽略出错信息 |
| -e | 设置环境变量的优先权高于makefile中的 |
| -I | 指定搜索目录 |
| -k | 遇到错误继续执行 |
| -n | 将原来执行的命令输出 |
| -s | 沉默模式在执行前不输出相应得命令信息 |
| -t | 更新目标文件 |
| -w | 给出提示信息在开始之前以及结束之后 |
| -p | 输出所有宏定义以及目标信息 |
| -q | 根据目标文件是否更新返回信息 |
| -d | 进入调试模式 |

#### 5.3.5.2 make测试用例检测

TSL是一个测试规格说明语言，用来生成基于规格说明书的测试用例。由TSL规格说明可以形成一定数目的测试帧，然后再由生成的测试帧得到可以执行的测试用例。

本文选make源代码中的第二个版本进行试验，在实验之前测试用例的生成与检查是应该首先要做的事情。SIR网站提供的项目源代码中附带了不同版本的各自的测试用例脚本，这些测试用例是由TSL规格说明生成的。用该网站提供的测试用例解析工具mts对测试用例脚本进行解析后得到可以执行的测试用例。

#### 5.3.5.3 make实验分区介绍

对所有的测试用例进行分析，发现绝大部分的测试用例集中检测4个方面：一，指定其它文件为makefile文件，即可选项-f；二，在指定其它文件为makefile文件的基础上设置系统环境变量的优先权高于makefile文件中的，即，-ef；三，在指定其它文件为makefile文件的基础上进行调试，即，-df；四，指定别的文件为makefile文件并且设置系统环境变量的优先权高于makefile文件中的基础上进行调试，即，-def。

将上面的四类可以分为四个分区，另外不属于这四类的测试用例归为一个分区之中，这种分区方式为本实验的分区方案一，如表5-24。有上面所述，第一、二方面可以归为一类，都是在指定makefile文件的基础上进行编译。第三、四方面可以归为一类，它们都是进入调试模式。另外将不属于这四个方面的测试用例归为一类，得到本实验的第二种分区方式，如表5-25。

表5-24 make实验分区方案一

|  |  |
| --- | --- |
| 分区号 | 功能介绍 |
| 1 | 指定其它文件为makefile文件 |
| 2 | 指定其它文件为makefile文件，且系统环境变量的优先权高于makefile文件中的 |
| 3 | 进入调试模式，并且指定其它文件为makefile文件 |
| 4 | 进入调试模式，并且指定其它文件为makefil文件，系统环境变量的优先权高于makefile文件中 |
| 5 | 其它功能 |

表5-25 make实验分区方案二

|  |  |
| --- | --- |
| 分区号 | 功能介绍 |
| 1 | 指定其它文件为makefile文件，并进行编译 |
| 2 | 进入调试模式 |
| 3 | 其它功能 |

#### 5.3.5.4 make变异体选择

在源程序中，一共有19个变异体，对于每一变异体用随机测试的方法进行测试，重复20遍取均值结果如表5-26所示。

表5-26 make实验杀死变异体信息

|  |  |
| --- | --- |
| 变异体编号 | 杀死变异体所需要的测试用例数目 |
| Mutant\_1 | 杀不死 |
| Mutant\_2 | 杀不死 |
| Mutant\_3 | 杀不死 |
| Mutant\_4 | 杀不死 |
| Mutant\_5 | 杀不死 |
| Mutant\_6 | 杀不死 |
| Mutant\_7 | 杀不死 |
| Mutant\_8 | 杀不死 |
| Mutant\_9 | 杀不死 |
| Mutant\_10 | 1.05 |
| Mutant\_11 | 杀不死 |
| Mutant\_12 | 杀不死 |
| Mutant\_13 | 161.70 |
| Mutant\_14 | 杀不死 |
| Mutant\_15 | 207.80 |
| Mutant\_16 | 1.60 |
| Mutant\_17 | 杀不死 |
| Mutant\_18 | 杀不死 |
| Mutant\_19 | 杀不死 |

从表5-26可以看出本实验中的19个变异体只有4个开始杀死，由于Mutant\_16与Mutant\_10很容易被杀死，因此选择Mutant\_13与Mutant\_15作为实验的对象。

#### 5.3.5.5 make试验参数设定

本实验的DRT、RDRT参数与grep、gzip、flex、bash实验的相同。但是MDRT实验的参数不一致，参数信息如表5-27。

表5-27 make实验MDRT参数列表

|  |  |
| --- | --- |
| 参数 | 取值 |
|  | 0.00001 |
|  | 0.0001 |
|  | 0.001 |
|  | 0.01 |
|  | 0.1 |
|  | 0.2 |
|  | 0.25 |
|  | 0.3 |
|  | 0.35 |
|  | 0.4 |
|  | 0.45 |
|  | 0.5 |
|  | 0.55 |
|  | 0.6 |
|  | 0.65 |
|  | 0.7 |
| punishment | 10 |

#### 5.3.5.6 make试验初始概率设定

将bash实验的测试用例按照前面阐述的分区规则分区之后，在测试之前并不知道哪些分区具有较高的检测能力，怎么设置分区对应的初始概率成为一个不容易解决的难题。一方面，一般来说测试用例多的分区，由于测试用例的基数大，更有可能发现软件中的缺陷；另一方面，测试用例少的分区也存在发现软件中缺陷的可能。针对以上两个方面，本文提出了两种设定初始概率的方式：一，按照每个分区的测试用例个数占所有测试用例的百分比作为该分区对应的初始概率，DRT、RDRT如（22）(23)，MDRT的初始状态转移矩阵每一行的状态转移概率和DRT的相同；二，由于含有较少测试用例的分区也有可能具有较高的检测能力，因此将初始概率设定为均等分布，DRT、RDRT如（24）(25)，MDRT的初始状态转移矩阵每一行的状态转移概率和DRT的相同。



## 5.4 实验结果

根据上一章节介绍的实验方案进行试验，编写shell脚本控制实验的进行，Java程序控制不同的算法进行试验。

然后对于每一个实验的可以杀死变异体，随机从测试用例集中选取测试用例，得到所有变异体被杀死所需要的测试用例。如若变异体被杀死所需要的测试用例数目比较少，不能体现出不同算法在检测能力，因此本实验挑选相对难杀的变异体。

选取变异体之后，设置一系列各个算法的参数、不同方案的分区、每一个分区方案的均等初始概率和不均等初始概率以及设置每一个实验的不同数目的测试用例个数，比较不同算法杀死变异体的能力。

在实验过程中，本实验对每一次测试都重复20次，设置的随机数种子为1-20。除了RT算法之外，本实验的算法都是基于分区测试的。基于分区的算法选取测试用例一般分为两个步骤：一，选取分区；二，选取对应分区中测试用例。本实验为了避免在对比算法的测试效率随机数的影响，不是选完分区之后，在该分区之中随机选择测试用例，而是读取同一个试验下RT算法产生的随机序列，直到序列对应的测试用例在选择的分区之中，最后再执行测试用例。不同情况下的各种组合实验结果在下面的章节中展示。

### 5.4.1 算法参数对实验的影响

在实验的过程中，参数对算法的影响属于内部因素，因此如果该算法在合适的参数下，不管外部条件如何，都有较好的表现。本节对前面介绍的5个实验结果进行分析得到一个启发式规则：MDRT算法的参数范围在[0.35-0.5]之间有较高的检测效率，DRT、RDRT参数在[0.001-0.1]之间具有较好的检测能力。并且如果初始概率为均等分布参数设置可以靠近0.001，如果初始概率不为均等概率参数设置可以0.1。

#### 5.4.1.1 flex实验参数研究

在上面的章节中已经介绍本实验的实验方案。参数对于算法本身属于内部因素，初始概率分布以及分区的多少属于外部因素。对于MDRT、DRT、RDRT三种算法的17个参数不同情况下的测试结果如表5-28、表5-29、表5-30。在表中DRT以及RDRT根据初始概率分布的不同，参数取值也不同，如果是均等的初始概率分布参数范围，如果是不均等分布取值范围为。表5-28、表5-29、表5-30中展示不同度量标准下各个参数的均值，以及MDRT、DRT、RDRT算法在各自较好参数区间下最优值。

表5-28 MDRT参数对实验的影响

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| flex实验 | 度量标准 | 所有参数均值 | 所有参数方差均值 | 参数在[0.35,0.5] | 参数在[0.35,0.5]内方差 |
| 3分区均等初始概率 | F-Measure | 7.61 | 6.51 | 6.45 | 4.45 |
| NF-Measure | 7.47 | 8.96 | 5.45 | 6.45 |
| T-Measure | 28.40 | 20.10 | 24.65 | 14.90 |
| 3分区不均等初始概率 | F-Measure | 5.80 | 5.73 | 4.95 | 5.12 |
| NF-Measure | 6.17 | 7.40 | 4.75 | 4.73 |
| T-Measure | 27.52 | 20.26 | 24 | 14.43 |
| 6分区均等初始概率 | F-Measure | 5.18 | 4.57 | 4.3 | 3.47 |
| NF-Measure | 6.49 | 7.89 | 5.15 | 4.97 |
| T-Measure | 26.93 | 23.04 | 25.3 | 22.25 |
| 6分区不均等初始概率 | F-Measure | 4.11 | 3.69 | 3.2 | 2.9427 |
| NF-Measure | 4.41 | 5.55 | 3.15 | 2.9542 |
| T-Measure | 24.99 | 20.77 | 19.65 | 12.6343 |

从表5-28可以看出，本实验在探究MDRT不同参数取值对实验结果的影响时，参数不同实验结果又很大的差别，但是当参数的范围在[0.35,0.5]之间时具有较高的检测效率，并且比较稳定。

表5-29 DRT参数对实验的影响

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| flex实验 | 度量标准 | 所有参数均值 | 所有参数方差均值 | 参数在D内均值 | 参数在D内方差 |
| 3分区均等初始概率 | F-Measure | 7.44 | 6.72 | 5.75 | 5.77 |
| NF-Measure | 6.94 | 8.41 | 5.15 | 7.24 |
| T-Measure | 29.63 | 20.21 | 26.75 | 17.25 |
| 3分区不均等初始概率 | F-Measure | 6.47 | 6.30 | 4.15 | 3.21 |
| NF-Measure | 5.65 | 6.90 | 2.85 | 3.30 |
| T-Measure | 27.97 | 18.99 | 26.55 | 25.31 |
| 6分区均等初始概率 | F-Measure | 5.28 | 4.37 | 4.45 | 4.32 |
| NF-Measure | 6.11 | 7.79 | 6.00 | 8.82 |
| T-Measure | 25.64 | 21.03 | 23.95 | 18.79 |
| 6分区不均等初始概率 | F-Measure | 4.38 | 4.11 | 3.9 | 3.16 |
| NF-Measure | 4.56 | 5.68 | 2.1 | 1.89 |
| T-Measure | 26.68 | 23.84 | 22.3 | 19.63 |

从表5-29可以看出，本实验在探究DRT不同参数对实验结果的影响时，参数不同实验的结果有很大的差别。但是当参数的范围在[0.001-0.1]之间时具有较高的检测效率，并且比较稳定。

表5-30 RDRT参数对实验的影响

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| flex实验 | 度量标准 | 所有参数均值 | 所有参数方差均值 | 参数在D内均值 | 参数在D内方差 |
| 3分区均等初始概率 | F-Measure | 7.98 | 6.77 | 6.2 | 5.11 |
| NF-Measure | 6.54 | 8.17 | 5.5 | 5.88 |
| T-Measure | 28.81 | 18.48 | 25.65 | 14.52 |
| 3分区不均等初始概率 | F-Measure | 6.10 | 5.59 | 3.55 | 2.15 |
| NF-Measure | 6.87 | 8.86 | 4.75 | 7.11 |
| T-Measure | 30.66 | 23.03 | 22.7 | 18.85 |
| 6分区均等初始概率 | F-Measure | 5.05 | 4.18 | 3.75 | 2.50 |
| NF-Measure | 5.57 | 7.83 | 5.6 | 9.74 |
| T-Measure | 25.45 | 23.22 | 19.2 | 15.92 |
| 6分区不均等初始概率 | F-Measure | 4.34 | 3.87 | 2.95 | 2.13 |
| NF-Measure | 4.87 | 6.45 | 2.30 | 2.86 |
| T-Measure | 24.83 | 20.83 | 22.7 | 12.08 |

从表5-30可以明显看出参数在[0.001-0.1]范围内具有较好的检测能力。

#### 5.4.1.2 make实验参数研究

该实验由于只有两个变异体，因此该实验的度量标准没有NF-Measure。对于MDRT、DRT、RDRT三种算法的17个参数不同情况下的测试结果如表5-31、表5-32、表5-33。

表5-31 MDRT参数对实验的影响

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| make实验 | 度量标准 | 所有参数均值 | 所有参数方差均值 | 参数在[0.35-0.5]内 | 参数在[0.35-0.5]内方差 |
| 3分区均等初始概率 | F-Measure | 7.98 | 6.77 | 8.1 | 8.88 |
| T-Measure | 28.81 | 18.48 | 21.3 | 16.80 |
| 3分区不均等初始概率 | F-Measure | 30.57 | 26.42 | 16.2 | 13.33 |
| T-Measure | 86.99 | 64.41 | 37.8 | 26.04 |
| 5分区均等初始概率 | F-Measure | 11.30 | 13.91 | 10.00 | 12.06 |
| T-Measure | 33.88 | 27.76 | 28.45 | 20.27 |
| 5分区不均等初始概率 | F-Measure | 51.92 | 44.19 | 35.60 | 29.38 |
| T-Measure | 154.00 | 120.67 | 93.95 | 33.64 |

表5-32 DRT参数对实验的影响

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| make实验 | 度量标准 | 所有参数均值 | 所有参数方差均值 | 参数在D内均值 | 参数在D内方差 |
| 3分区均等初始概率 | F-Measure | 8.38 | 7.71 | 7.65 | 7.06 |
| T-Measure | 21.11 | 15.66 | 20.75 | 15.87 |
| 3分区不均等初始概率 | F-Measure | 23.41 | 17.07 | 13.55 | 8.98 |
| T-Measure | 54.82 | 37.11 | 27.8 | 17.07 |
| 5分区均等初始概率 | F-Measure | 13.76 | 11.98 | 13.95 | 12.11 |
| T-Measure | 33.99 | 25.83 | 33.35 | 23.82 |
| 5分区不均等初始概率 | F-Measure | 28.41 | 24.09 | 14.15 | 10.90 |
| T-Measure | 70.95 | 46.02 | 38.55 | 21.56 |

表5-33 RDRT参数对实验的影响

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| make实验 | 度量标准 | 所有参数均值 | 所有参数方差均值 | 参数在D内均值 | 参数在D内方差 |
| 3分区均等初始概率 | F-Measure | 8.44 | 7.58 | 6.65 | 6.93 |
| T-Measure | 19.02 | 14.70 | 16.50 | 11.18 |
| 3分区不均等初始概率 | F-Measure | 12.52 | 8.11 | 11.60 | 7.69 |
| T-Measure | 22.77 | 14.71 | 22.00 | 14.52 |
| 5分区均等初始概率 | F-Measure | 13.17 | 11.34 | 11.75 | 9.89 |
| T-Measure | 31.70 | 24.33 | 26.60 | 18.92 |
| 5分区不均等初始概率 | F-Measure | 19.64 | 13.61 | 15.65 | 13.30 |
| T-Measure | 38.00 | 24.73 | 33.90 | 24.66 |

从上面的数据中可以看出，MDRT参数在[0.35,0.5]之间时具有较为明显的检测能力。DRT、RDRT实验当初始概率为均等分布时，参数在[0.001-0.01]之间具有较明显的检测能力。并且在实验中发现当初始概率分布不是均匀分布时，参数在靠近0.1时具有明显的检测能力，当初始概率为均匀分布时参数靠近0.001时具有较高的检测效率。

#### 5.4.1.3 gzip实验参数研究

在上面的章节中已经介绍本实验的实验方案。参数对于算法本身属于内部因素，初始概率分布以及分区的多少属于外部因素。对于MDRT、DRT、RDRT三种算法的17个参数不同情况下的测试结果如表5-34、表5-35、表5-36。在表中DRT以及RDRT根据初始概率分布的不同，参数取值也不同，如果是均等的初始概率分布参数范围，如果是不均等分布取值范围为。

表5-34 MDRT参数对实验的影响

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| gzip实验 | 度量标准 | 所有参数均值 | 所有参数方差均值 | 参数在[0.35,0.5]内 | 参数在[0.35,0.5]内方差 |
| 3分区均等初始概率 | F-Measure | 38.79 | 41.4 | 33.8 | 42.05 |
| NF-Measure | 117.07 | 143.81 | 103.5 | 89.94 |
| T-Measure | 422.61 | 285.98 | 442.7 | 280.21 |
| 3分区不均等初始概率 | F-Measure | 42.53 | 43.52 | 36.25 | 40.81 |
| NF-Measure | 137.74 | 184.73 | 113.8 | 146.85 |
| T-Measure | 438.78 | 311.93 | 360.55 | 236.89 |
| 4分区均等初始概率 | F-Measure | 23.95 | 21.19 | 18.5 | 14.29 |
| NF-Measure | 39.89 | 36.41 | 26.1 | 28.61 |
| T-Measure | 163.73 | 108.18 | 131.7 | 88.47 |
| 4分区不均等初始概率 | F-Measure | 28.00 | 26.02 | 20.5 | 18.10 |
| NF-Measure | 44.17 | 45.61 | 28.45 | 22.37 |
| T-Measure | 167.87 | 116.47 | 117.2 | 60.24 |

表5-35 DRT参数对实验的影响

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| gzip实验 | 度量标准 | 所有参数均值 | 所有参数方差均值 | 参数在D内 | 参数在D内方差 |
| 3分区均等初始概率 | F-Measure | 42.30 | 44.06 | 28.70 | 27.35 |
| NF-Measure | 166.27 | 186.36 | 109.10 | 179.97 |
| T-Measure | 496．00 | 378.84 | 390.35 | 339.83 |
| 3分区不均等初始概率 | F-Measure | 42.69 | 43.88 | 40.95 | 34.60 |
| NF-Measure | 148.32 | 163.57 | 83.15 | 67.25 |
| T-Measure | 530.06 | 382.15 | 408.30 | 362.21 |
| 4分区均等初始概率 | F-Measure | 19.10 | 16.70 | 11.15 | 8.35 |
| NF-Measure | 33.35 | 36.27 | 38.60 | 32.05 |
| T-Measure | 135.5 | 100.15 | 106.35 | 59.38 |
| 4分区不均等初始概率 | F-Measure | 20.44 | 18.31 | 16.50 | 14.26 |
| NF-Measure | 37.56 | 41.95 | 34．00 | 24.27 |
| T-Measure | 142.36 | 99.48 | 115.55 | 66.41 |

表5-36 RDRT参数对实验的影响

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| gzip实验 | 度量标准 | 所有参数均值 | 所有参数方差均值 | 参数在D内均值 | 参数在D内方差 |
| 3分区均等初始概率 | F-Measure | 38.79 | 41.44 | 36.00 | 39.87 |
| NF-Measure | 117.07 | 143.81 | 78.55 | 89.04 |
| T-Measure | 370.55 | 218.38 | 293.95 | 196.20 |
| 3分区不均等初始概率 | F-Measure | 39.78 | 38.96 | 36.30 | 29.83 |
| NF-Measure | 146.32 | 163.98 | 107.60 | 100.21 |
| T-Measure | 483.01 | 369.19 | 332.45 | 221.87 |
| 4分区均等初始概率 | F-Measure | 19.46 | 16.83 | 22.15 | 16.31 |
| NF-Measure | 26.47 | 24.15 | 23.65 | 20.39 |
| T-Measure | 115.50 | 80.32 | 121.70 | 83.61 |
| 4分区不均等初始概率 | F-Measure | 20.19 | 17.01 | 18.95 | 14.47 |
| NF-Measure | 24.76 | 22.34 | 23.20 | 15.68 |
| T-Measure | 105.63 | 76.84 | 89.60 | 65.16 |

从上面的数据中可以看出，MDRT参数在[0.35,0.5]与所有参数的均值比较都比较小，也就是说该算法的参数在这个区间里时，具有更高的检测效率。DRT、RDRT实验当初始概率为均等分布时，参数在[0.001-0.01]之间具有较明显的检测能力。当初始概率分布不是均匀分布时，参数在靠近0.1时具有明显的检测能力。

#### 5.4.1.4 grep实验参数研究

在上面的章节中已经介绍本实验的实验方案。参数对于算法本身属于内部因素，初始概率分布以及分区的多少属于外部因素。对于MDRT、DRT、RDRT三种算法的17个参数不同情况下的测试结果如表5-37、表5-38、表5-39。在表中DRT以及RDRT根据初始概率分布的不同，参数取值也不同，如果是均等的初始概率分布参数范围，如果是不均等分布取值范围为。

表5-37 MDRT参数对实验的影响

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| grep实验 | 度量标准 | 所有参数均值 | 所有参数方差均值 | 参数在[0.35,0.5]内 | 参数在[0.35,0.5]内方差 |
| 4分区均等初始概率 | F-Measure | 61.80 | 49.59 | 58.24 | 44.35 |
| NF-Measure | 92.62 | 88.14 | 90.23 | 81.99 |
| T-Measure | 1948.22 | 1553.53 | 1913.14 | 1399.51 |
| 4分区不均等初始概率 | F-Measure | 90.07 | 75.65 | 80.60 | 69.41 |
| NF-Measure | 137.10 | 116.93 | 117.94 | 109.10 |
| T-Measure | 1529.76 | 1042.70 | 1515.78 | 1103.30 |
| 6分区均等初始概率 | F-Measure | 63.10 | 55.24 | 52.00 | 49.24 |
| NF-Measure | 84.62 | 78.30 | 60.45 | 104.20 |
| T-Measure | 2247.26 | 1495.24 | 1883.75 | 1488.37 |
| 6分区不均等初始概率 | F-Measure | 92.30 | 81.89 | 94.50 | 85.70 |
| NF-Measure | 126.32 | 113.32 | 105.20 | 105.81 |
| T-Measure | 1578.99 | 1025.51 | 1490.54 | 832.58 |

表5-38 DRT参数对实验的影响

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| grep实验 | 度量标准 | 所有参数均值 | 所有参数方差均值 | 参数在D内 | 参数在D内方差 |
| 4分区均等初始概率 | F-Measure | 65.45 | 49.55 | 62.30 | 57.38 |
| NF-Measure | 97.96 | 100.39 | 102.40 | 91.08 |
| T-Measure | 2068.76 | 1805.55 | 2700.13 | 2575.51 |
| 4分区不均等初始概率 | F-Measure | 76.24 | 61.93 | 61.38 | 53.22 |
| NF-Measure | 94.32 | 84.38 | 74.53 | 59.24 |
| T-Measure | 1901.81 | 1391.04 | 2085.95 | 1316.56 |
| 6分区均等初始概率 | F-Measure | 55.43 | 45.83 | 53.20 | 40.45 |
| NF-Measure | 90.90 | 78.67 | 65.03 | 51.80 |
| T-Measure | 2065.31 | 1442.19 | 1985.58 | 1241.75 |
| 6分区不均等初始概率 | F-Measure | 80.48 | 65.55 | 60.93 | 50.48 |
| NF-Measure | 100.44 | 93.87 | 97.40 | 82.46 |
| T-Measure | 2131.69 | 1471.25 | 2555.18 | 1644.44 |

表5-39 RDRT参数对实验的影响

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| grep实验 | 度量标准 | 所有参数均值 | 所有参数方差均值 | 参数在D内均值 | 参数在D内方差 |
| 4分区均等初始概率 | F-Measure | 65.48 | 53.56 | 64.53 | 63.89 |
| NF-Measure | 89.40 | 89.64 | 78.63 | 78.72 |
| T-Measure | 2063.52 | 1616.68 | 2209.80 | 1772.87 |
| 4分区不均等初始概率 | F-Measure | 73.41 | 54.46 | 74.40 | 50.38 |
| NF-Measure | 96.70 | 91.88 | 89.25 | 94.98 |
| T-Measure | 2129.36 | 1675.29 | 1552.53 | 929.00 |
| 6分区均等初始概率 | F-Measure | 56.15 | 48.67 | 52.00 | 44.61 |
| NF-Measure | 77.14 | 75.02 | 60.45 | 53.83 |
| T-Measure | 2190.65 | 1538.23 | 1883.75 | 1385.88 |
| 6分区不均等初始概率 | F-Measure | 67.34 | 55.53 | 76.55 | 73.19 |
| NF-Measure | 89.72 | 86.75 | 98.68 | 116.30 |
| T-Measure | 1911.93 | 1498.83 | 1760.85 | 1417.12 |

从上面的数据中可以看出，MDRT参数在[0.35,0.5]与所有参数的均值比较都比较小，也就是说该算法的参数在这个区间里时，具有更高的检测效率。DRT、RDRT实验当初始概率为均等分布时，参数在[0.001-0.01]之间具有较明显的检测能力。当初始概率分布不是均匀分布时，参数在靠近0.1时具有明显的检测能力

#### 5.4.1.5 bash实验参数研究

在上面的章节中已经介绍本实验的实验方案。参数对于算法本身属于内部因素，初始概率分布以及分区的多少属于外部因素。对于MDRT、DRT、RDRT三种算法的17个参数不同情况下的测试结果如表5-40、表5-41、表5-42。在表中DRT以及RDRT根据初始概率分布的不同，参数取值也不同，如果是均等的初始概率分布参数范围，如果是不均等分布取值范围为。

表5-40 MDRT参数对实验的影响

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| bash实验 | 度量标准 | 所有参数均值 | 所有参数方差均值 | 参数在[0.35,0.5]内 | 参数在[0.35,0.5]内方差 |
| 4分区均等初始概率 | F-Measure | 61.80 | 49.59 | 58.24 | 44.35 |
| NF-Measure | 92.62 | 88.14 | 90.23 | 81.99 |
| T-Measure | 1948.22 | 1553.53 | 1913.14 | 1399.51 |
| 4分区不均等初始概率 | F-Measure | 90.07 | 75.65 | 80.60 | 69.41 |
| NF-Measure | 137.10 | 116.93 | 117.94 | 109.10 |
| T-Measure | 1529.76 | 1042.70 | 1515.78 | 1103.30 |
| 6分区均等初始概率 | F-Measure | 63.10 | 55.24 | 59.31 | 57.70 |
| NF-Measure | 84.62 | 78.30 | 88.75 | 104.20 |
| T-Measure | 2247.26 | 1495.24 | 2241.35 | 1542.37 |
| 6分区不均等初始概率 | F-Measure | 92.30 | 81.89 | 94.50 | 85.70 |
| NF-Measure | 126.32 | 113.32 | 105.20 | 105.81 |
| T-Measure | 1578.99 | 1025.51 | 1490.54 | 832.58 |

表5-41 DRT参数对实验的影响

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| bash实验 | 度量标准 | 所有参数均值 | 所有参数方差均值 | 参数在D内 | 参数在D内方差 |
| 4分区均等初始概率 | F-Measure | 65.45 | 49.55 | 62.30 | 57.38 |
| NF-Measure | 97.96 | 100.39 | 102.40 | 91.08 |
| T-Measure | 2068.76 | 1805.55 | 2700.13 | 2575.51 |
| 4分区不均等初始概率 | F-Measure | 76.24 | 61.93 | 61.38 | 53.22 |
| NF-Measure | 94.32 | 84.38 | 74.53 | 59.24 |
| T-Measure | 1901.81 | 1391.04 | 2085.95 | 1316.56 |
| 6分区均等初始概率 | F-Measure | 55.43 | 45.83 | 53.20 | 40.45 |
| NF-Measure | 90.90 | 78.67 | 65.03 | 51.80 |
| T-Measure | 2065.31 | 1442.19 | 1985.58 | 1241.75 |
| 6分区不均等初始概率 | F-Measure | 80.48 | 65.55 | 60.93 | 50.48 |
| NF-Measure | 100.44 | 93.87 | 97.40 | 82.46 |
| T-Measure | 2131.69 | 1471.25 | 2555.18 | 1644.44 |

表5-42 RDRT参数对实验的影响

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| bash实验 | 度量标准 | 所有参数均值 | 所有参数方差均值 | 参数在D内均值 | 参数在D内方差 |
| 4分区均等初始概率 | F-Measure | 65.48 | 53.56 | 64.53 | 63.89 |
| NF-Measure | 89.40 | 89.64 | 78.63 | 78.72 |
| T-Measure | 2063.52 | 1616.68 | 2209.80 | 1772.87 |
| 4分区不均等初始概率 | F-Measure | 73.41 | 54.46 | 74.40 | 50.38 |
| NF-Measure | 96.70 | 91.88 | 89.25 | 94.98 |
| T-Measure | 2129.36 | 1675.29 | 1552.53 | 929.00 |
| 6分区均等初始概率 | F-Measure | 56.15 | 48.67 | 52.00 | 44.61 |
| NF-Measure | 77.14 | 75.02 | 60.45 | 53.83 |
| T-Measure | 2190.65 | 1538.23 | 1883.75 | 1385.88 |
| 6分区不均等初始概率 | F-Measure | 67.34 | 55.53 | 76.55 | 73.19 |
| NF-Measure | 89.72 | 86.75 | 98.68 | 116.30 |
| T-Measure | 1911.93 | 1498.83 | 1760.85 | 1417.12 |

从上面的数据中可以看出，MDRT参数在[0.35,0.5]与所有参数的均值比较都比较小，也就是说该算法的参数在这个区间里时，具有更高的检测效率。DRT、RDRT实验当初始概率为均等分布时，参数在[0.001-0.01]之间具有较明显的检测能力。当初始概率分布不是均匀分布时，参数在靠近0.1时具有明显的检测能力

### 5.4.2 初始概率分布对实验的影响

初始的概率分布采取了两种方式：均等初始概率分布；分区内的测试用例数占所有测试用例的百分比。 通过5个实验结果的研究，得到一条启发式规则：当存在一个失效率比较大的分区时，选择不均等初始概率分布。当所有的分区失效率很低时，不同的初始概率分布在检测第一个和第二个缺陷时差别不大，但是不均等的初始概率分布在检测所有的失效时具有更高的效率。

下面实验部分探究初始概率分布对实验的影响时，由于上面的阐述MDRT参数在[0.35-0.5]之间具有较好的测试效果，DRT、RDRT参数在D范围内具有较好的检测效率。所以下面这三个算法均是在较好的参数区间下取均值。

#### 5.4.2.1 flex实验初始概率研究

该实验的设计方案在上面的章节中已经展示，利用功能分区的思想，将所有的测试用例划分为3个分区和6个分区。分别对这两种方案设置两种初始概率分布：均等概率分布和不均等概率分布，实验结果如图5-2、5-3、5-4、5-5。

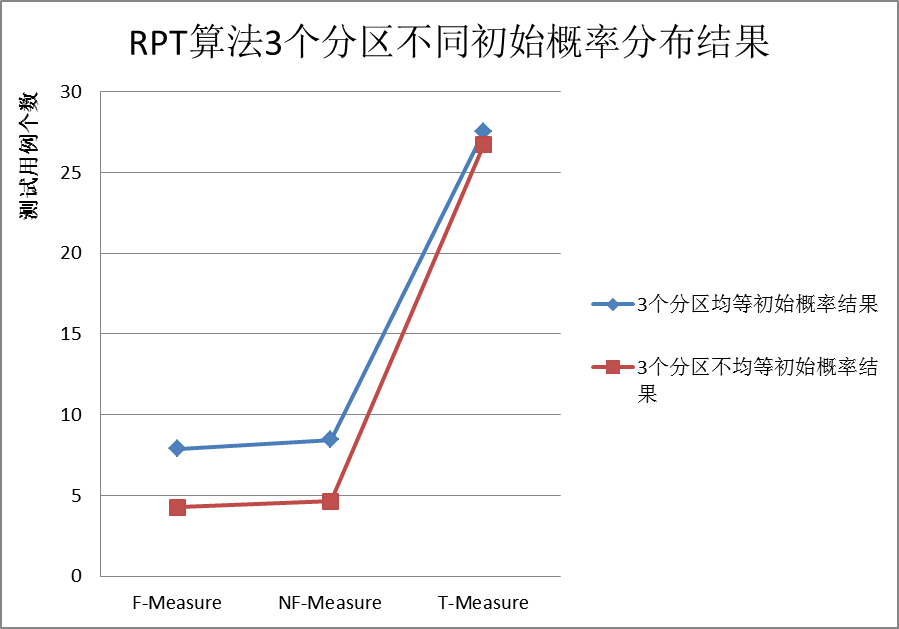


图5-2（a） RPT算法3个分区不同初始概率分布下的结果图

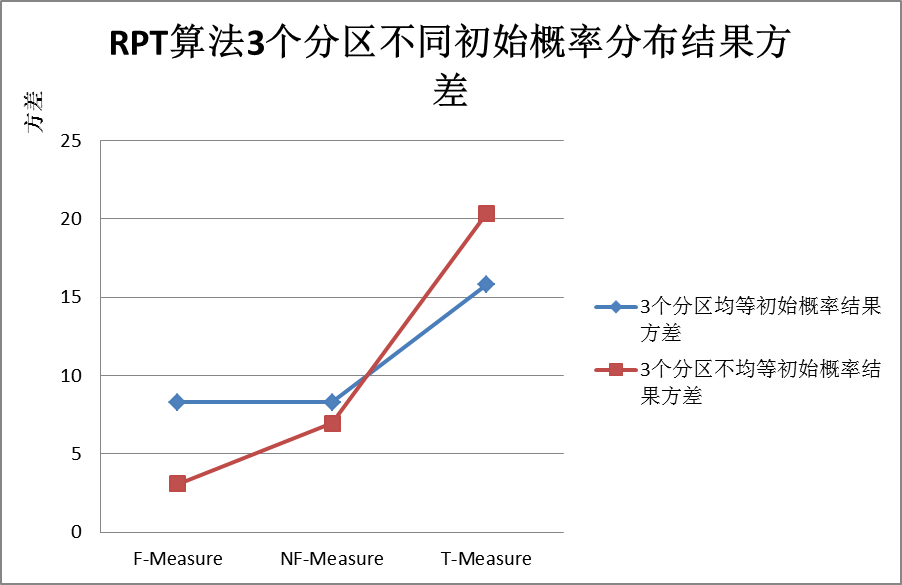


图5-2（b） RPT算法3个分区不同初始概率分布下的结果方差图

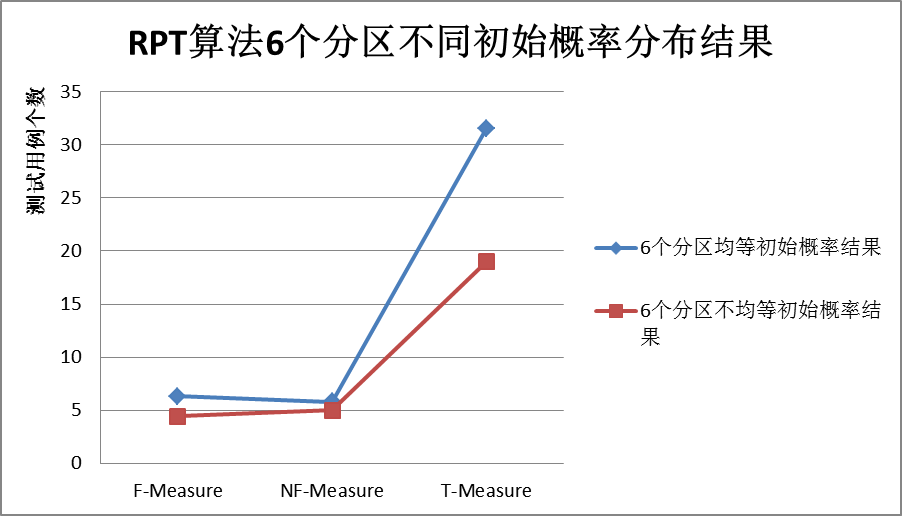


图5-2（c） RPT算法6个分区不同初始概率分布下的结果图

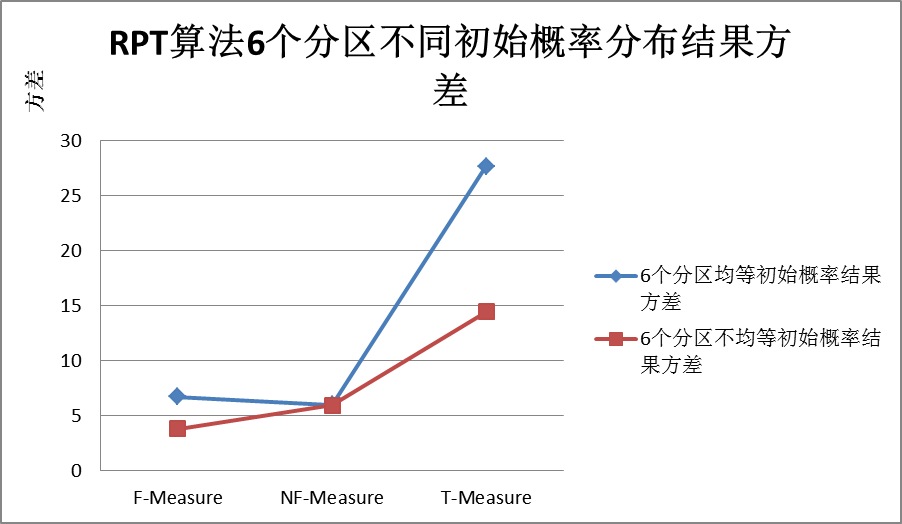


图5-2（d） RPT算法6个分区不同初始概率分布下的结果方差图

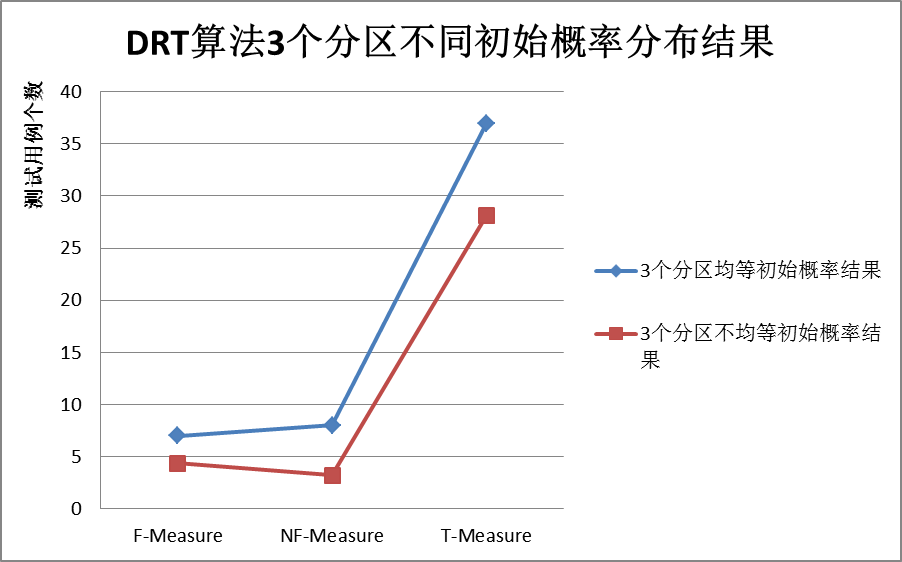


图5-3（a） DRT算法3个分区不同初始概率分布下的结果图

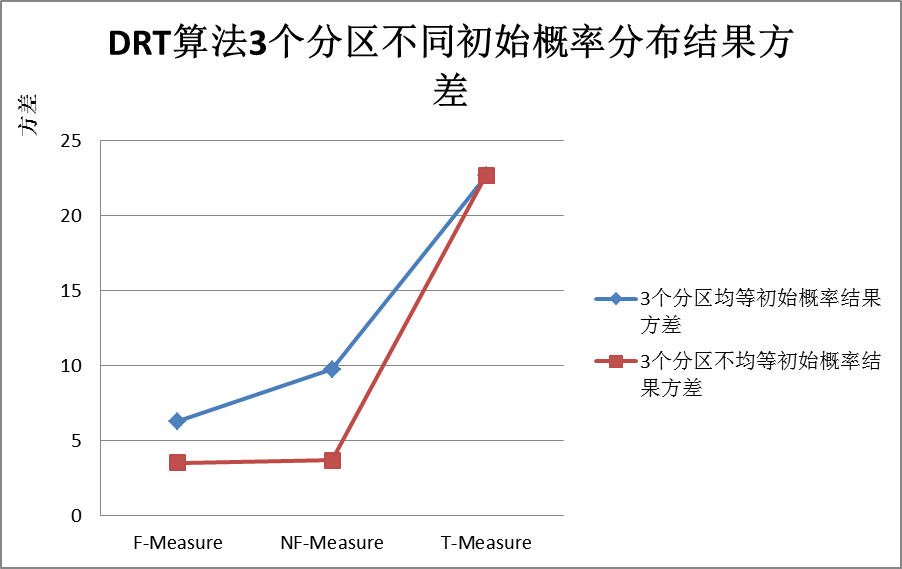


图5-3（b） DRT算法3个分区不同初始概率分布下的结果方差图

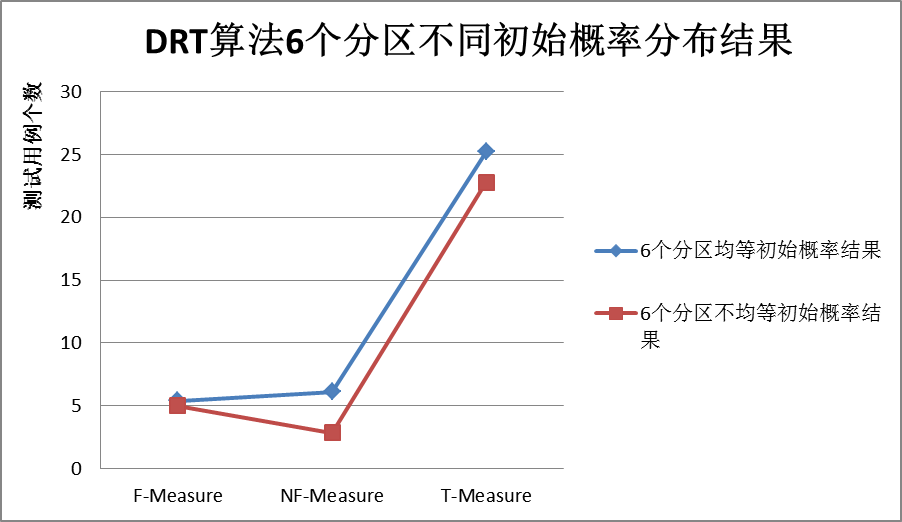


图5-3（c） DRT算法6个分区不同初始概率分布下的结果图

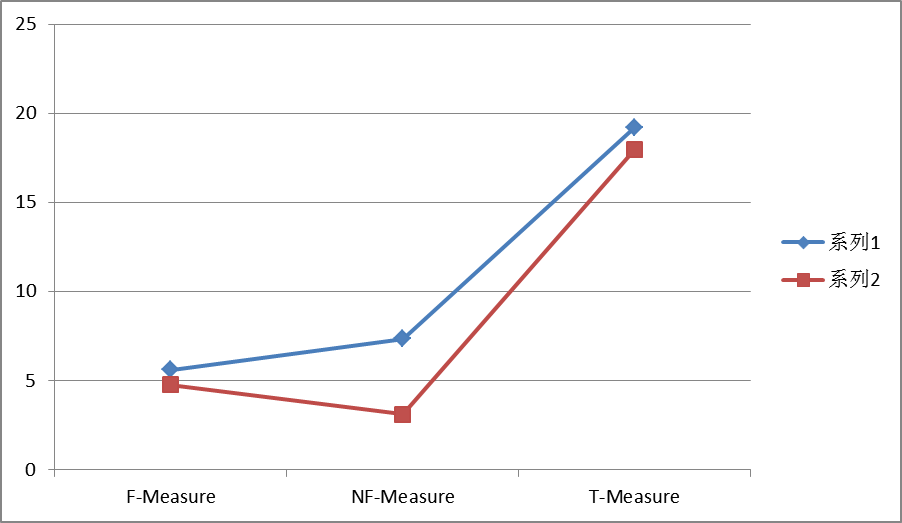


图5-3（d） DRT算法6个分区不同初始概率分布下的结果方差图

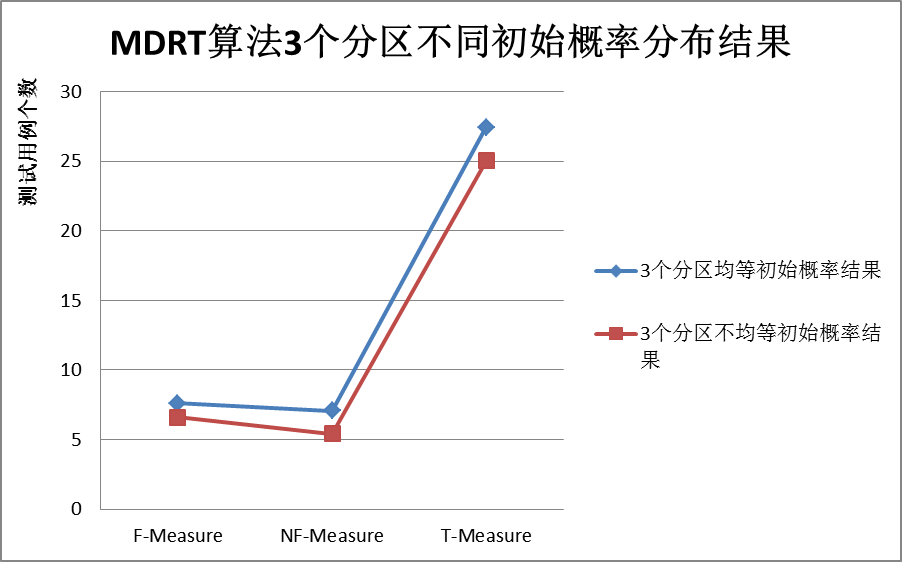


图5-4（a） MDRT算法3个分区不同初始概率分布下的结果图

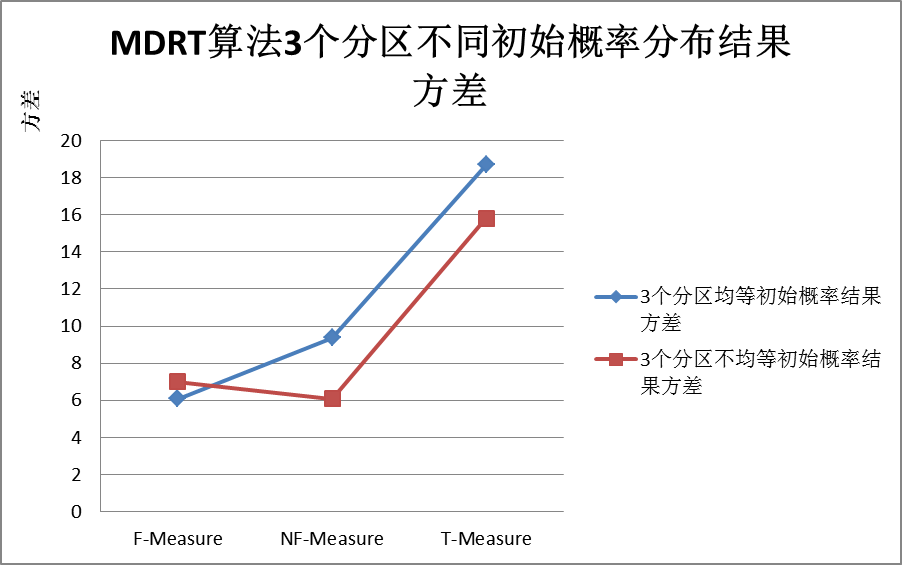


图5-4（b） MDRT算法3个分区不同初始概率分布下的结果方差图

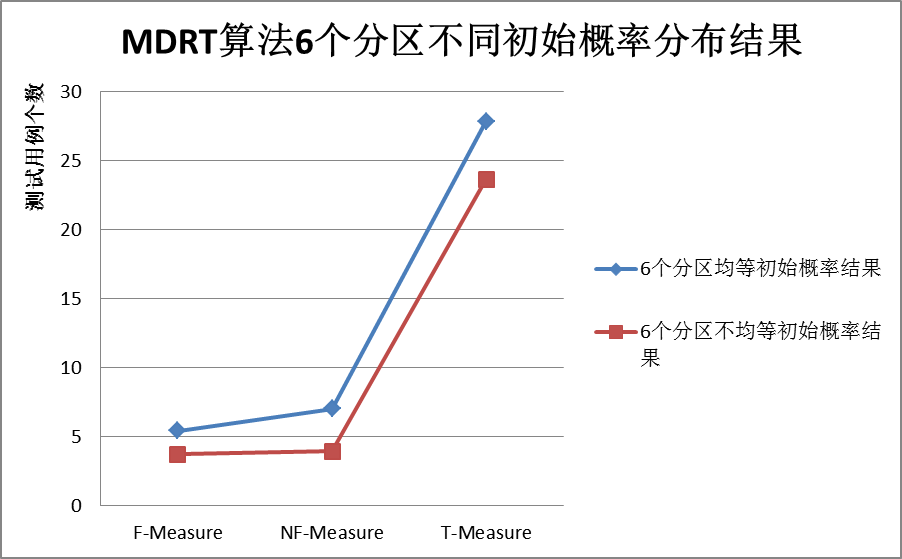


图5-4（c） MDRT算法6个分区不同初始概率分布下的结果图

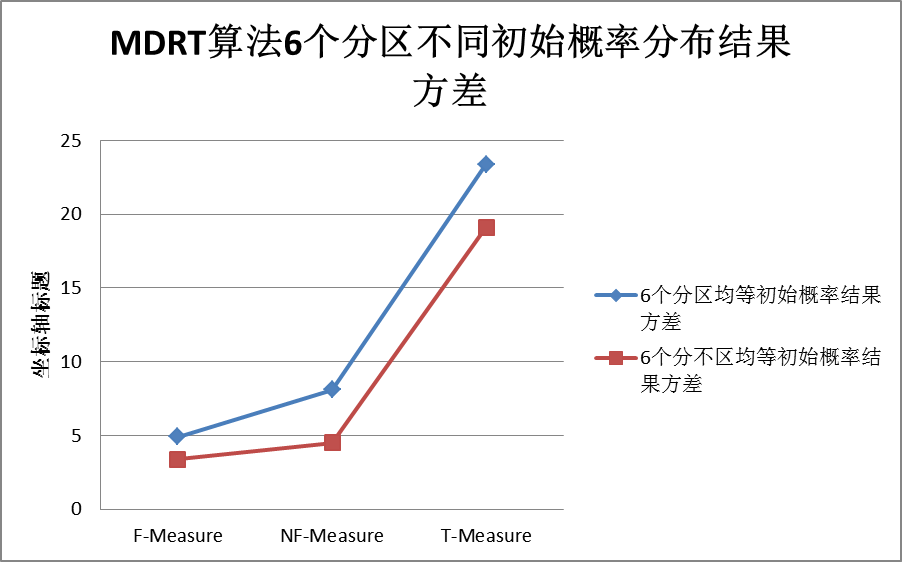


图5-4（d） MDRT算法6个分区不同初始概率分布下的结果方差图

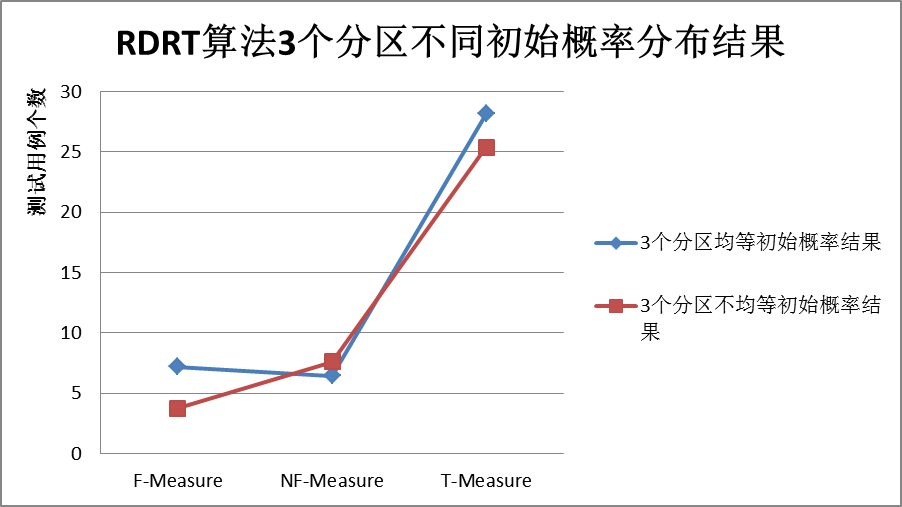


图5-5（a） RDRT算法3个分区不同初始概率分布下的结果图

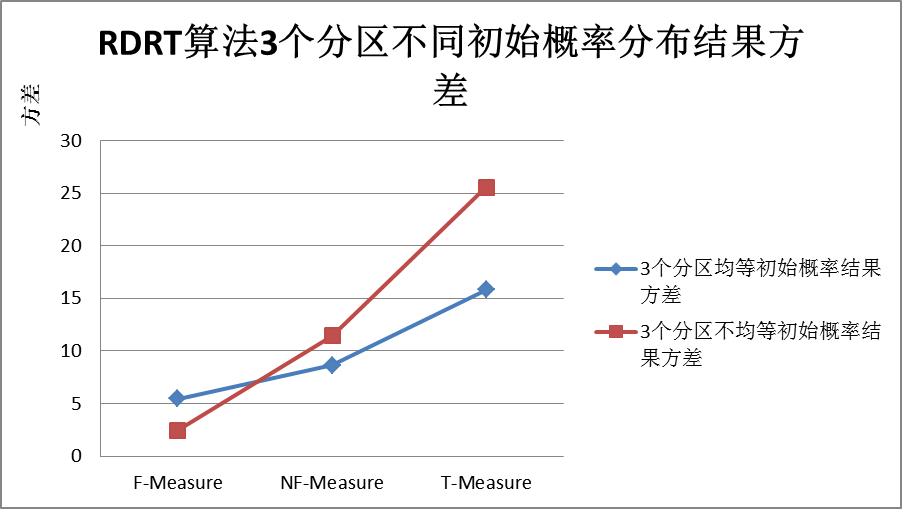


图5-5（b） RDRT算法3个分区不同初始概率分布下的结果方差图

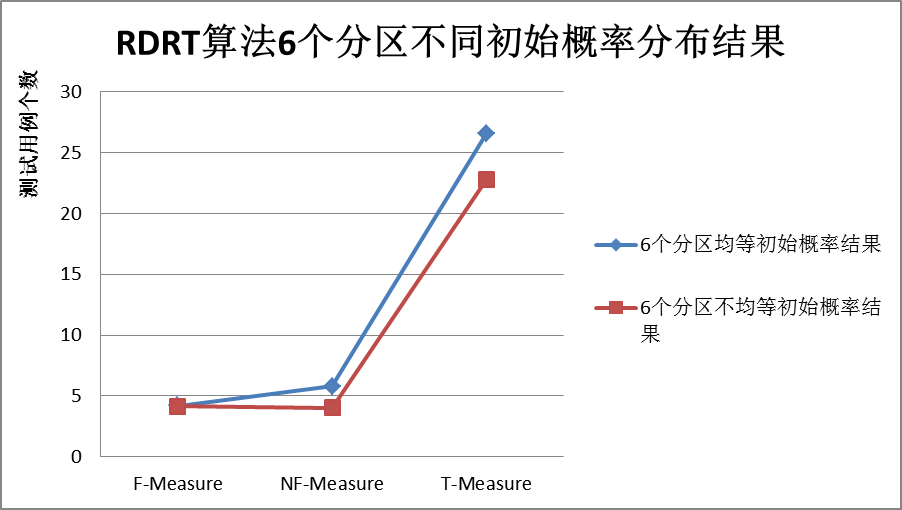


图5-5（c） RDRT算法6个分区不同初始概率分布下的结果图

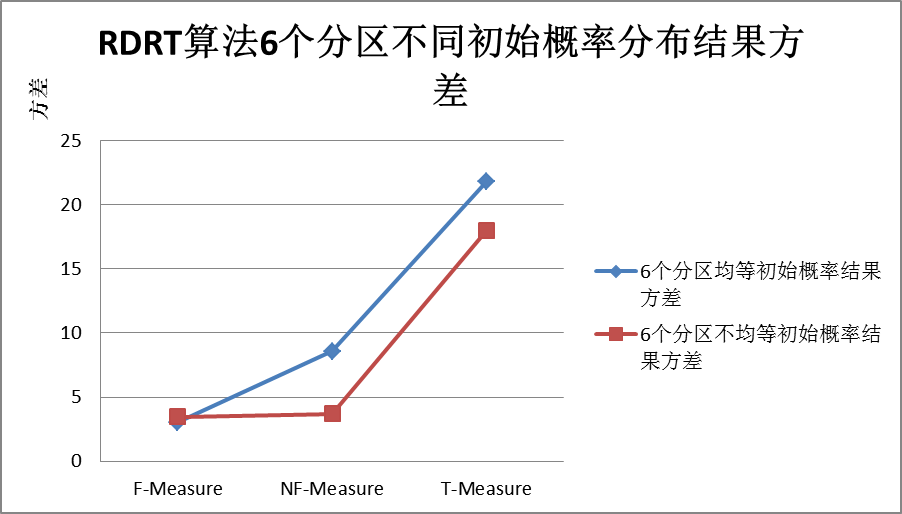


图5-5（d） RDRT算法6个分区不同初始概率分布下的结果方差图

从上图中可以明显看出该实验中基于分区思想的各个算法，采取不均等的初始概率分布，即，以分区内的测试用例占所有测试用例的百分比作为该分区的初始概率时，测试效率比均等分布的初始概率要高，并且比较稳定。

#### 5.4.2.2 make实验初始概率研究

该实验的设计方案在上面的章节中已经展示，利用功能分区的思想，将所有的测试用例划分为3个分区和6个分区。分别对这两种方案设置两种初始概率分布：均等概率分布和不均等概率分布，实验结果如图5-6、5-7、5-8、5-9。

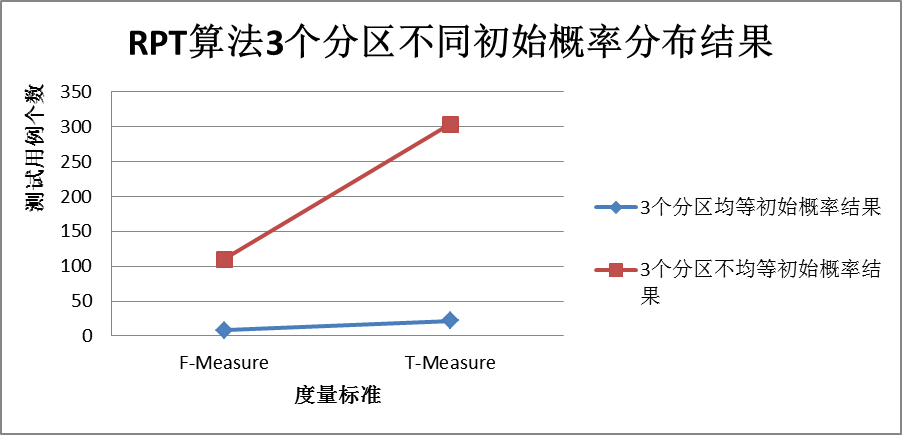


图5-6（a） RPT算法3个分区不同初始概率分布下的结果图

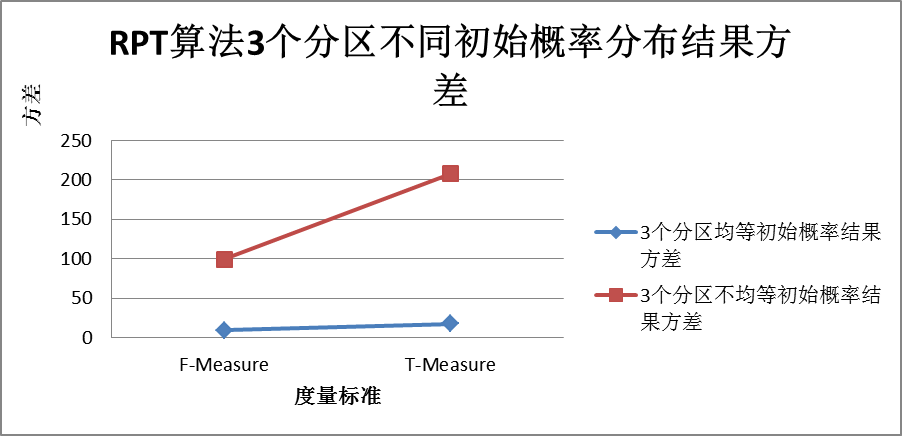


图5-6（b） RPT算法3个分区不同初始概率分布下的结果方差图

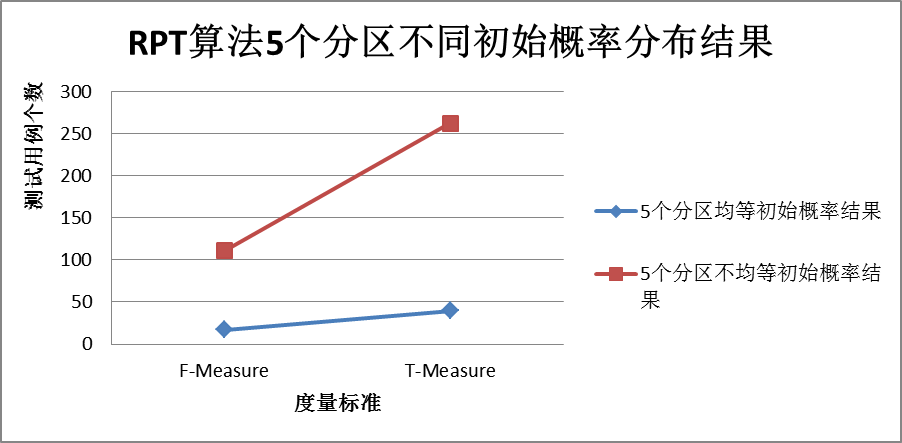


图5-6（c） RPT算法5个分区不同初始概率分布下的结果图

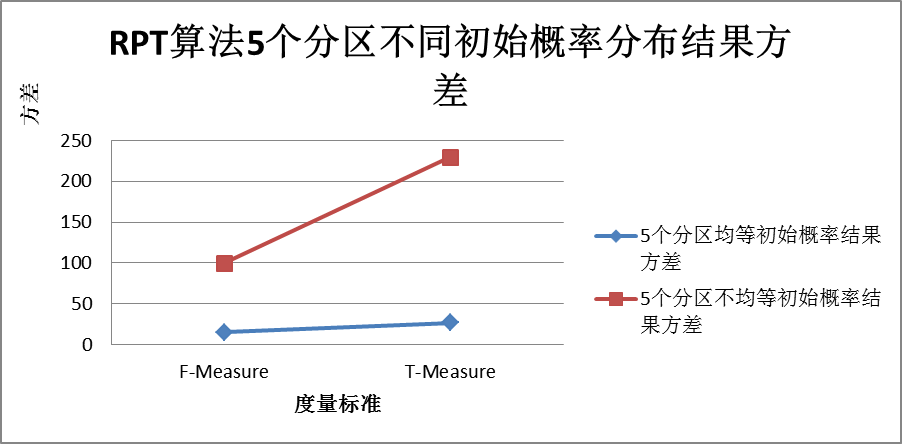


图5-6（d） RPT算法5个分区不同初始概率分布下的结果方差图

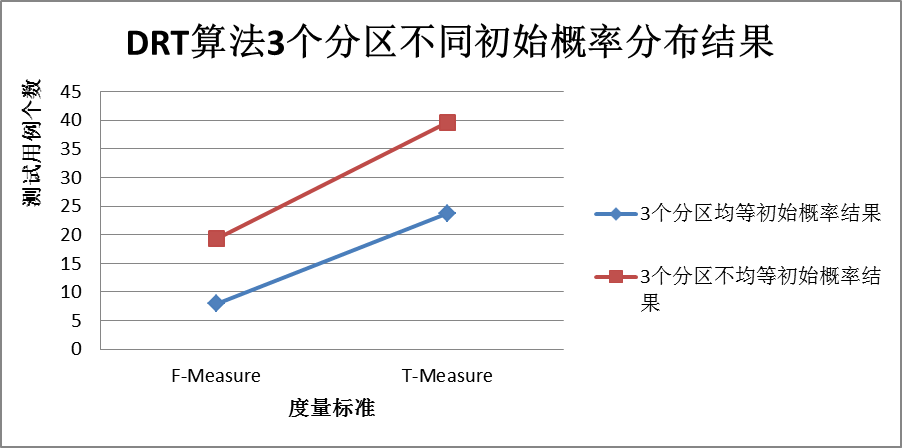


图5-7（a） DRT算法3个分区不同初始概率分布下的结果图

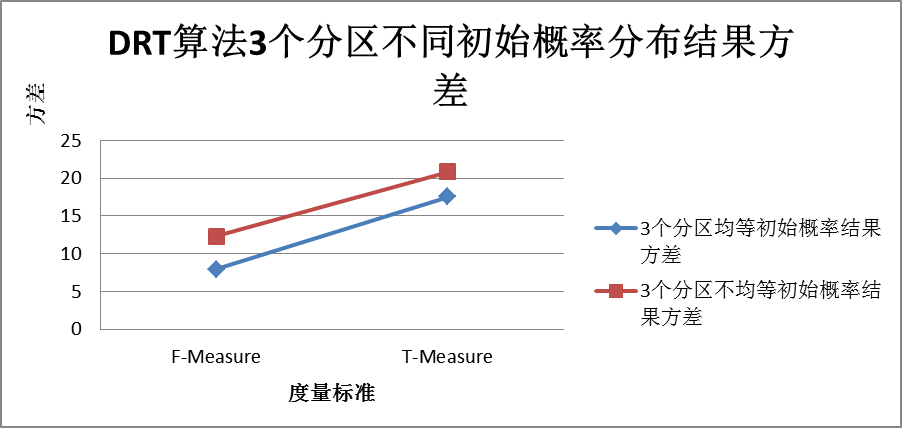


图5-7（b） DRT算法3个分区不同初始概率分布下的结果方差图

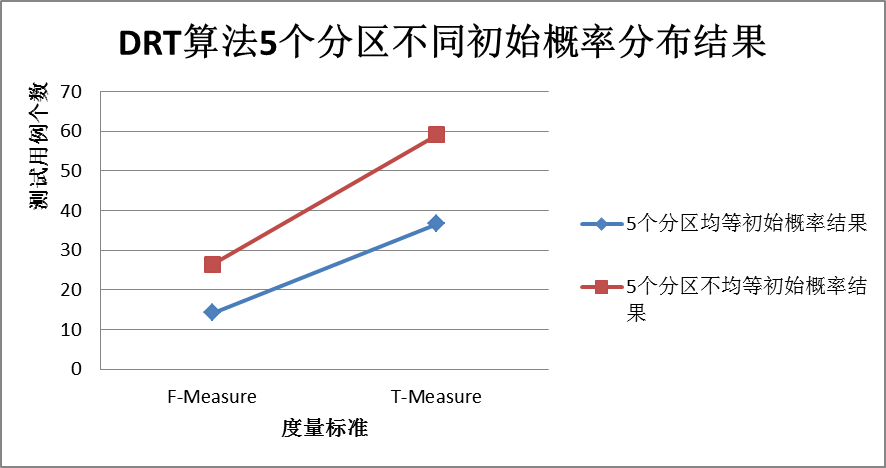


图5-7（c） DRT算法5个分区不同初始概率分布下的结果图

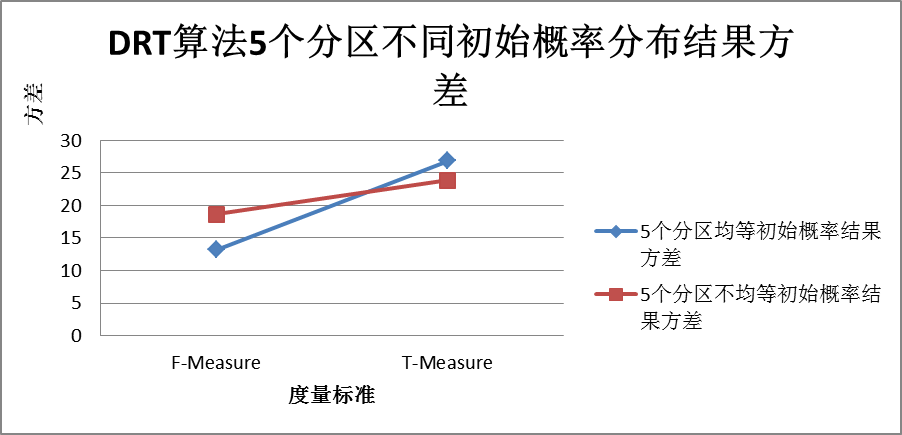


图5-7（d） DRT算法5个分区不同初始概率分布下的结果方差图

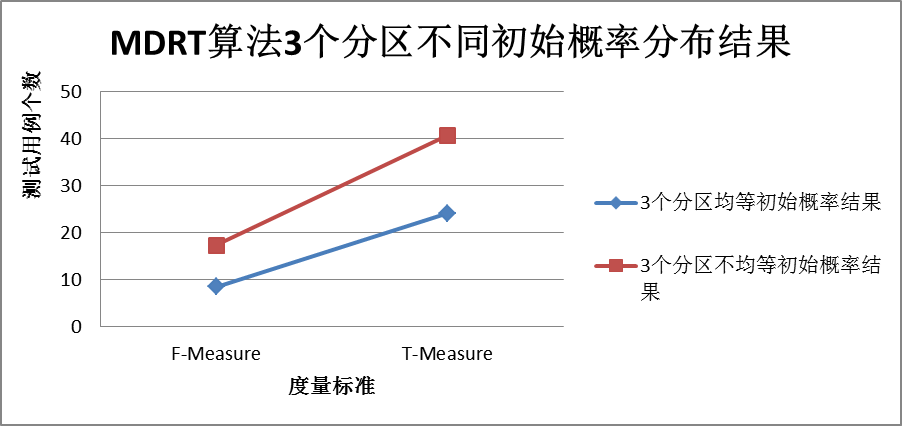


图5-8（a） MDRT算法3个分区不同初始概率分布下的结果图

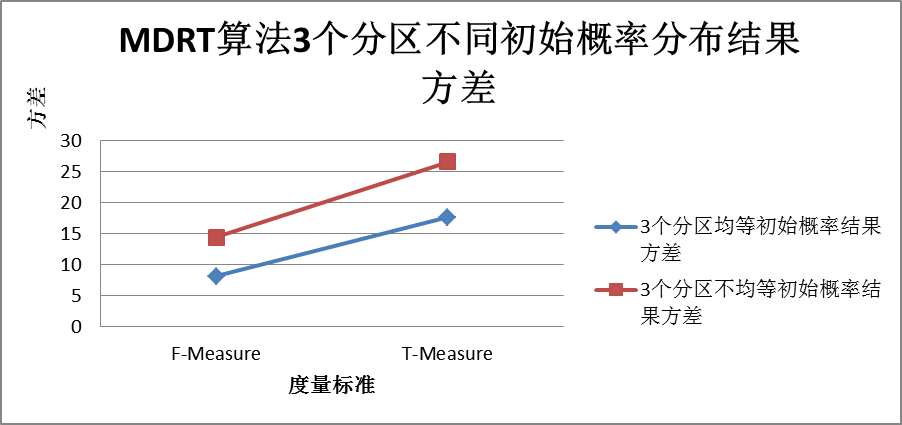


图5-8（b） MDRT算法3个分区不同初始概率分布下的结果方差图

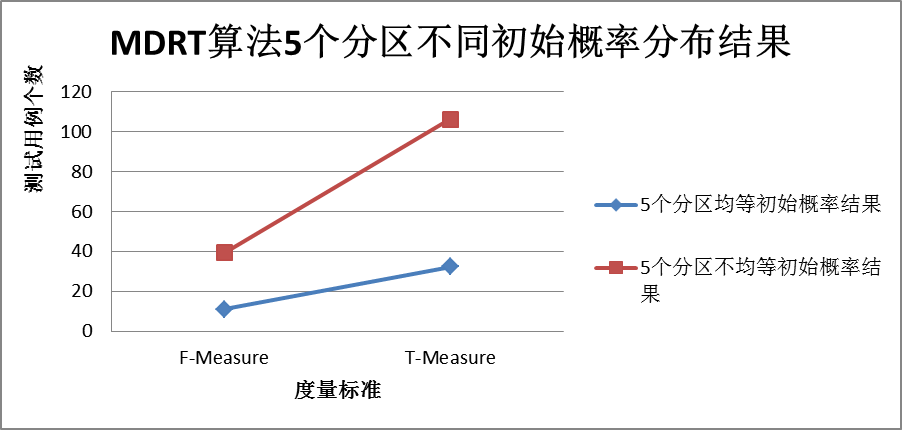


图5-8（c） MDRT算法5个分区不同初始概率分布下的结果图

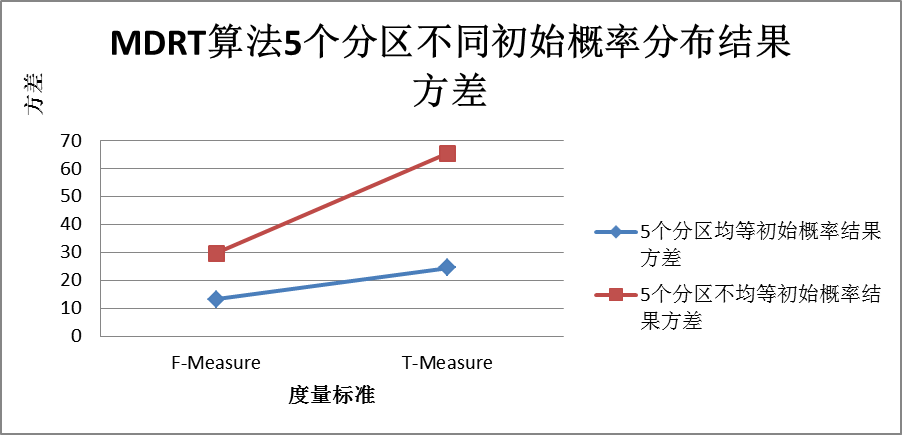


图5-8（d） MDRT算法5个分区不同初始概率分布下的结果方差图

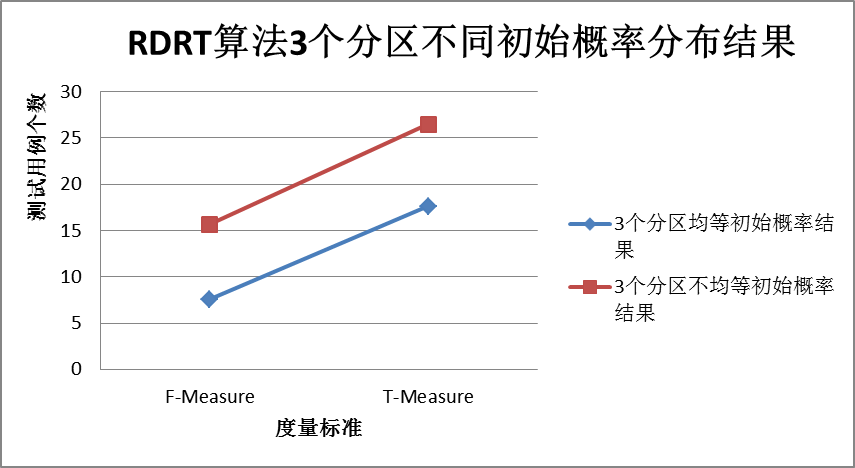


图5-9（a） RDRT算法3个分区不同初始概率分布下的结果图

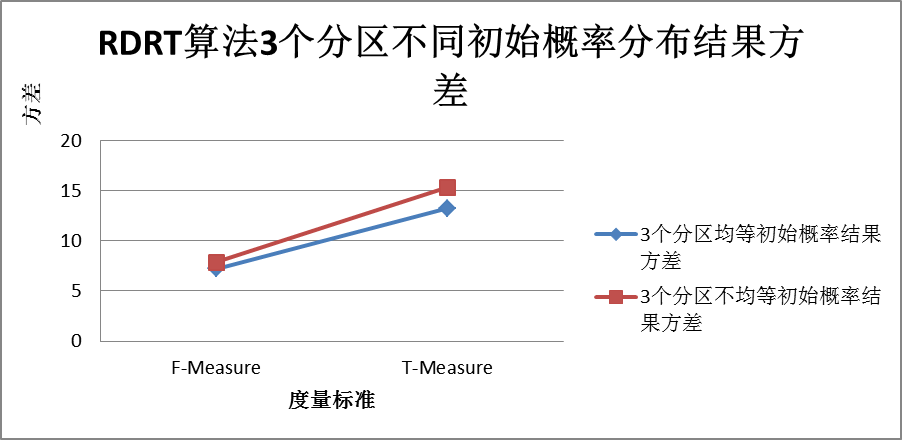


图5-9（b） RDRT算法3个分区不同初始概率分布下的结果方差图

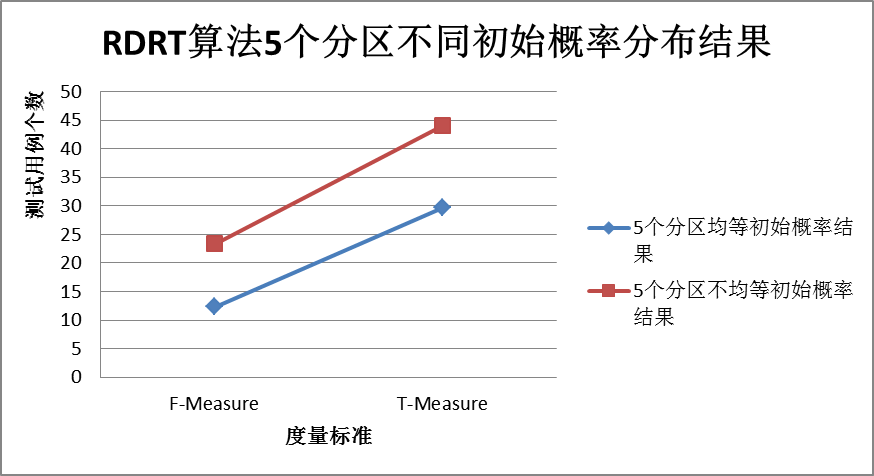


图5-9（c） RDRT算法5个分区不同初始概率分布下的结果图

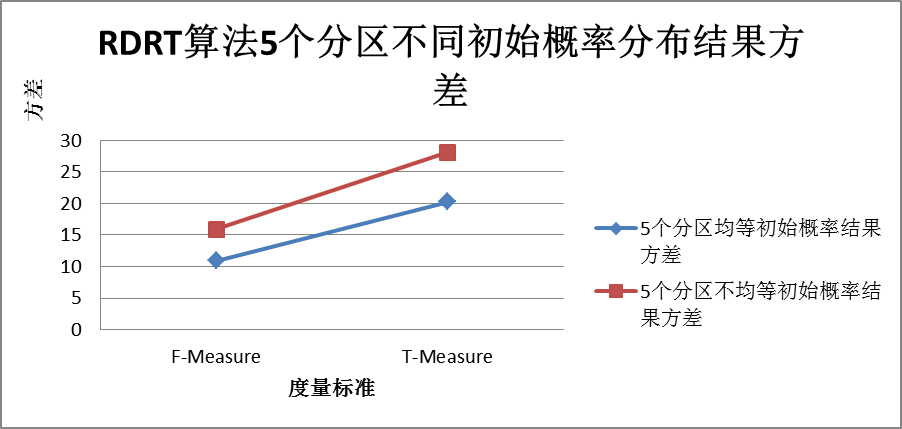


图5-9（d） RDRT算法5个分区不同初始概率分布下的结果方差图

通过上图的展示可以看出，以分区思想为基础的各个算法在该实验中，初始概率分布为均等分布时，算法的测试效率远远高于初始概率分布不为均等分布时的测试效率。经进一步研究发现，该实验能够杀死变异体的测试用例全部集中在一个测试用例数目比较少的分区内，因此如果将分区内的测试用例占所有测试用例的百分比作为初始概率分布，能够检测出失效的分区很难被选中，因此不均等的初始概率分布比均等的初始概率分布测试效率更高。

#### 5.4.2.3 gzip实验初始概率研究

该实验的设计方案在上面的章节中已经展示，利用功能分区的思想，将所有的测试用例划分为3个分区和6个分区。分别对这两种方案设置两种初始概率分布：均等概率分布和不均等概率分布，实验结果如图5-10、5-11、5-12、5-13。

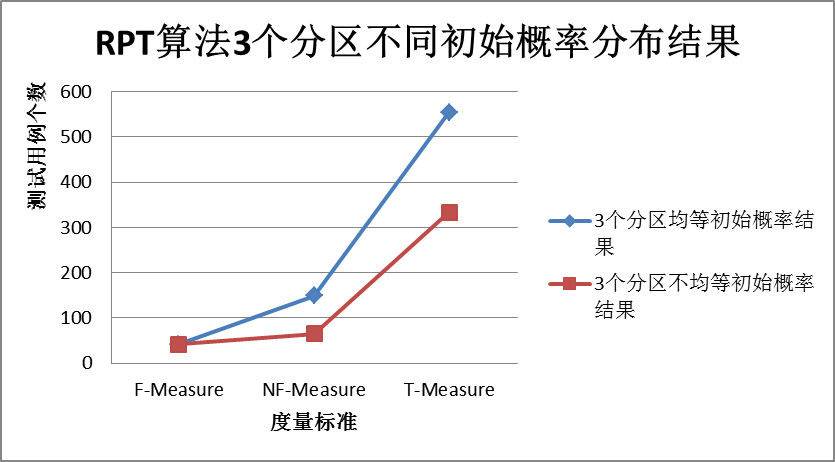


图5-10（a） RPT算法3个分区不同初始概率分布下的结果图

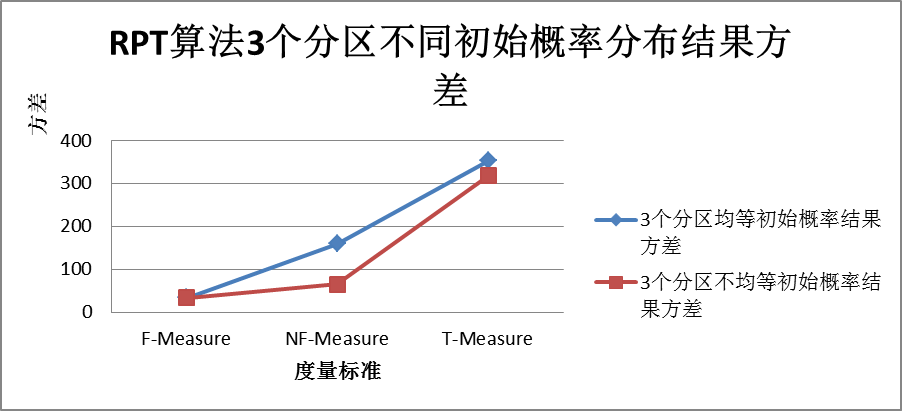


图5-10（b） RPT算法3个分区不同初始概率分布下的结果方差图

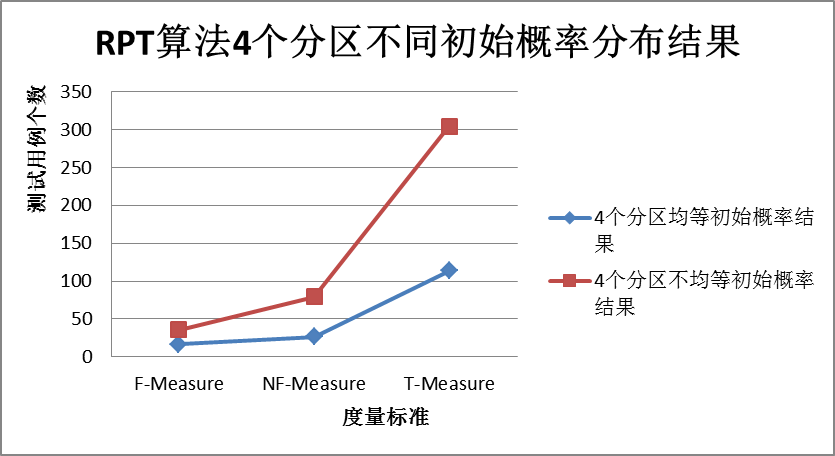


图5-10（c） RPT算法4个分区不同初始概率分布下的结果图

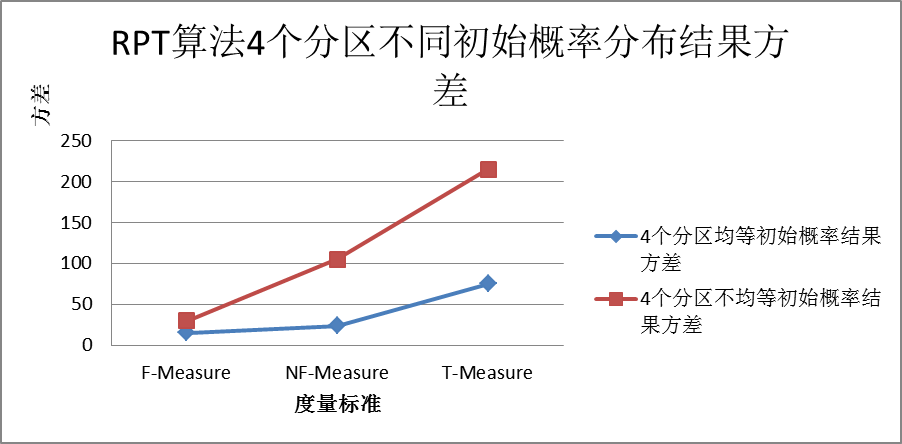


图5-10（d） RPT算法4个分区不同初始概率分布下的结果方差图

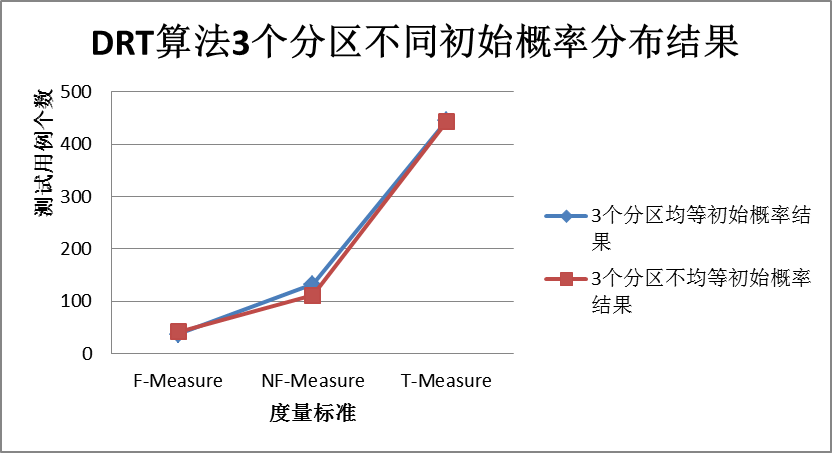


图5-11（a） DRT算法3个分区不同初始概率分布下的结果图

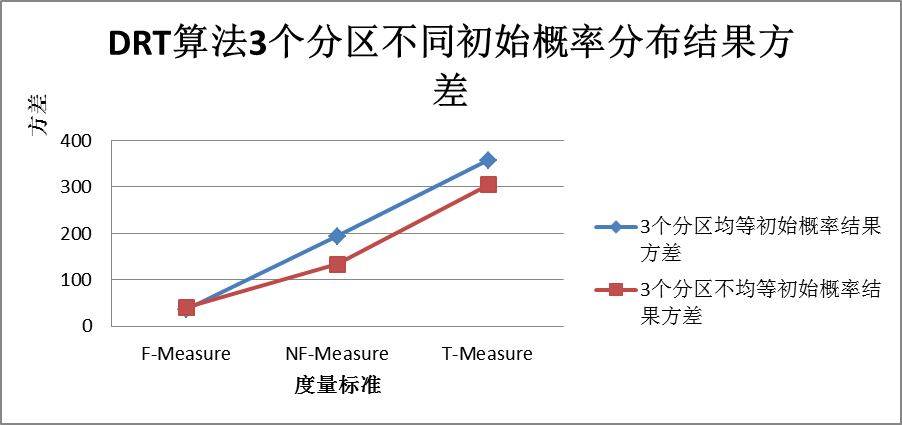


图5-11（b） DRT算法3个分区不同初始概率分布下的结果方差图



图5-11（c） DRT算法4个分区不同初始概率分布下的结果图

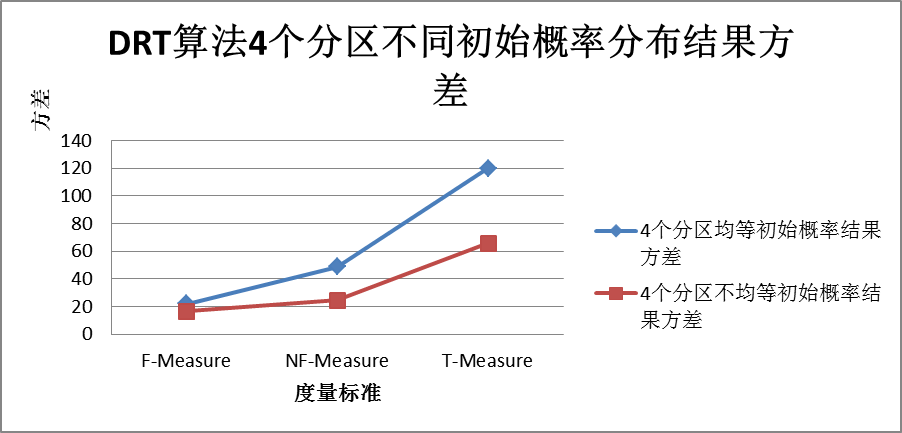


图5-11（d） DRT算法4个分区不同初始概率分布下的结果方差图

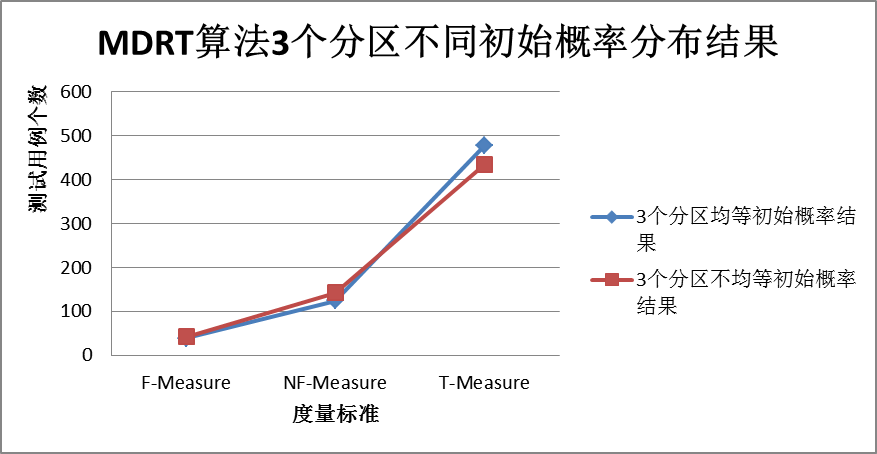


图5-12（a） MDRT算法3个分区不同初始概率分布下的结果图



图5-12（b） MDRT算法3个分区不同初始概率分布下的结果方差图

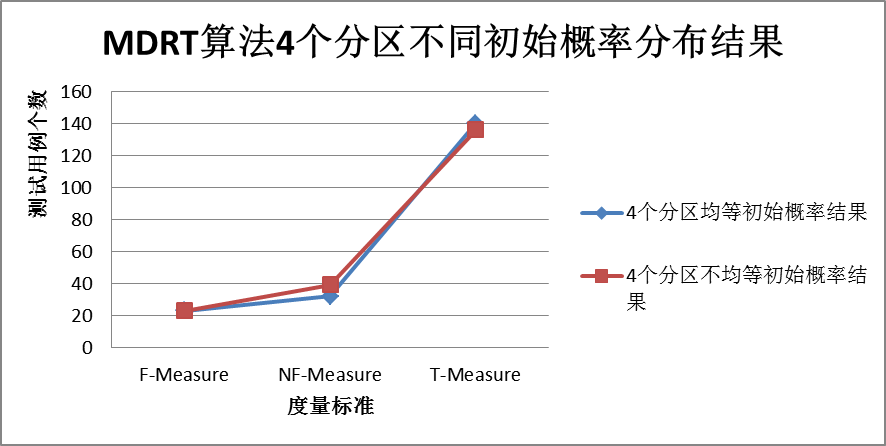


图5-12（c） MDRT算法4个分区不同初始概率分布下的结果图

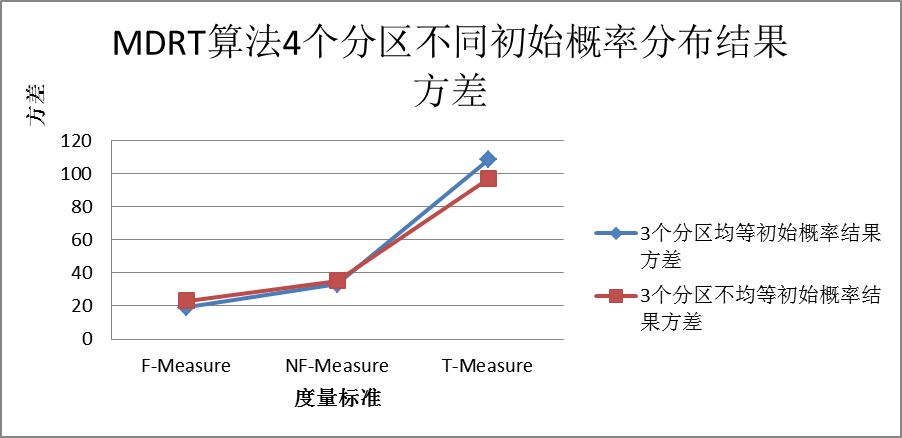


图5-12（d） MDRT算法4个分区不同初始概率分布下的结果方差图

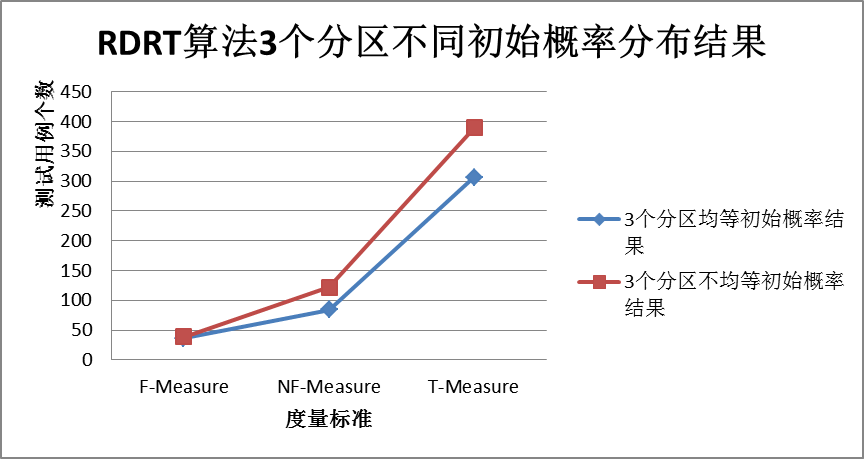


图5-13（a） RDRT算法3个分区不同初始概率分布下的结果图

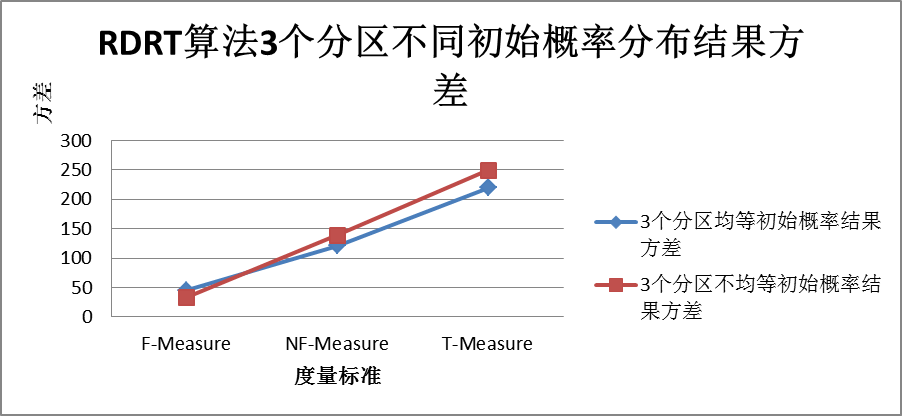


图5-13（b） RDRT算法3个分区不同初始概率分布下的结果方差图

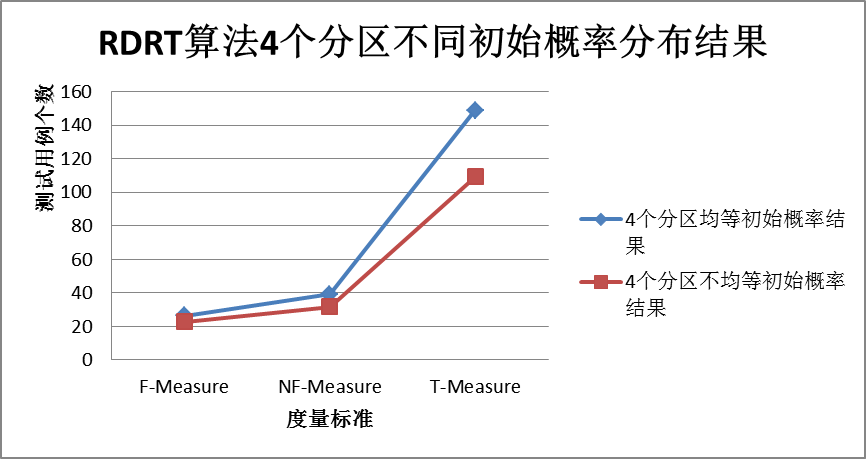


图5-13（c） RDRT算法4个分区不同初始概率分布下的结果图

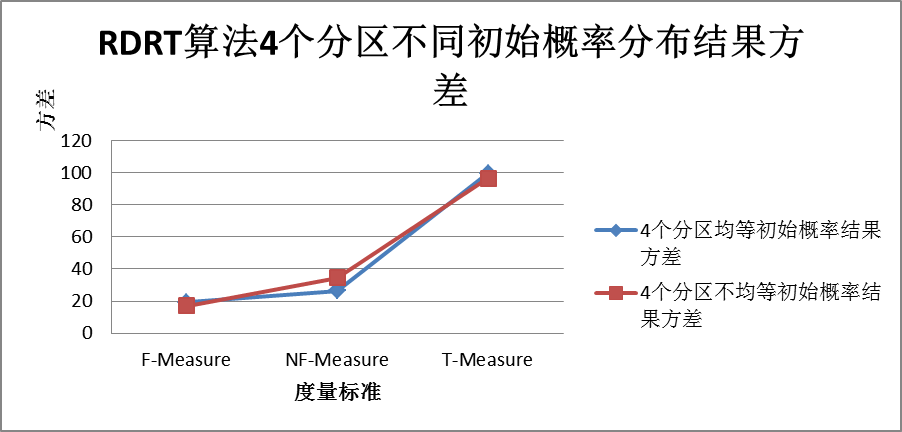


图5-13（d） RDRT算法4个分区不同初始概率分布下的结果方差图

#### 5.4.2.4 grep实验初始概率研究

该实验的设计方案在上面的章节中已经展示， 利用功能分区的思想，将所有的测试用例划分为4个分区和6个分区。分别对这两种方案设置两种初始概率分布：均等概率分布和不均等概率分布，实验结果如图5-14、5-15、5-16、5-17。由于该实验杀死所有的测试用例所需要的测试用例数目比较多，为了更清晰的表示出各个算法在不同的初始概率分布下的测试结果，本文将测试用例数目取以10为底的自然对数。



图5-14（a） RPT算法4个分区不同初始概率分布下的结果图

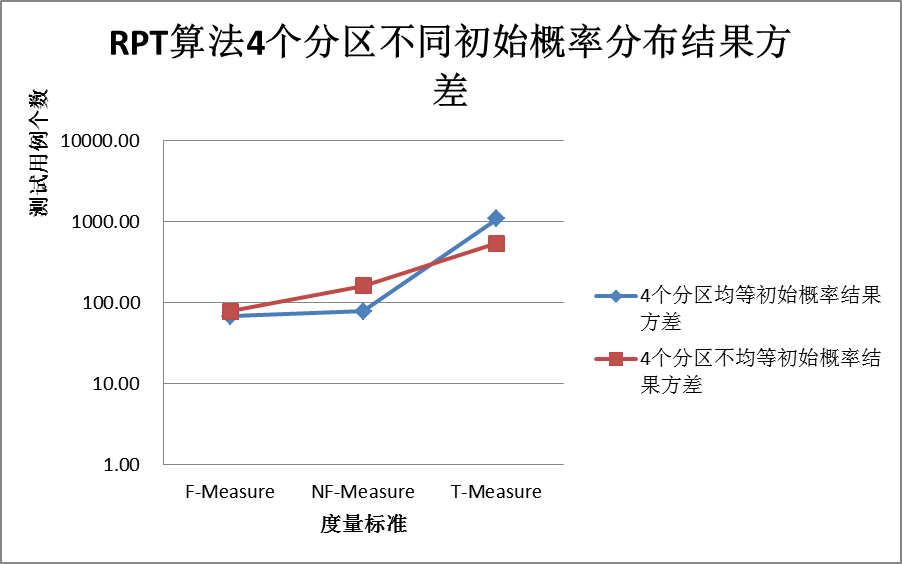


图5-14（b） RPT算法4个分区不同初始概率分布下的结果方差图

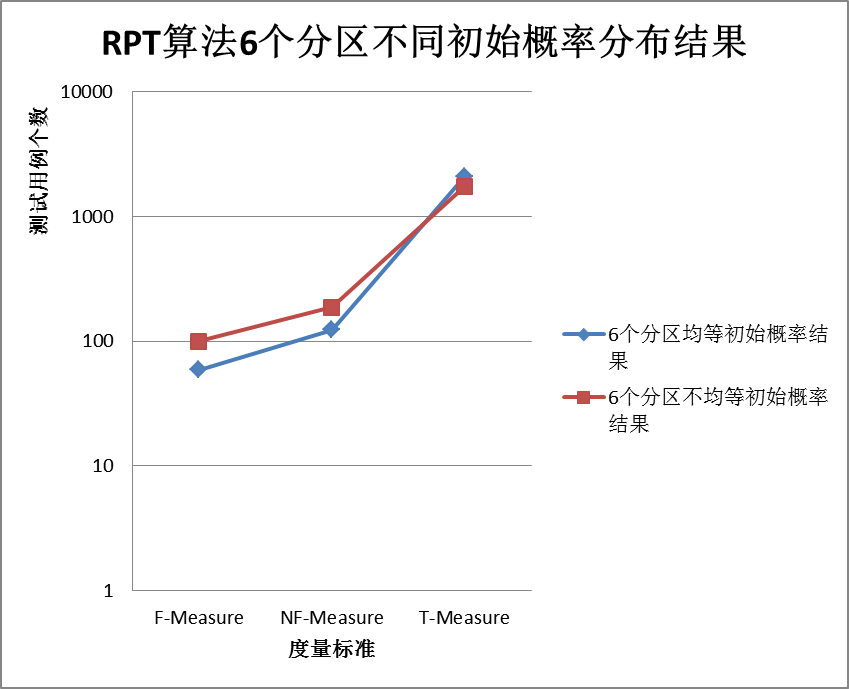


图5-14（c） RPT算法6个分区不同初始概率分布下的结果图

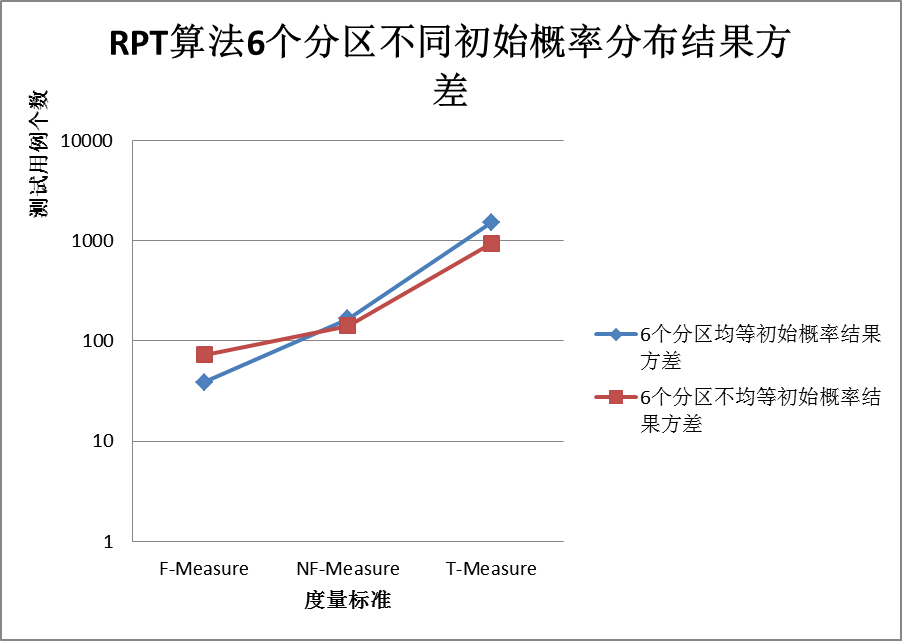


图5-14（d） RPT算法6个分区不同初始概率分布下的结果方差图

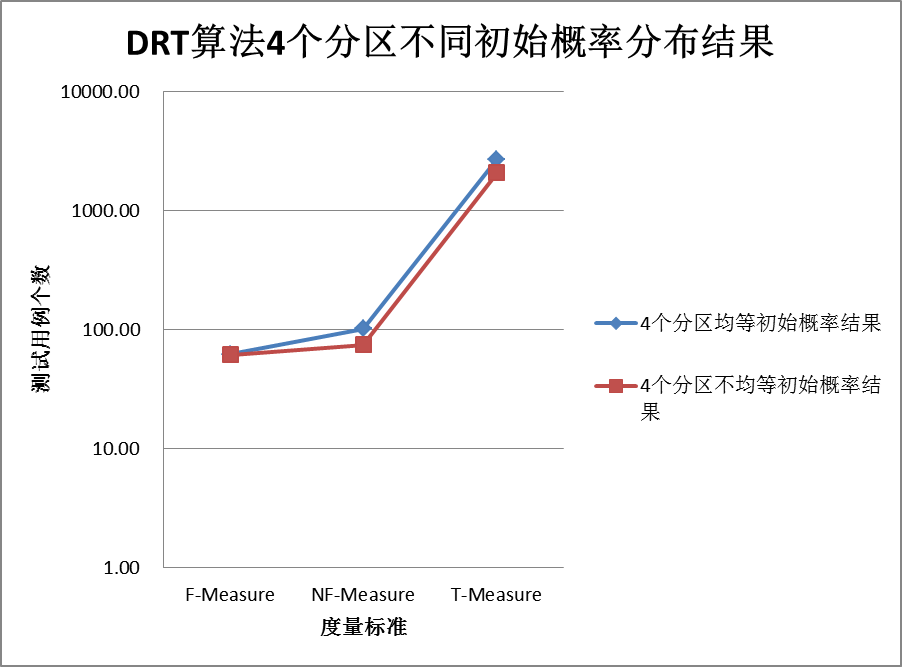


图5-15（a） DRT算法4个分区不同初始概率分布下的结果图

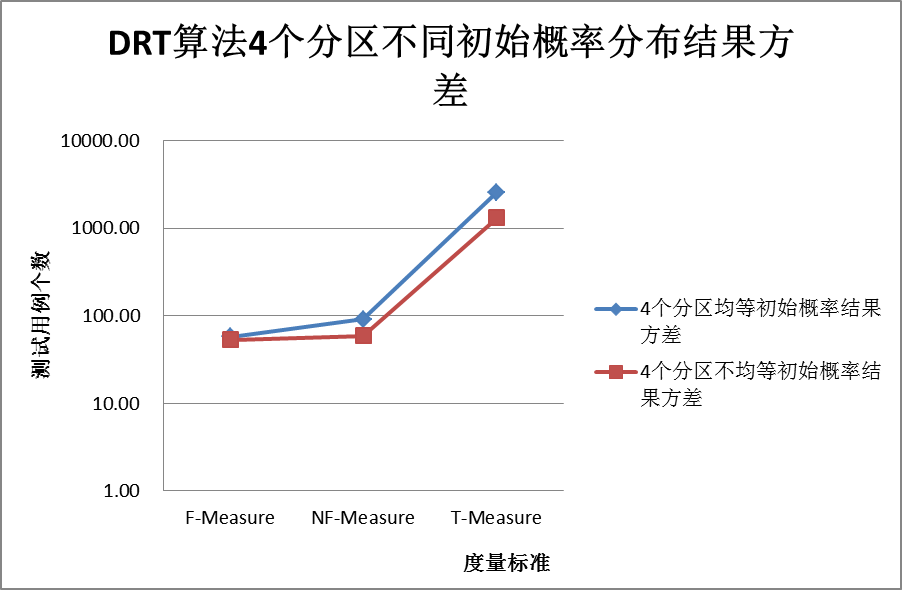


图5-15（b） DRT算法4个分区不同初始概率分布下的结果方差图

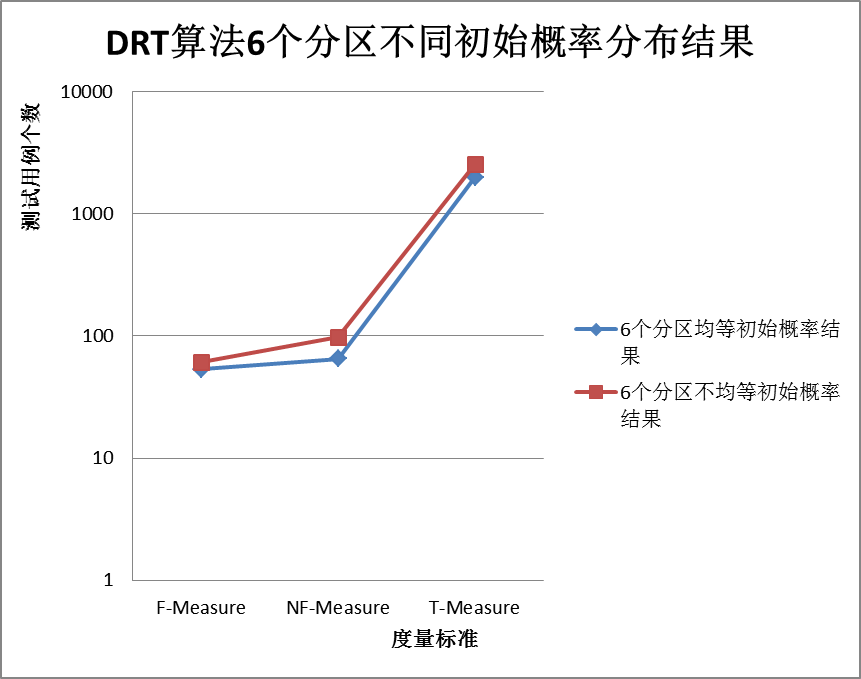


图5-15（c） DRT算法6个分区不同初始概率分布下的结果图

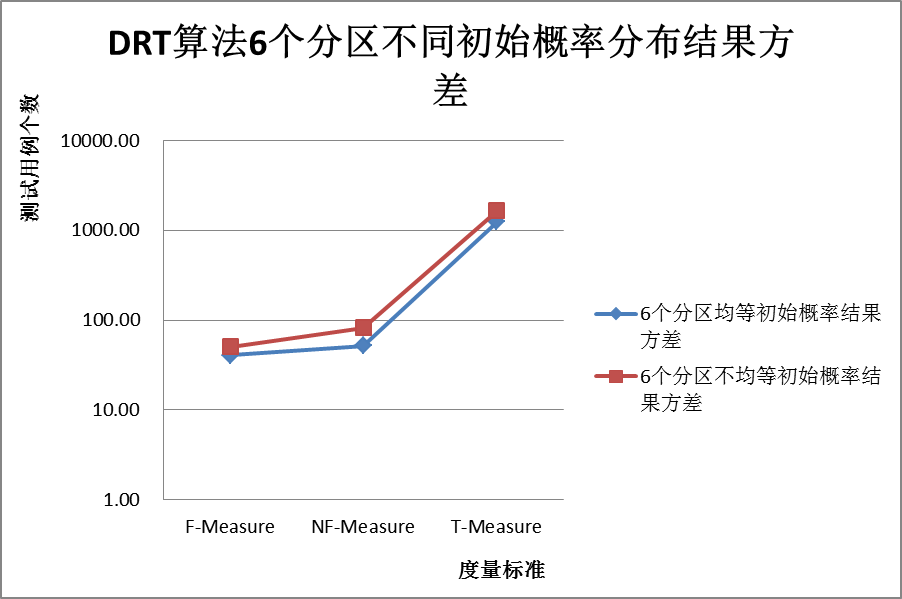


图5-15（d） DRT算法6个分区不同初始概率分布下的结果方差图

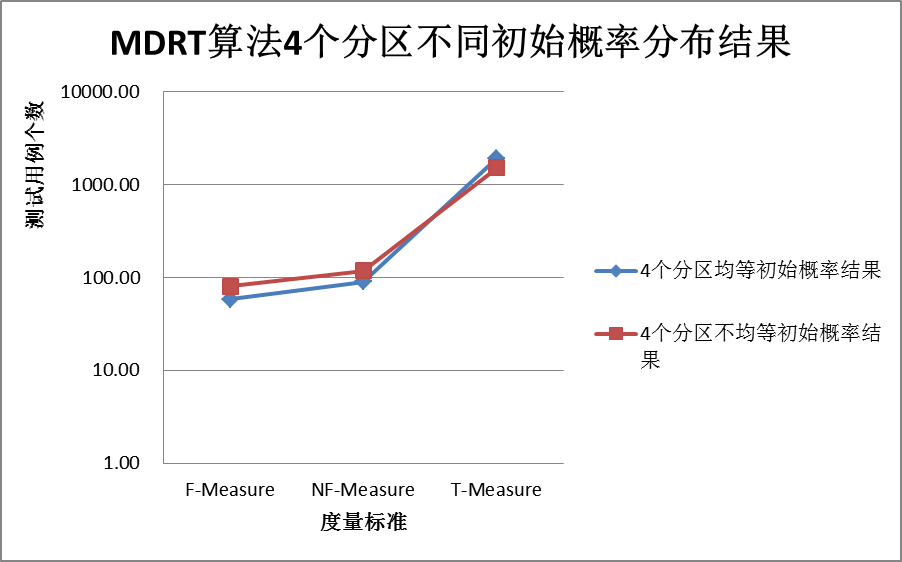


图5-16（a） MDRT算法4个分区不同初始概率分布下的结果图

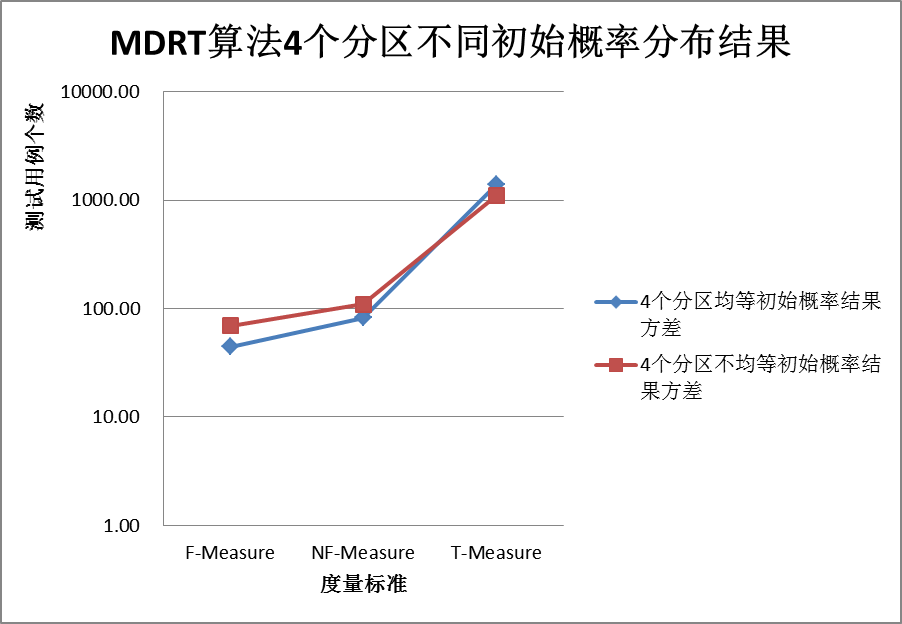


图5-16（b） MDRT算法4个分区不同初始概率分布下的结果方差图

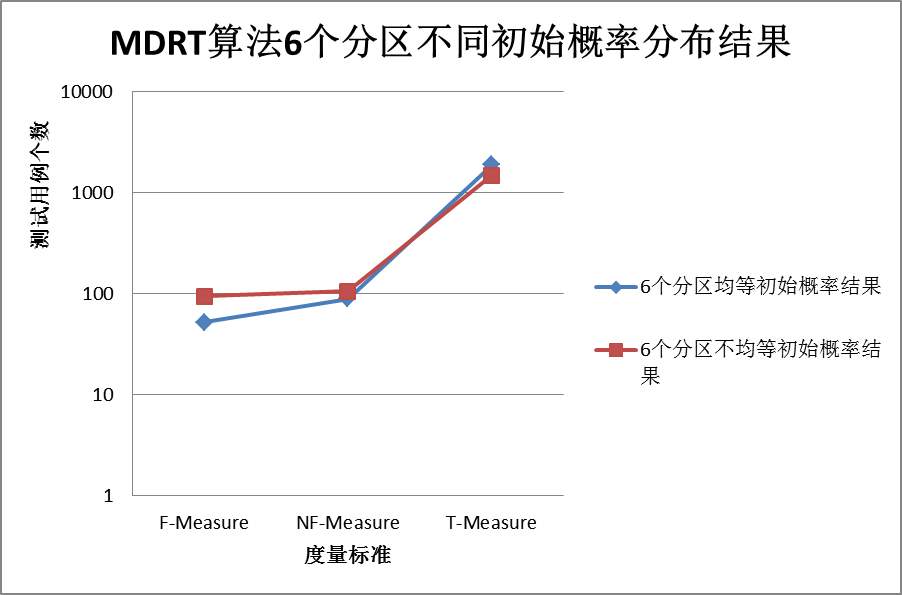


图5-16（c） MDRT算法6个分区不同初始概率分布下的结果图

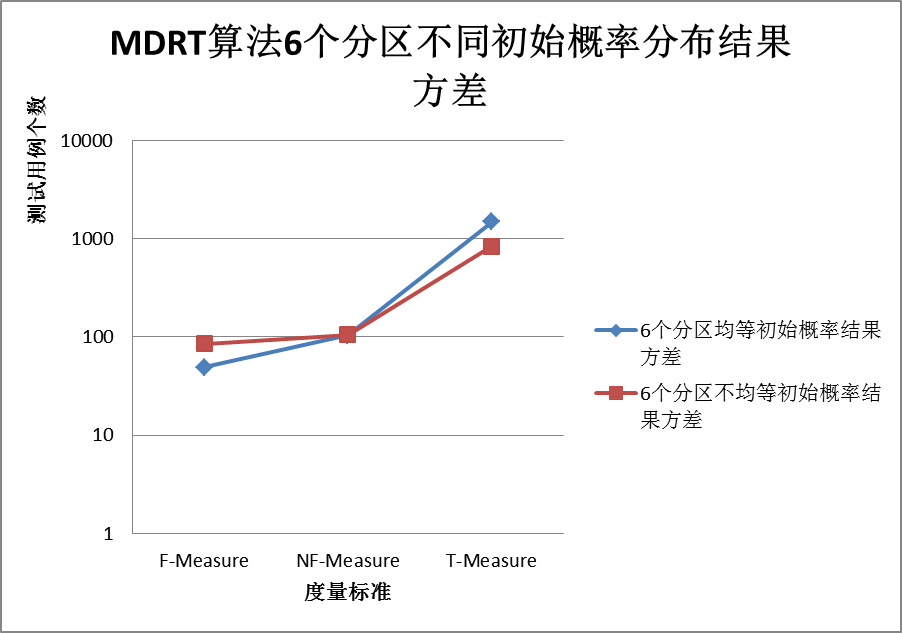


图5-16（d） MDRT算法6个分区不同初始概率分布下的结果方差图

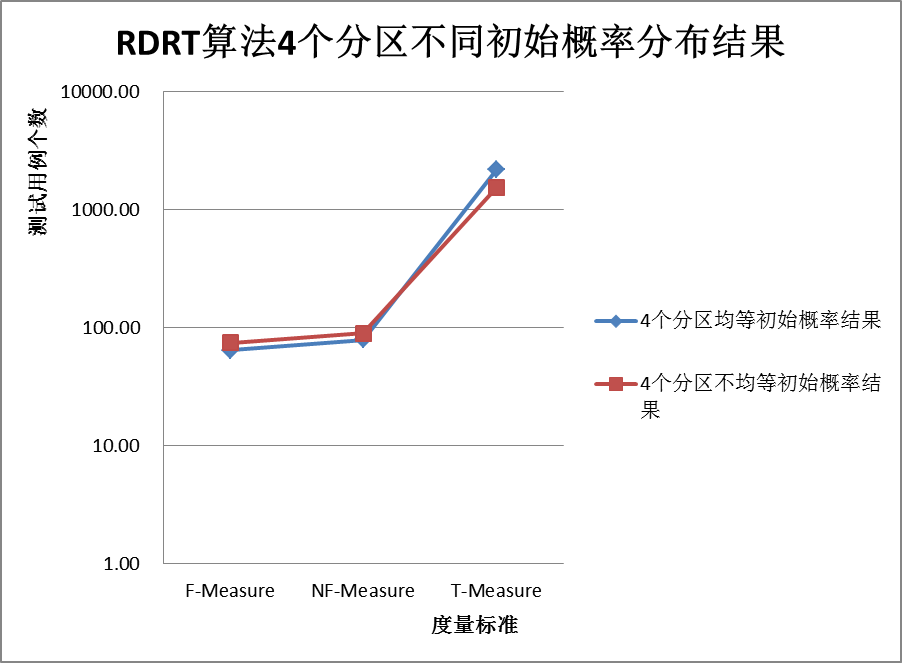


图5-17（a） RDRT算法4个分区不同初始概率分布下的结果图

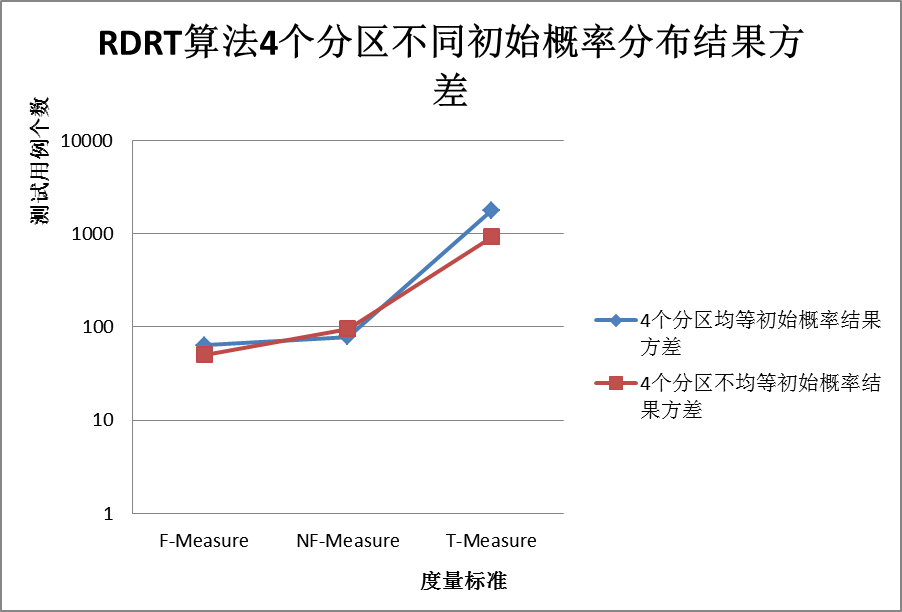


图5-17（b） RDRT算法4个分区不同初始概率分布下的结果方差图

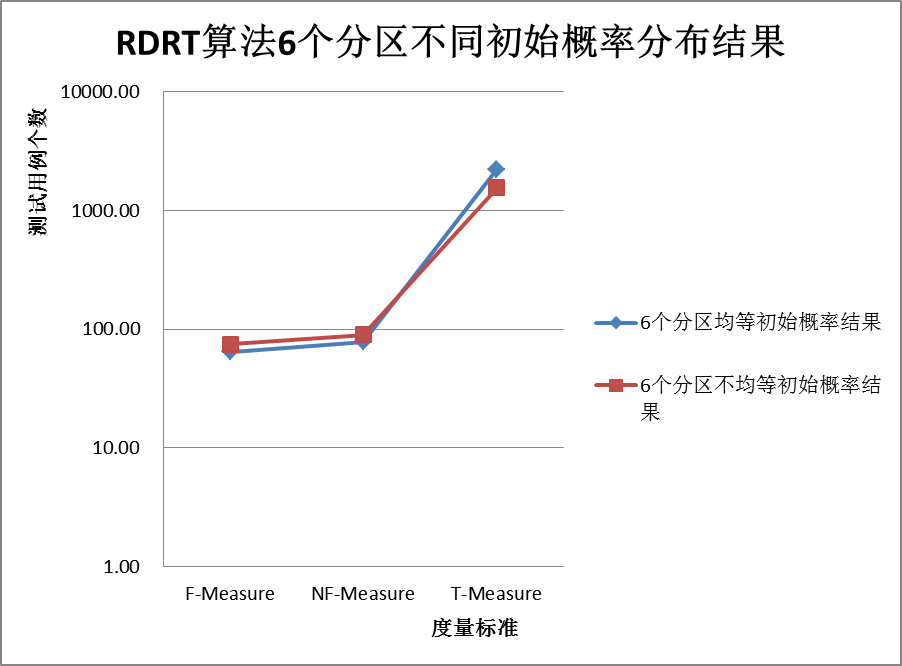


图5-17（c） RDRT算法6个分区不同初始概率分布下的结果图

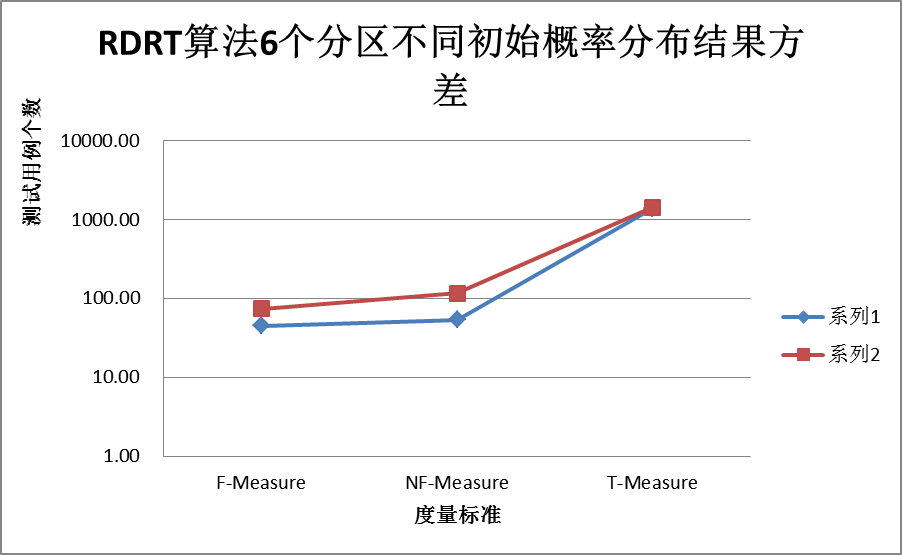


图5-17（d） RDRT算法6个分区不同初始概率分布下的结果方差图

通过图5-14、5-15、5-16、5-17，可以看出，在大多数情况下，均等初始概率分布比不均等的初始概率分布在发现第一个以及第二个缺陷时表现的更好，但是不均等的初始概率分布在找到所有的缺陷时，表现更好。

#### 5.4.2.5 bash实验初始概率研究

该实验的设计方案在上面的章节中已经展示，利用功能分区的思想，将所有的测试用例划分为4个分区和6个分区。分别对这两种方案设置两种初始概率分布：均等概率分布和不均等概率分布，实验结果如图5-18、5-19、5-20、5-21。由于该实验杀死所有的测试用例所需要的测试用例数目比较多，为了更清晰的表示出各个算法在不同的初始概率分布下的测试结果，本文将测试用例数目取以10为底的自然对数。

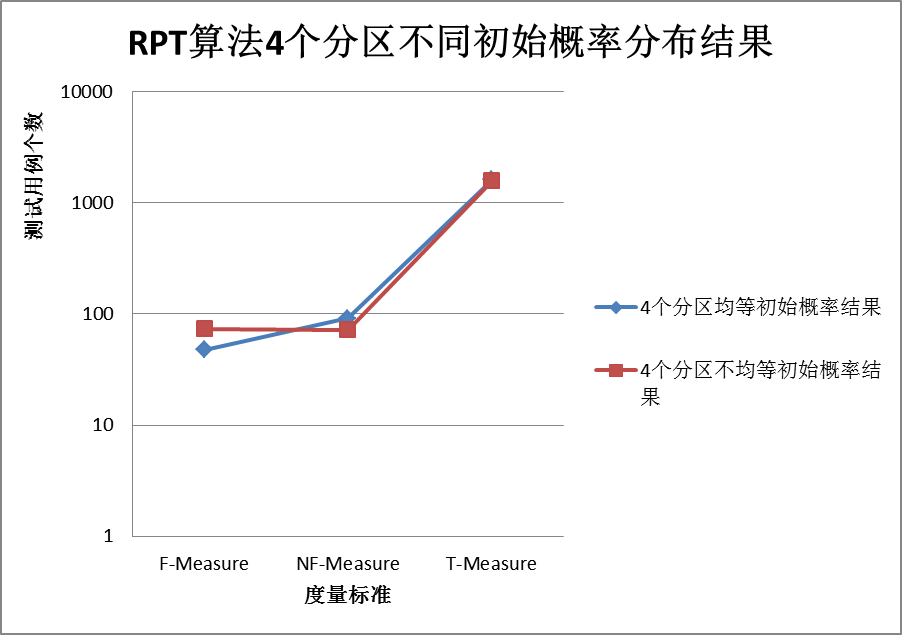


图5-18（a） RPT算法4个分区不同初始概率分布下的结果图

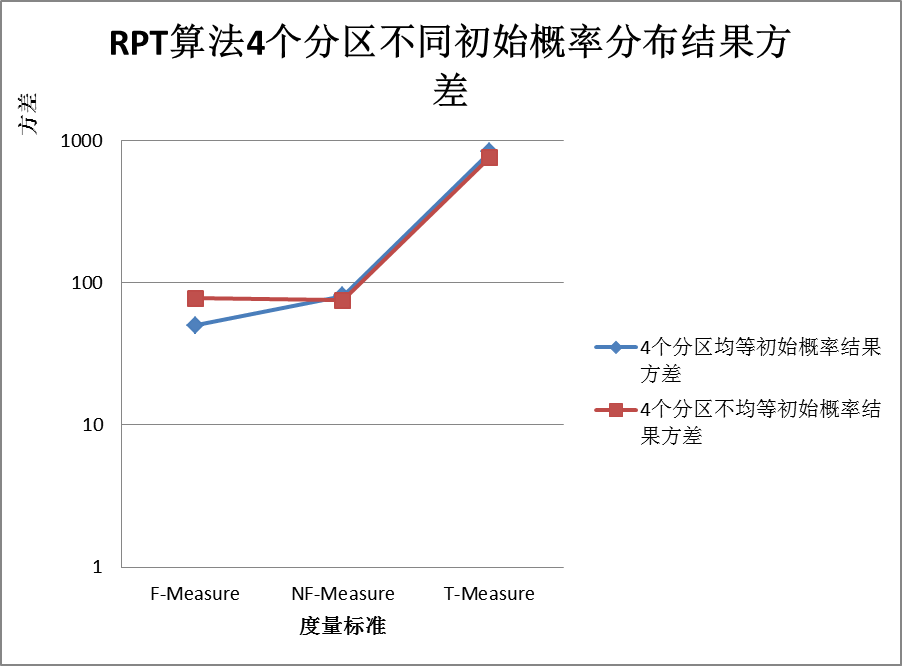


图5-18（b） RPT算法4个分区不同初始概率分布下的结果方差图

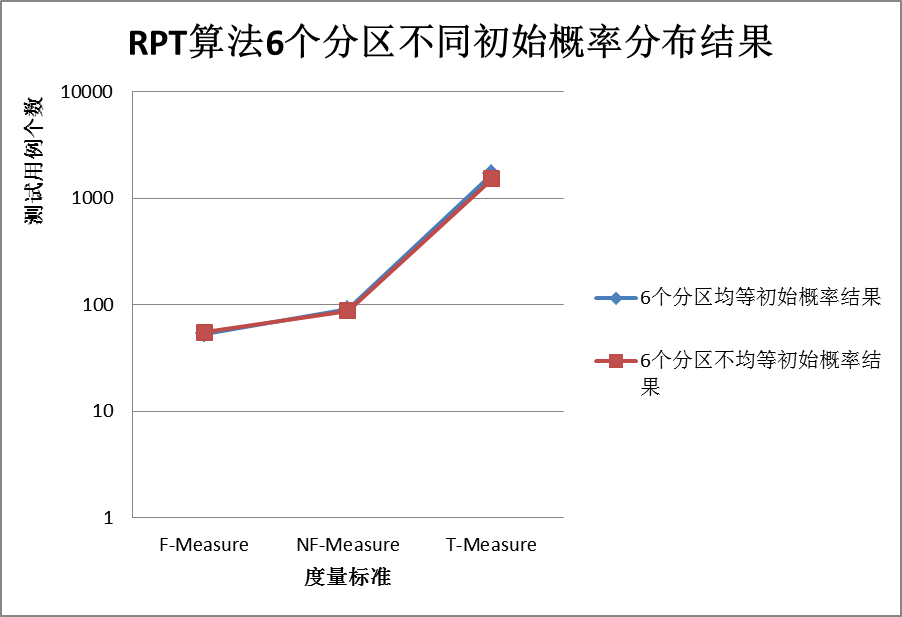


图5-18（c） RPT算法6个分区不同初始概率分布下的结果图

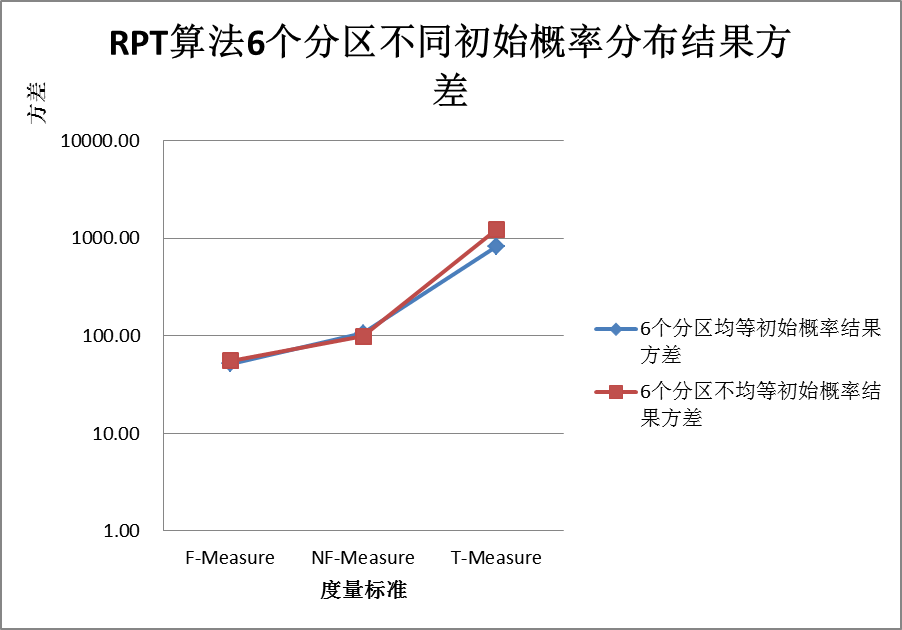


图5-18（d） RPT算法6个分区不同初始概率分布下的结果方差图

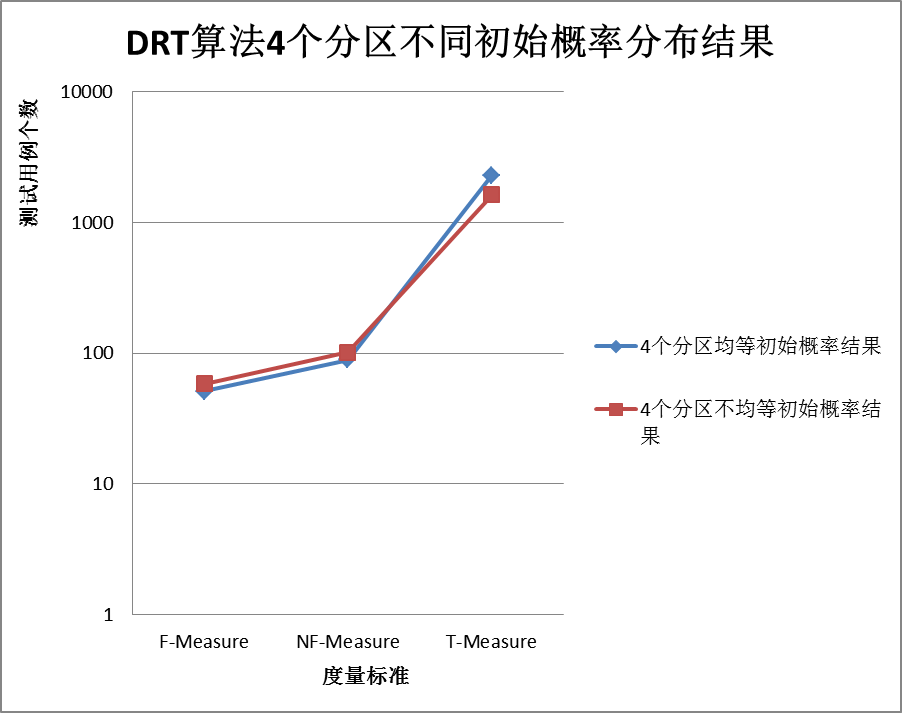


图5-19（a） DRT算法4个分区不同初始概率分布下的结果图

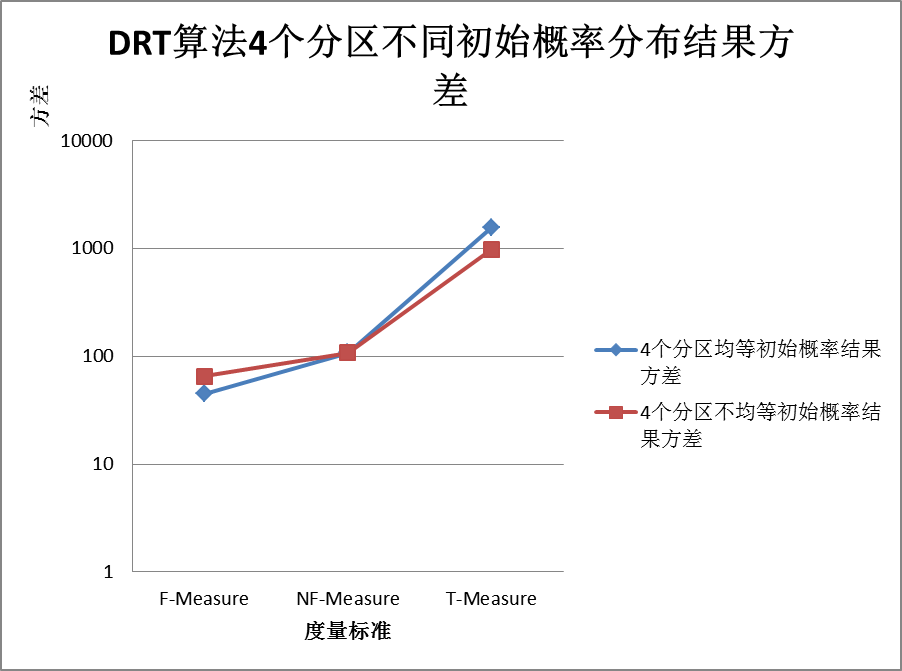


图5-19（b） DRT算法4个分区不同初始概率分布下的结果方差图

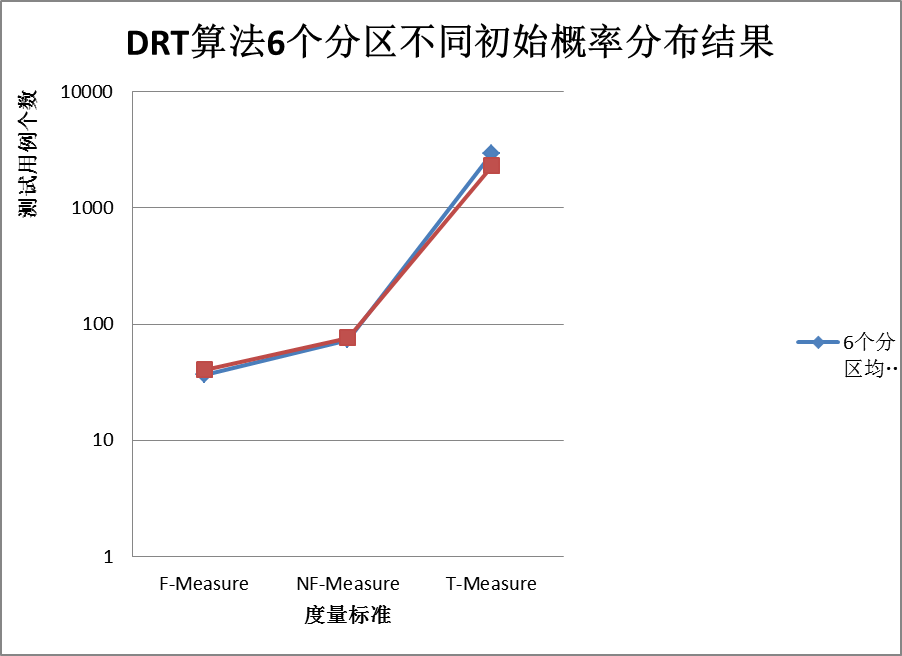


图5-19（c） DRT算法6个分区不同初始概率分布下的结果图

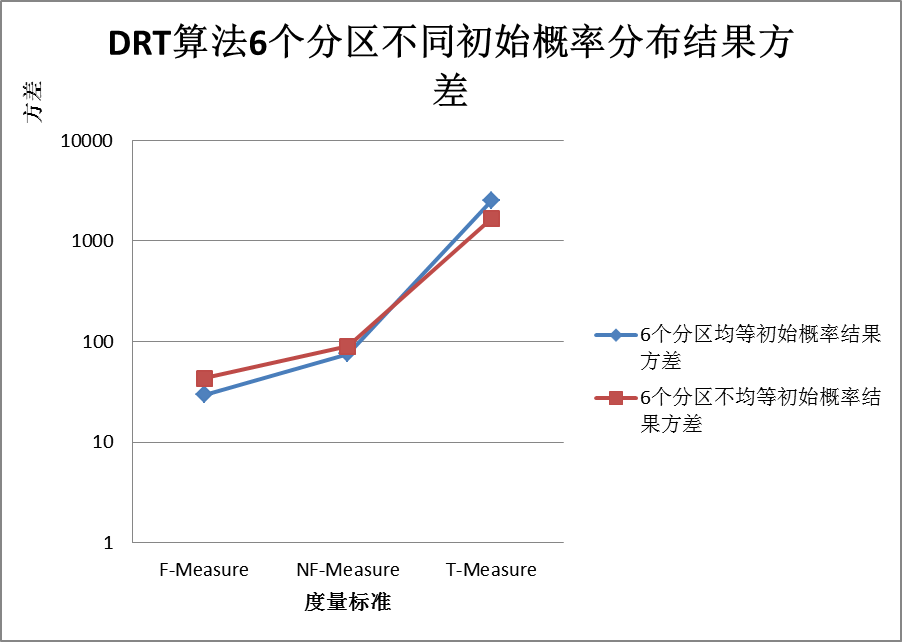


图5-19（d） DRT算法6个分区不同初始概率分布下的结果方差图

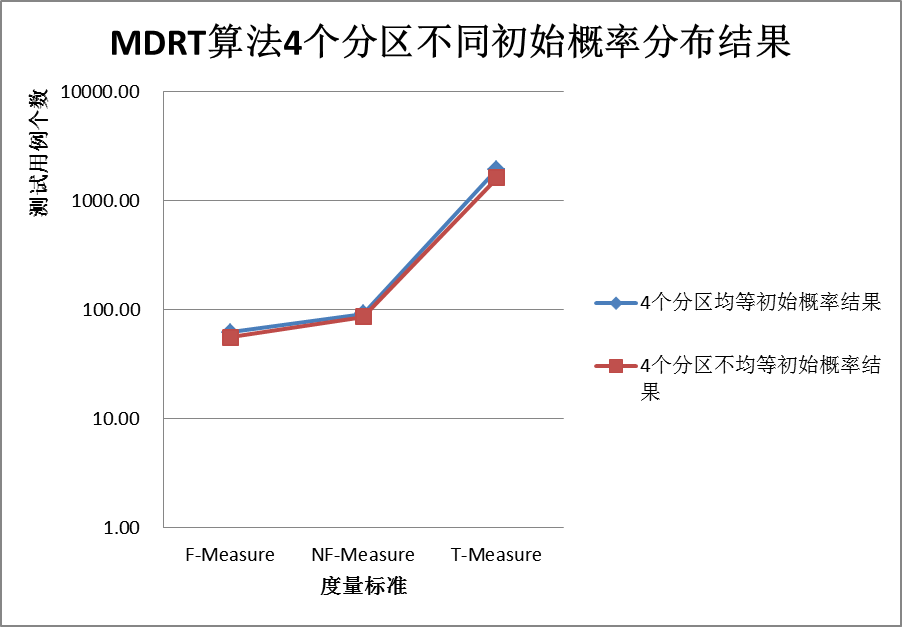


图5-20（a） MDRT算法4个分区不同初始概率分布下的结果图

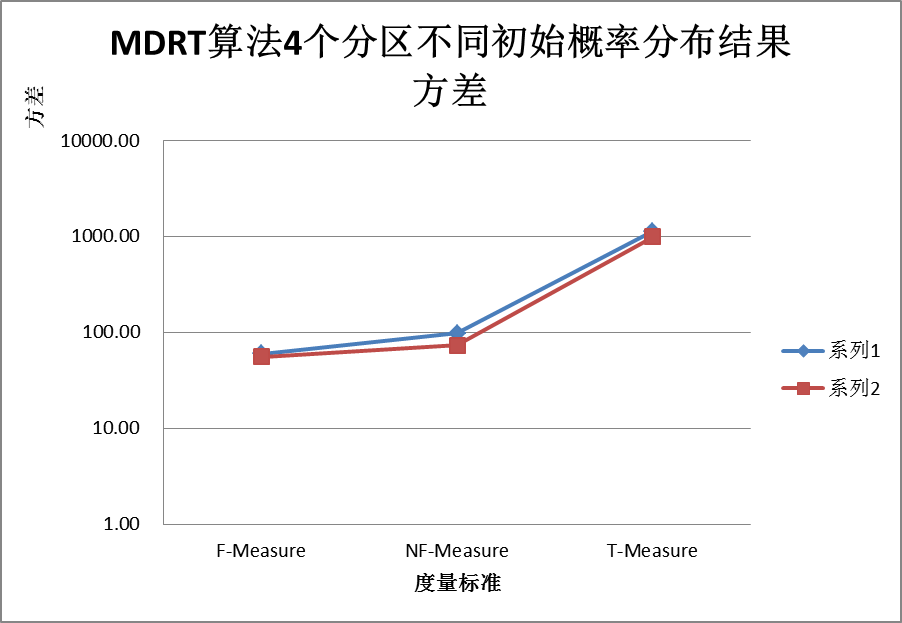


图5-20（b） MDRT算法4个分区不同初始概率分布下的结果方差图

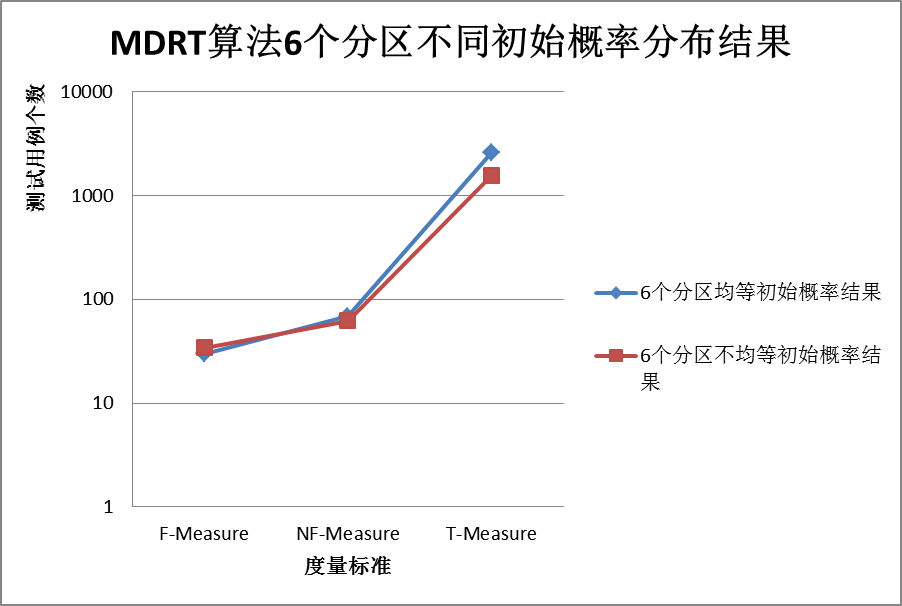


图5-20（c） MDRT算法6个分区不同初始概率分布下的结果图

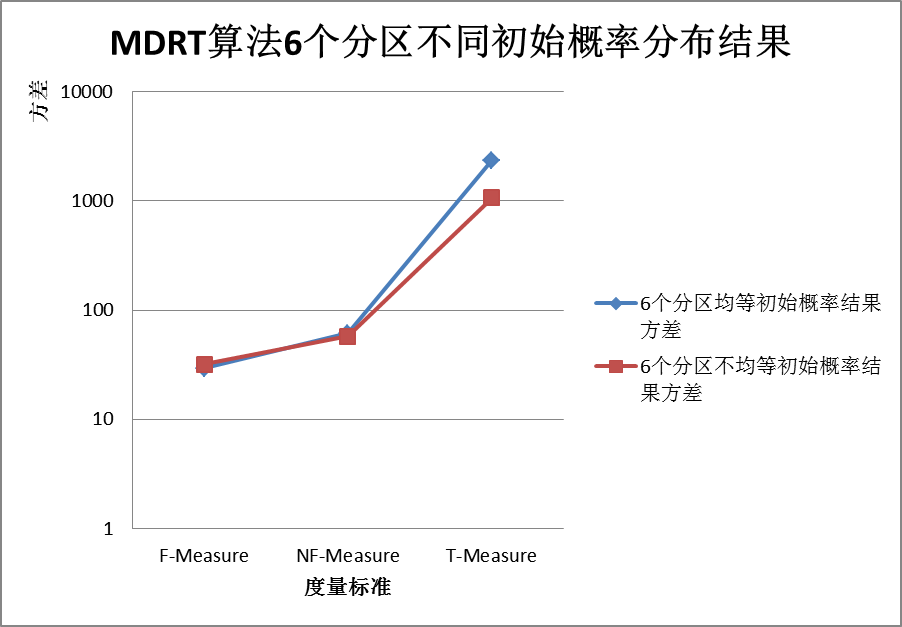


图5-20（d） MDRT算法6个分区不同初始概率分布下的结果方差图

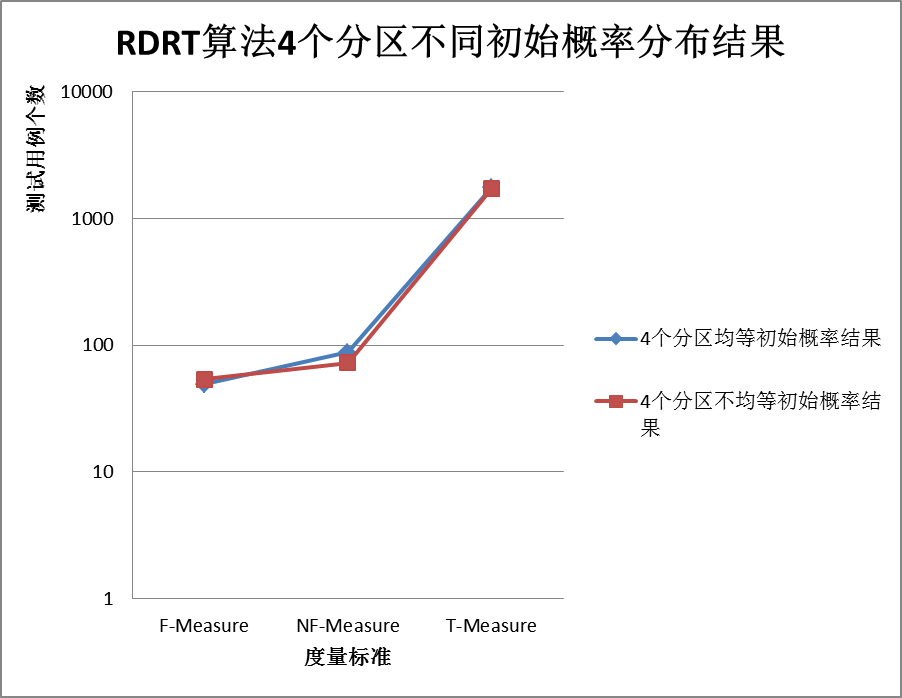


图5-21（a） RDRT算法4个分区不同初始概率分布下的结果图

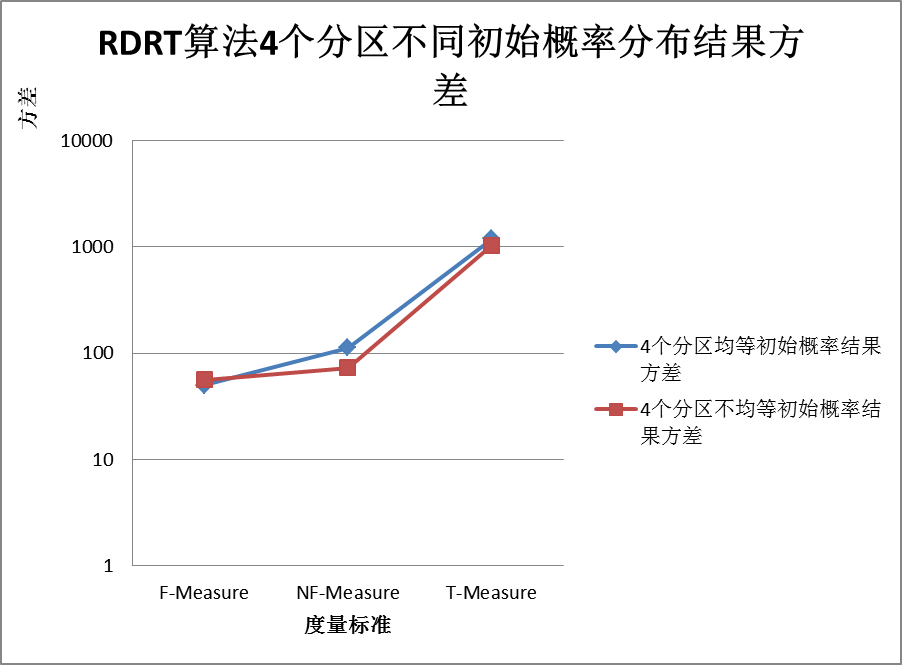


图5-21（b） RDRT算法4个分区不同初始概率分布下的结果方差图

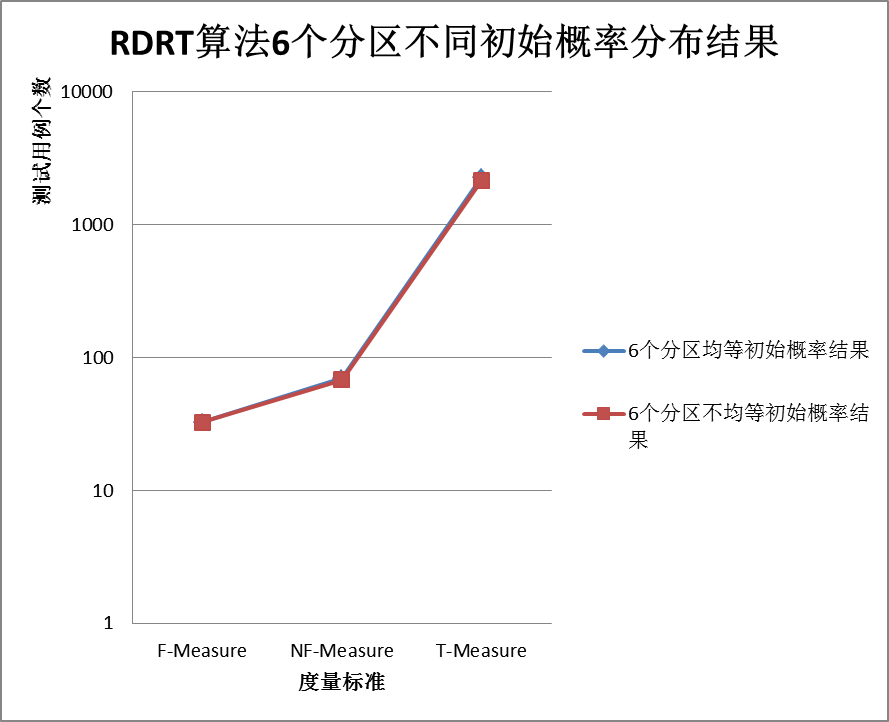


图5-21（c） RDRT算法6个分区不同初始概率分布下的结果图

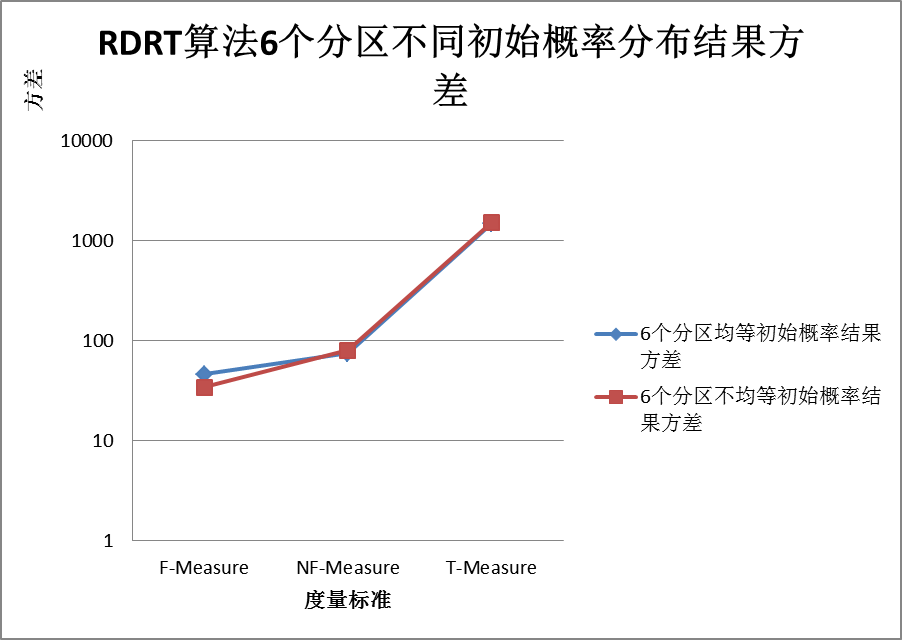


图5-21（d） RDRT算法6个分区不同初始概率分布下的结果方差图

通过图5-18、5-19、5-20、5-21，可以看出，除了make实验，当存在一个失效率比较大的分区时，不均等初始概率分布具有更高的检测效率。当所有的分区失效率很低时，不同的初始概率分布在检测第一个和第二个缺陷时差别不大，但是不均等的初始概率分布在检测所有的失效时具有更高的效率。

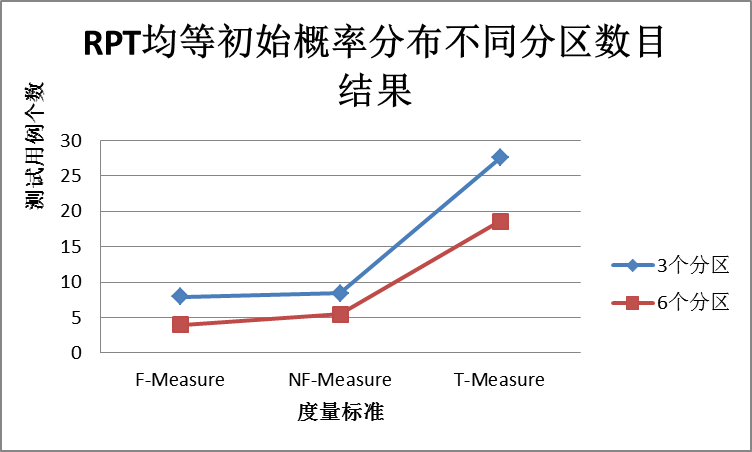
Make实验有表5-23可以知道，该实验存在一个分区具有很高的失效率，但是该分区的测试用例很少，以至于按照分区内测试用例数占所有测试用例的百分比作为初始概率时，该分区被选择的初始概率很小。由此均等分布的检测效率更高。

### 5.4.3 算法分区多少对实验的影响

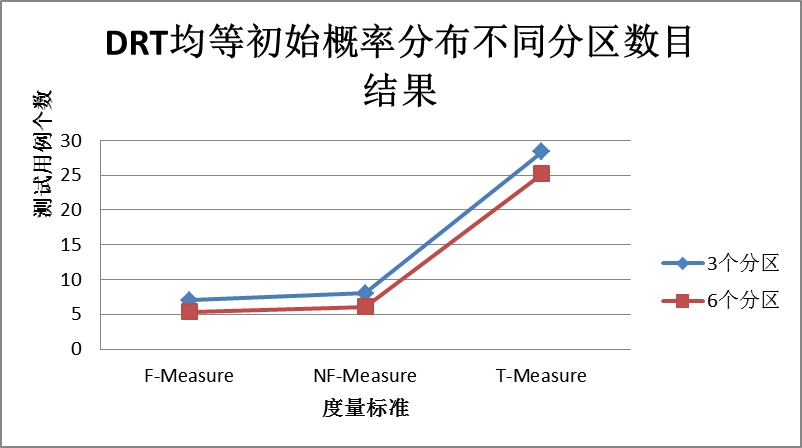
在软件测试中，基于分区思想的算法例如：PT、RPT、DRT以及本文提出的MDRT、RDRT，不同的分区对它们的影响很大。输入域的失效率指的是当在输入域中随机选取一个测试用例，检测到软件中存在缺陷的概率。由此我们可以定义每一个分区的失效率。由于本文在实验过程中测试用例执行完之后依然放回输入域中，因此在整个软件测试过程中，每一个分区的失效率是不变的。对于同一个输入域不同的分区方式可能把能够检测出软件中存在缺陷的测试用例重新分布，因此会导致分区的失效率改变。另一方面，分区数目不同也会导致分区的失效率发生改变。分区内测试用例数目增加或者减少都会影响分区的失效率。

直觉上，如果能够检测出错误的测试用例都集中在一个分区内，此时基于软件控制理论的算法：DRT、MDRT、RDRT效果最好。但是在软件测试之前测试人员并不知道哪些测试用例可以杀死变异体，因此不知道怎么分区才能让可以检测出软件失效的测试用例更加集中。

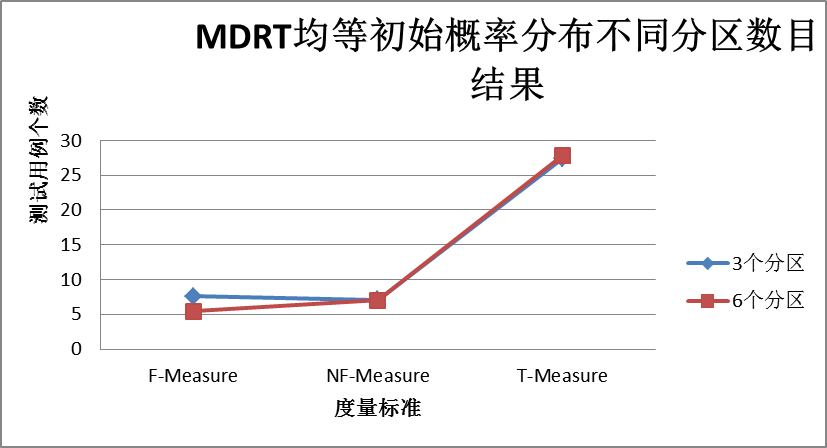
本文通过实验的方式结果如图5-22、5-23、5-24、5-25、5-26，在图5-25、5-26中，为了更直观的体现不同分区的差别，纵坐标采用了以10为底的对数坐标。总结出一条启发式规则：当整个输入域的失效率较小时，一般情况下，应当尽可能的细致的划分分区。当整个输入域的失效率较低的时候，由于能够检测出软件中存在缺陷的测试用例具有聚簇性，分区越细致分配到每个分区的测试用一般就越少，就有可能使得分区的失效率变大。因此当软件中的输入域不容易检测出软件中存在缺陷时，划分分区的数目要多一点，即，若分区方式为功能分区，则将功能考虑的细致一些。



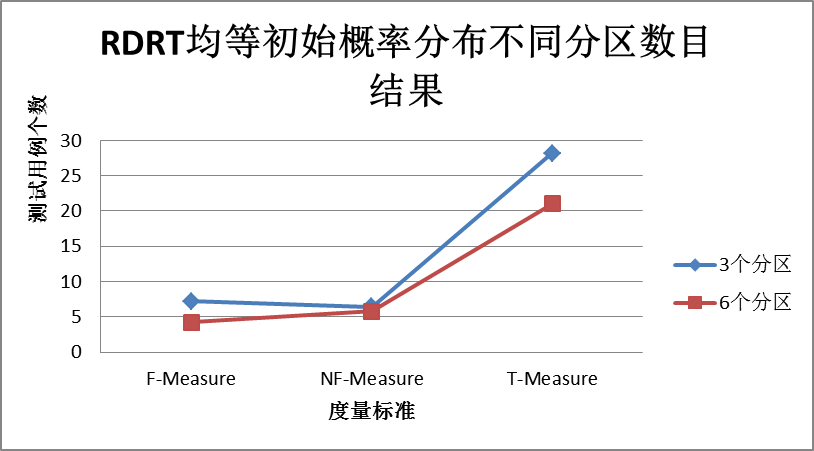
5-22（a）flex实验RPT算法均等初始概率下不同分区数目结果



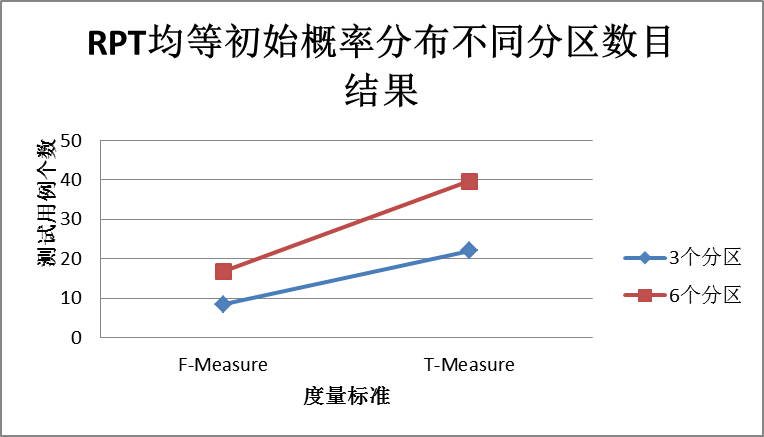
5-22（b）flex实验DRT算法均等初始概率下不同分区数目结果



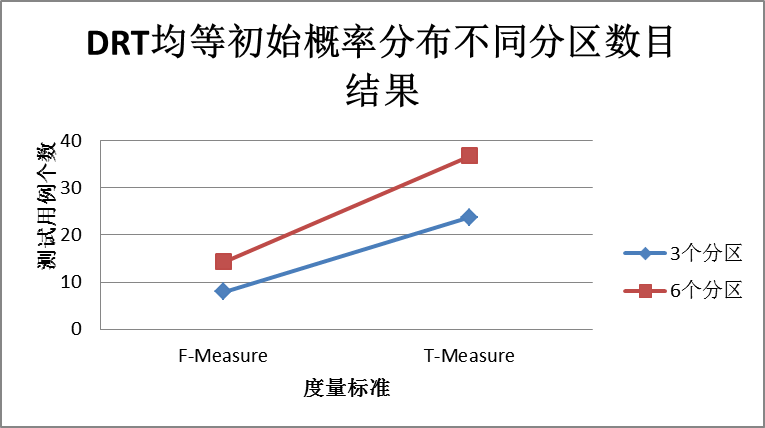
5-22（c）flex实验MDRT算法均等初始概率下不同分区数目结果



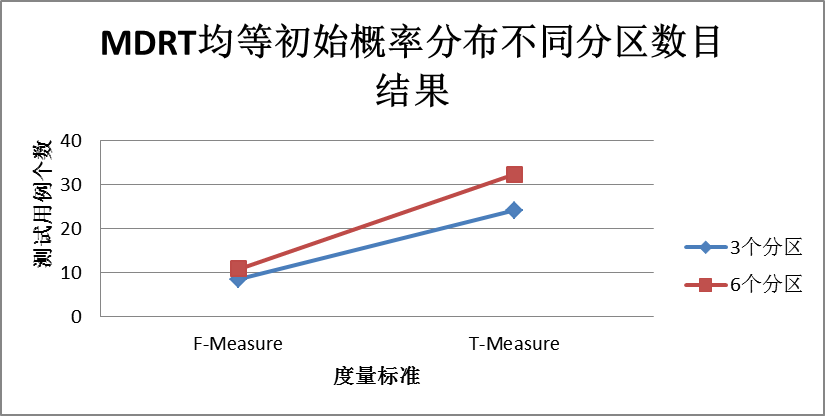
5-22（d）flex实验RDRT算法均等初始概率下不同分区数目结果



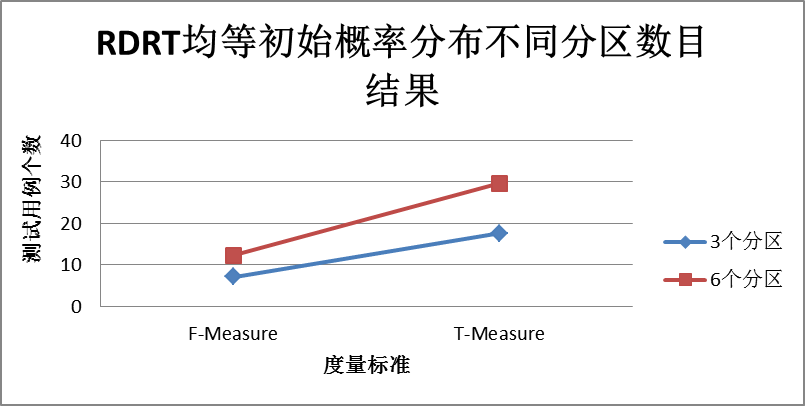
5-23（a）make实验RPT算法均等初始概率下不同分区数目结果



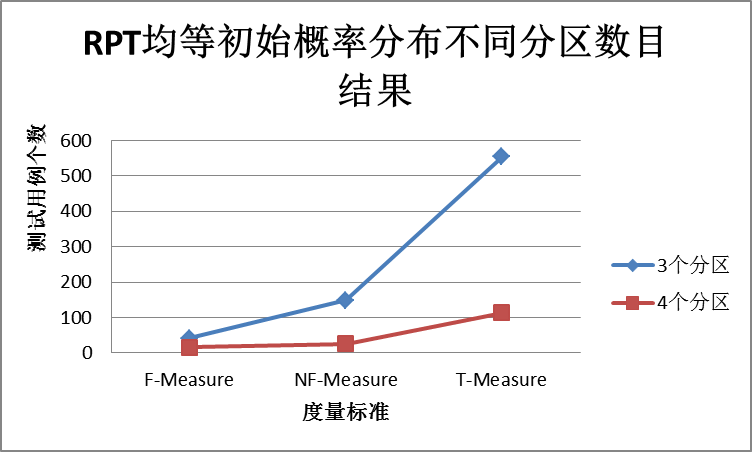
5-23（b）make实验DRT算法均等初始概率下不同分区数目结果



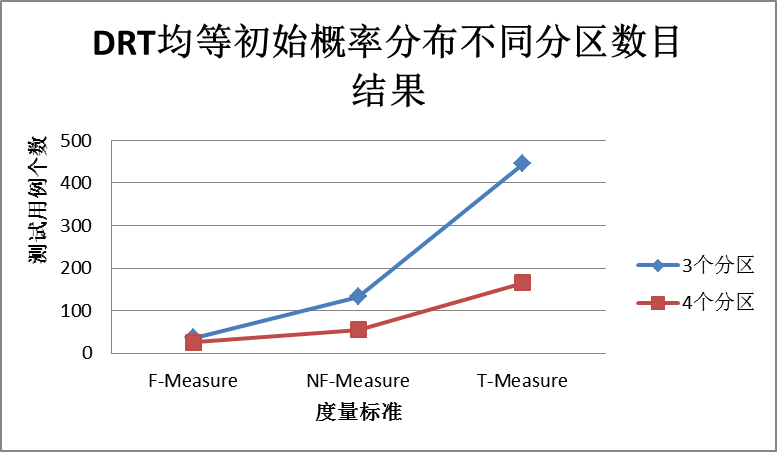
5-23（c）make实验MDRT算法均等初始概率下不同分区数目结果



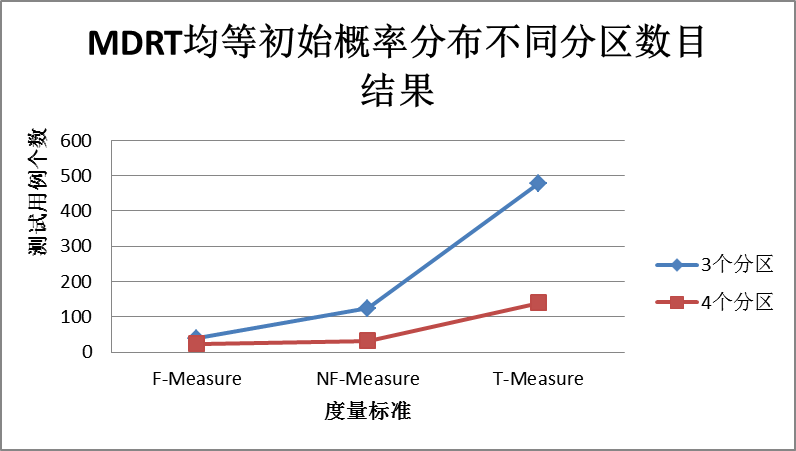
5-23（d）make实验RDRT算法均等初始概率下不同分区数目结果



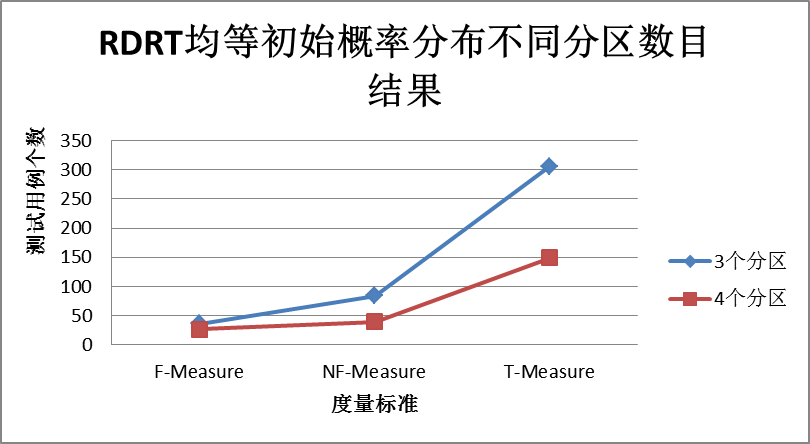
5-24（a）gzip实验RPT算法均等初始概率下不同分区数目结果



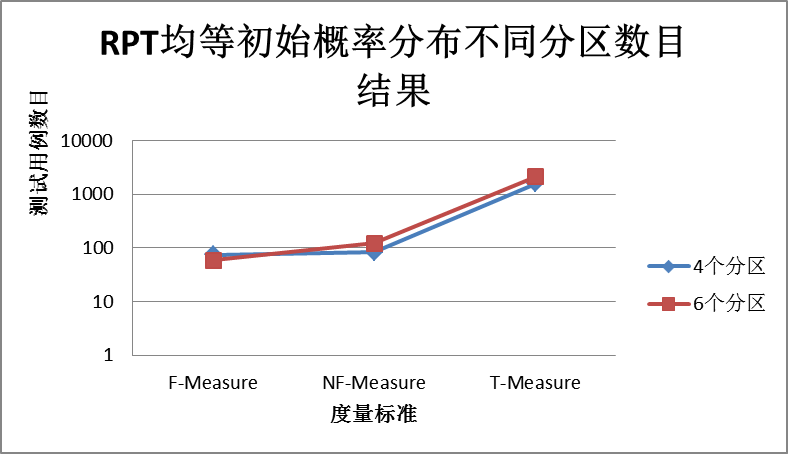
5-24（b）gzip实验DRT算法均等初始概率下不同分区数目结果



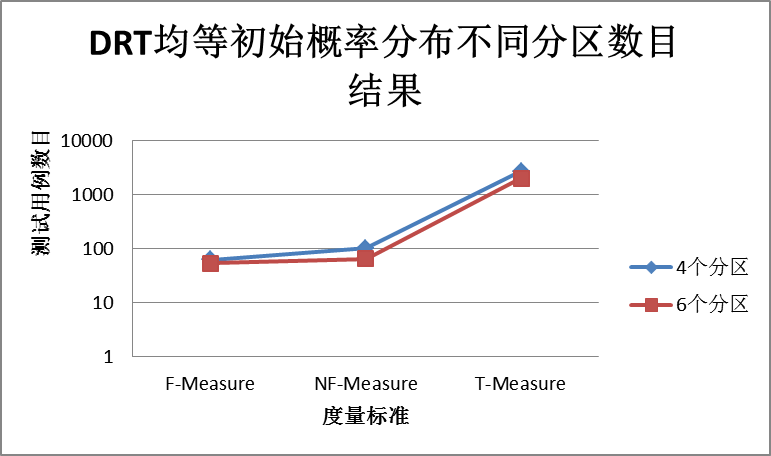
5-24（c）gzip实验MDRT算法均等初始概率下不同分区数目结果



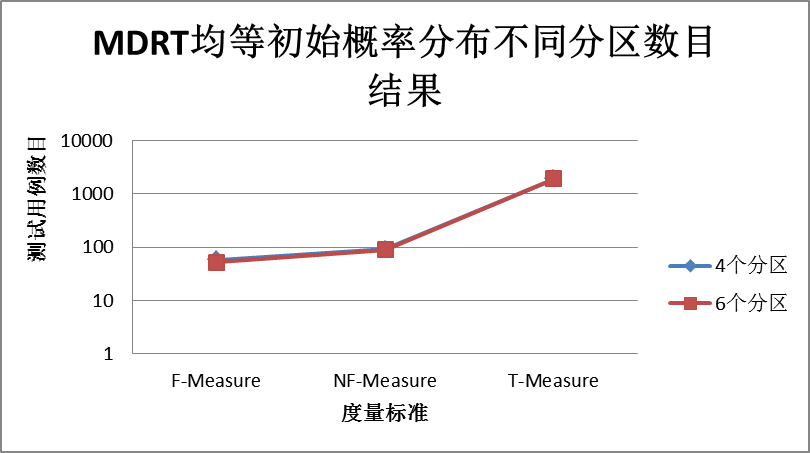
5-24（d）gzip实验RDRT算法均等初始概率下不同分区数目结果



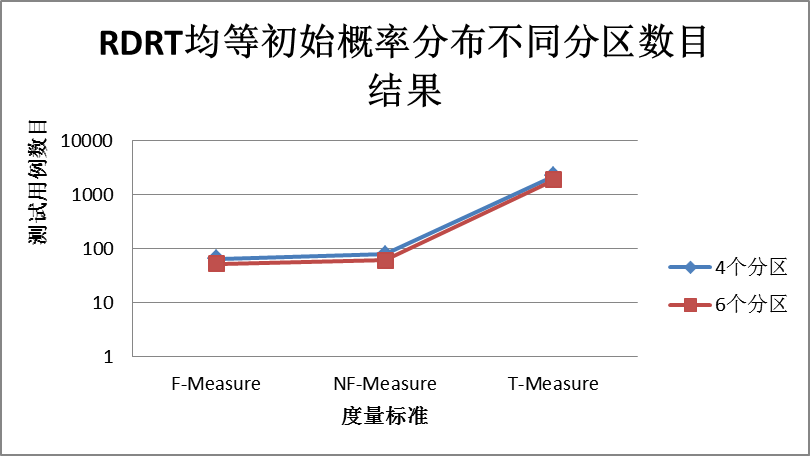
5-25（a）grep实验RPT算法均等初始概率下不同分区数目结果



5-25（b）grep实验DRT算法均等初始概率下不同分区数目结果



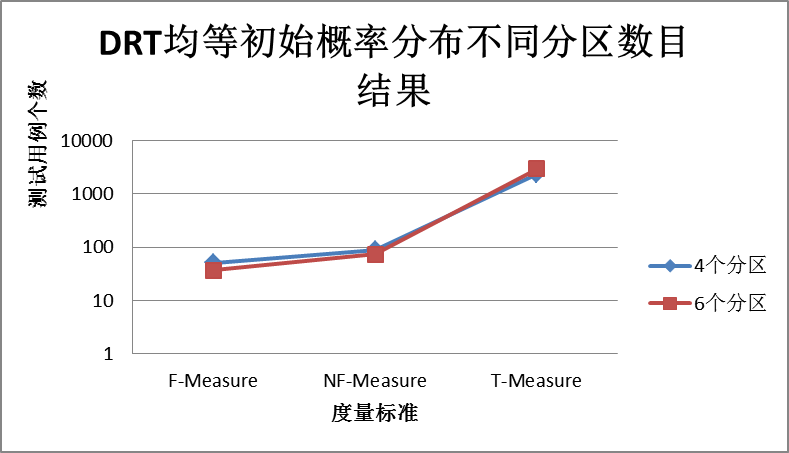
5-25（c）grep实验MDRT算法均等初始概率下不同分区数目结果



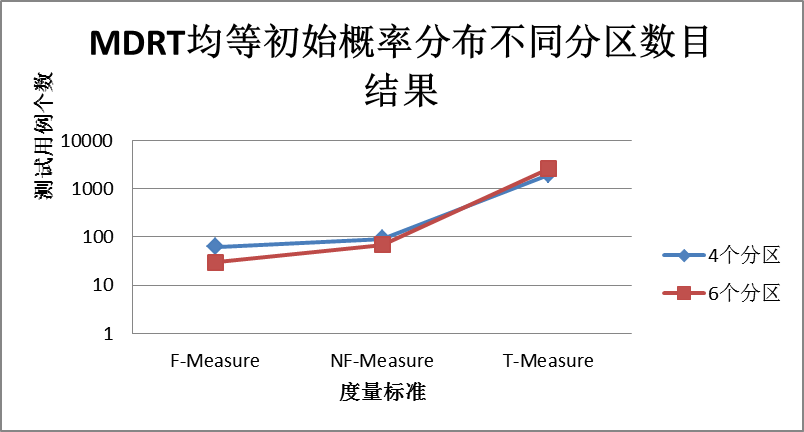
5-25（d）grep实验RDRT算法均等初始概率下不同分区数目结果



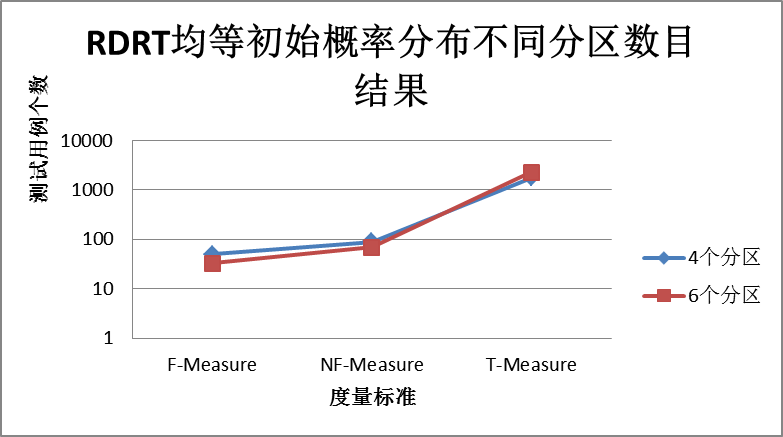
5-26（a）bash实验RPT算法均等初始概率下不同分区数目结果



5-26（b）bash实验DRT算法均等初始概率下不同分区数目结果



5-26（c）bash实验MDRT算法均等初始概率下不同分区数目结果



5-26（d）bash实验RDRT算法均等初始概率下不同分区数目结果

从图5-22、5-23、5-24、5-25、5-26可以看出，除了make实验，其它四个实验在不同的初始概率分布下，分区较为细致的分区方式，即，分区数目较多的方案，比分区数目较少的分区方式测试效率更高。

make实验跟其它实验不同，分区数目小时，测试效率更高。由于make实验能杀死变异体的测试用例集中在一个分区之中，并且该分区的失效率很高，如表5-43。

表5-43 实验不同分区的失效率展示

|  |  |  |  |
| --- | --- | --- | --- |
| 实验项目 | 分区数目 | 分区失效率 | 输入域失效率 |
| flex | 3 | (6.25%,24.29%,0) | 22.93% |
|
|
| 6 | (11.11%, 0, 23.29%,0,0, 30.00%) |
|
|
| make | 3 | (36.36%,0,0) | 0.99% |
|
| 5 | (36.36%,0,0,0,0) |
|
| gzip | 3 | (1.56%,0,4.62%) | 2.34% |
|
|
| 4 | (0，15.38%，0,4.62%) |
|
|
| bash | 4 | (0,0,5.67%,1.69%) | 1.89% |
|
|
| 5 | (0,0,0,2.21%,16.21%,1.02%) |
|
|
| grep | 4 | (0,5.71%,0.690%,0) | 0.87% |
|
|
| 5 | (0,0,8.33%,0.690%,0) |
|
|

该实验下不同的分区方案能够杀死变异体的测试用例均集中在第一个分区之中，因此分区越多，在实验过程中选中第一分区能够杀死变异体的测试用例概率越小。

一般情况下，如果能够检测软件失效的测试用例不是集中在一个分区中时，较为细致的划分分区能够提高检测效率。若能够检测软件缺陷的测试用例集中在一个分区中，那么应该尽可能少地减少分区数目。

### 5.4.5 MDRT和RDRT算法的效率

前面的章节讨论了MDRT、DRT、RDRT算法，参数、分区数目以及初始概率对它们的影响，这一节将展示RT、RPT、DRT、MDRT以及RDRT的相互比较的结果。

为了验证MDRT、RDRT算法的有效性，本文选取各个实验较多的分区下的均等初始概率并在表现较好的参数区间下求得均值，结果如图5-27所示。

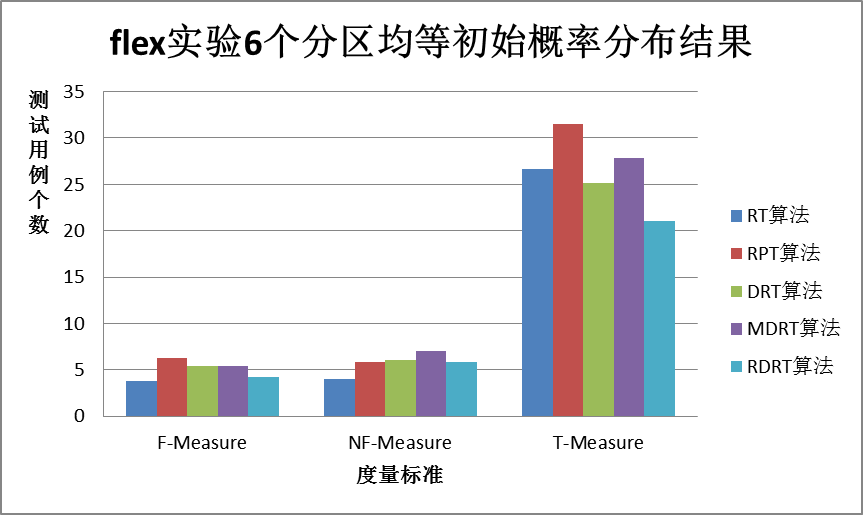
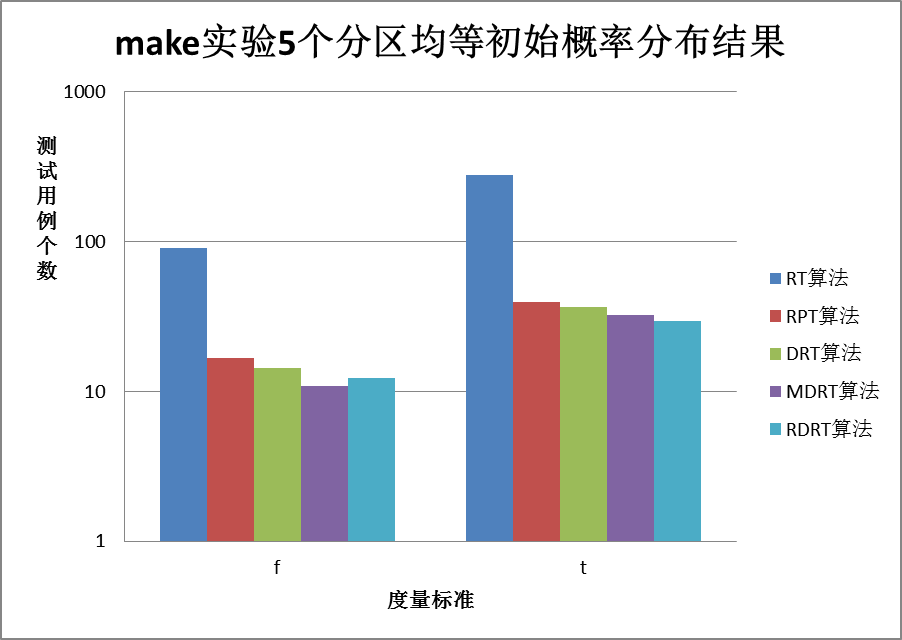
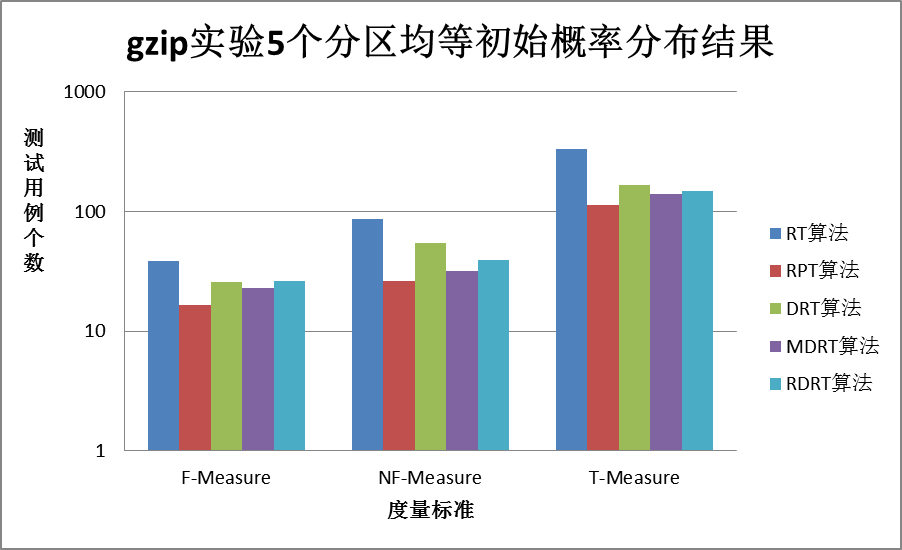


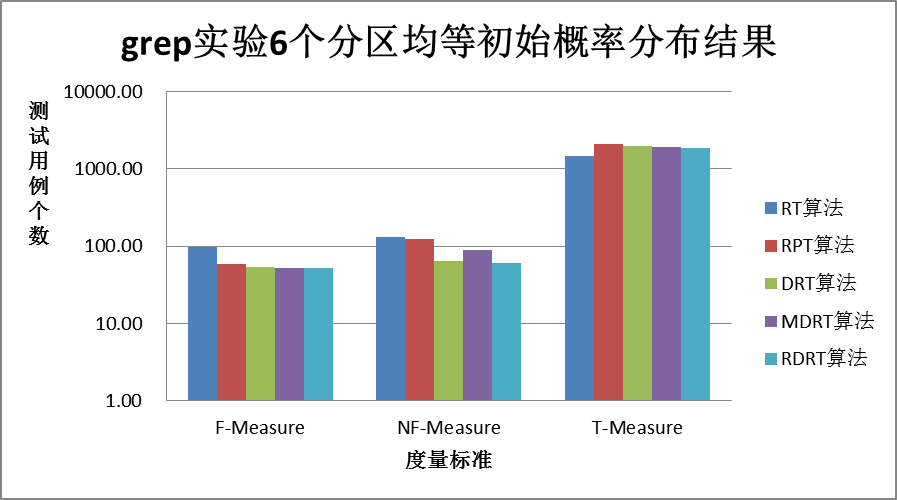
图5-27（a）flex实验各个算法比较图



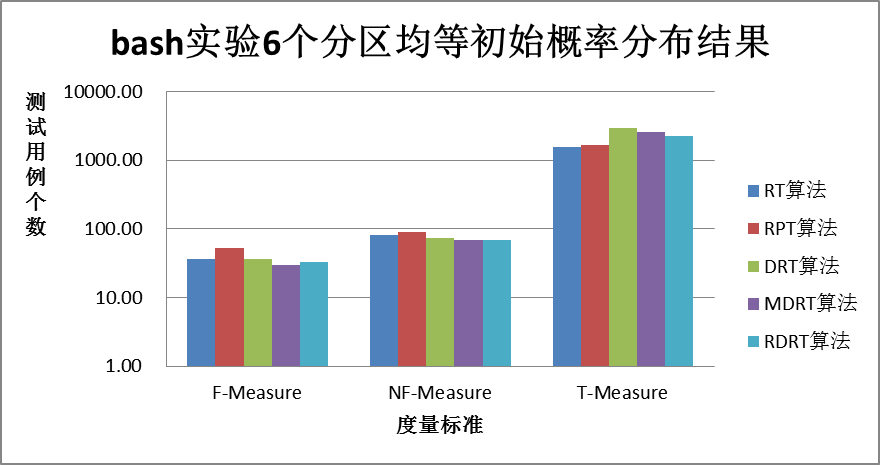
5-27（b）make实验各个算法比较图



5-27（c）gzip实验各个算法比较图



5-27（d）grep实验各个算法比较图



5-27（e）bash实验各个算法比较图

通过图5-27展示的5个实验的结果，可以看出除了flex实验，DRT、MDRT、RDRT算法存在一个失效率比较大的分区时在发现第一、第二以及最后一个缺陷比RT算法都有很大的提高。当DRT、MDRT、RDRT算法每一个分区的失效率都比较低时，在发现第一、第二个缺陷时比RT算法具有更高的检测效率。但是RT算法在检测所有的失效时，具有更高的检测效率。

flex实验由于该实验的输入域的失效率高达22.93%，在此情况下，随机选取一个测试用例有超过1/5的概率检测出失效。如果进行分区，那么特定分区内可以检测出失效的测试用例被选择的概率乘以该分区被选择的概率小于1/5。因此在输入域的失效率比较大的情况下，不必采用复杂的算法，RT算法是最优的选择。

通过5.3章节的介绍，本文所有的实验均是挑选出相对很难杀死的变异体进行试验，其中grep实验、bash实验分别有5个、6个变异体。其它实验的变异体均在3个或者3个一下。由于F-Measure、NF-Measure、T-Measure只能展示杀死第一个、第二个以及最后一个变异体的信息。grep实验以及bash实验的变异体超过了3个，因此图5-28展示了两个实验在均等的初始概率、较大分区数目以及不同算法最佳区间下杀死每一个变异体的情况。从图中可以看出grep实验与bash实验中DRT、MDRT、RDRT算法在发现第一、第二个变异体具有明显的优势，但是检测更多的缺陷时渐渐地不如RT算法。

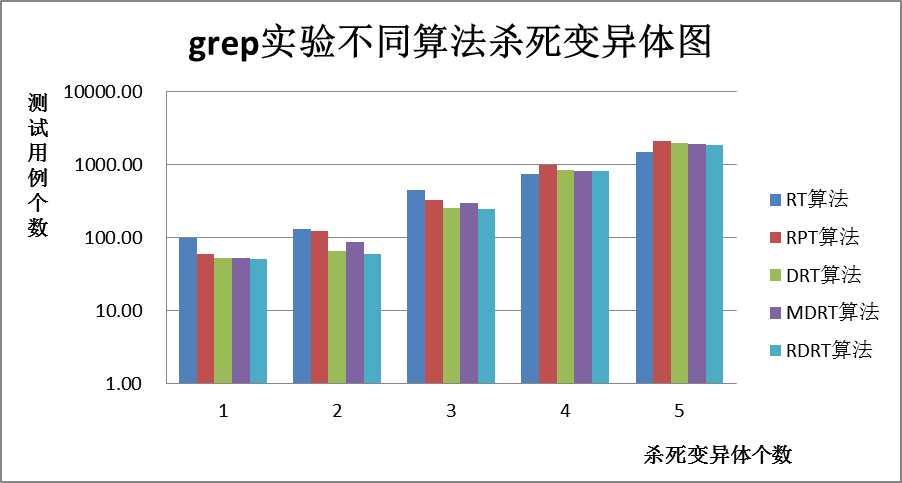


图5-28（a）grep实验不同算法杀死变异体情况

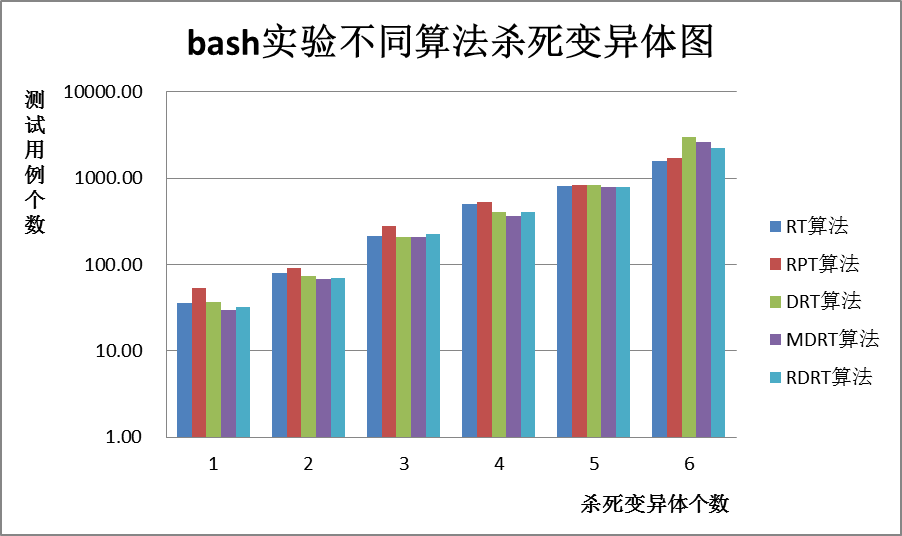


图5-28（b）bash实验不同算法杀死变异体情况

### 5.4.5 实验结论与分析

(1)评估与比较不同算法的性能

通过5.4.5章节的展示，可以看出，一方面当软件的测试的输入域的失效率比较大时（本文flex实验为23%）即便是存在分区对应的失效率也很大（本文flex实验为30%）基于控制理论的一系列算法：DRT、MDRT、RDRT等测试效率远远不如RT算法。以flex实验为例，在整个输入域随机选取一个测试用例有23%的概率检测出软件缺陷。但是对于失效率为30%的分区,如果开始时候为均等分布，那么实际上从挑选测试用例并且杀死的概率为5%（6个分区下）。因此当软件测试的输入域的失效率很大的时候，选择RT算法具有更高的检测效率。



另一方面如果软件测试的输入域的失效率很小（本文make试验为0.99%，gzip实验为2.34%，bash实验为1.89%，grep实验0.87%）如果存在一个分区的失效率很大（本文make实验为36.36%），这时基于控制理论的一系列算法：DRT、MDRT、RDRT等测试效率远远高于RT算法。并且从图5-27（b）可以看出DRT、MDRT、RDRT相对于RT、RPT有很大的提高，MDRT、RDRT比DRT具有更高的检测效率。如果分区的失效率也比较低（gzip实验，grep实验，bash实验分区的失效率都很低）DRT、MDRT、RDRT在检测前几个缺陷时比RT效率更高，继续检测效率不如RT算法。在检测前几个缺陷时从图5-27（c）、5-27（d）、5-27（e）、5-28（a）、5-28（b）可以看出RDRT、MDRT比DRT具有更高的检测效率。

(2)影响性能的算法因素(算法参数的影响)

通过5.4.1章节实验数据的展示与讨论MDRT算法参数在[0.35，0.5]之间时具有较好的检测能力。DRT、RDRT算法在[0.001,0.1]之间具有较好的检测能力，并且如果初始概率分布为均等分布时，参数在0.001附近具有较好的检测效率，当初始概率分布为不均等分布时，参数在0.1附近具有较好的检测效果。当初始概率为非均匀分布时，一般情况下为了快速找到具有较强的检测能力的分区时，参数接近0.1可以快速调整分区对应的概率，以便尽早使检测能力较强的分区对应的概率变大。

(3)实验对象的影响(初始概率分布的影响)

通过5.4.2章节实验数据的展示与分析，当初始概率分布为不均等分布时，基于分区思想的一系列算法例如：RPT、DRT、MDRT、以及RDRT的测试效率普遍比初始概率分布为均等时要好。

当整体的输入域的失效率比较大时，如果分区内的测试用例多，那么该分区内的测试用例能够检测出缺陷的概率也就越大。5.4.2.1章节展示了输入域的失效率比较大的时候，不均等分布比均等分布明显具有更高的检测效率。

当整体的输入域的失效率比较低时，如果分区的测试用例多，那么能够检测出软件失效的测试用例更可能在该分区中。从5.4.2.3、5.4.2.4、5.4.2.5章节可以看出，不均等分布的检测效率高于均等分布。

从5.4.2.2章节可以看出，不均等分布的检测效率低于均等分布的检测效率。由于该实验能够检测出是失效的测试用例集中在一个含有测试用例数目相对很少的分区之内，因此该分区被选中的概率很低，所以测试效率不如均等分布。

(4)实验对象的影响(分区大小影响)

通过5.4.3章节数据的展示，当整个输入域的失效率较小时，一般情况下，应当尽可能的细致的划分分区。当整个输入域的失效率较低的时候，由于能够检测出软件中存在缺陷的测试用例具有聚簇性，分区越细致分配到每个分区的测试用一般就越少，就有可能使得分区的失效率变大。因此当软件中的输入域不容易检测出软件中存在缺陷时，划分分区的数目要多一点，即，若分区方式为功能分区，则将功能考虑的细致一些。

### 5.4.6 实验难点

在实验的过程中，由于是对Linux内核程序进行测试，因此需要详细了解每一个实验的项目信息。另外由于本文的分区方式为功能分区，因此需要知道每一个实验的每一个测试用例的测试功能，这一点需要花费比较多的时间。

在比较算法之间的优劣时，怎么尽可能减少外界因素对算法的影响，是一个很重要的问题。在以往的测试过程中，基于控制理论的分区测试，一般都是采用在被选择的分区中按照均等概率随机选择测试用例。计算机中的随机都是伪随机，这样有可能导致随机数的不同，对实验产生影响。因此为了消除这一点影响，本文在做随机测试时，一次试验下，有一个随机数种子产生大量的随机数。RT算法通过这一系列的随机数对应的测试用例进行测试。其它算法例如：RPT、DRT、MDRT、RDRT都是在选择分区之后，读取RT算法采用的随机数种子产生的序列，直到该序列对应的测试用例在选择的分区之中。通过这样做消除随机数可能造成的影响。

# 6 创新与不足

Myers[13]曾经这样讲：“随机测试可能是检测效率最低的测试方法”。由于随机测试没有利用当前测试用例信息以及以往的测试信息，在某些情况下会降低测试效率。本文基于这一点利用Markov链的性质，将当前执行的测试用例所在的分区，当做一个状态，利用状态转移矩阵，并根据当前测试用例执行结果动态调整状态转移矩阵。使得具有更高检测能力的分区被选择的概率增大。另外，每一个分区每次增大或者减少的力度应该是不同的。一般情况下，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。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缺陷，那么以后的测试过程中如果该分区的测试用例没有检测到缺陷应该减小该分区的幅度较小使得该分区仍具有较大的优势下次仍然被选中。本文将这一思想引入MDRT算法之中，即增大或减少某一分区被选中的概率时以当时分区被选中的概率相关

DRT算法是对RT以及RPT算法的一个改进，但是由于DRT算法每一次都根据当前测试用例的执行结果改变测试剖面，使得具有较高检测能力的分区不容易被找到。因此本文提出了基于奖惩机制的动态随机算法。

本文选取了真实的5个程序进行试验，为了尽可能模拟实际的开发，本文选取的测试对象都是真实存在的并且LOC都在5000行以上。通过上文的展示一般情况下，MDRT算法比RT算法具有更好的检测效率；RDRT算法比DRT与RT算法具有更高的检测效率。

但是，对于RDRT算法，虽然该算法加速了找到具有更高检测效率的分区，但是算法本身的惩罚因子，随着实验对象不同产生变化。本文5个实验的惩罚因子根据以往的测试经验设置的。这一点可以根据实际的软件测试中，各个分区的测试用例数目作为参考设置。如果在某一个分区中的测试用例检验到一定数目都无法检测到软件失效就认为该分区具有较低的检测能力。

# 7 参考文献

[1] R Hamlet, Random Testing[J]. Encyclopedia of Software Engineering, 1997, 970—978.

[2] D. Hamlet, R. Taylor, Partition Testing Dose Not Inspire Confidence[J]. IEEE Transactions on Software Engineering, 1990, 16(12):1402-1411.

[3] K. Y. Cai, H. Hu, C. H. Jiang. Random testing with dynamically updated test profile[C]//Proceedings of the 20th International Symposium On Software Reliability Engineering (ISSRE 2009). 2009: 1-2.

[4] P. E.Ammann, J. C. Knight. Data diversity: An approach to software fault tolerance[J]. IEEE Transactions on Computers, 1988, 37(4): 418-425.

[5] G. B. Finelli. NASA software failure characterization experiments[J]. Reliability Engineering & System Safety, 1991, 32(1): 155-169.

[6] K. Y. Cai, B. Gu, H. Hu, et al. Adaptive software testing with fixed-memory feedback[J]. Journal of Systems and Software, 2007, 80(8): 1328-1348.

[7] K. Y. Cai, T. Jing, C. G. Bai. Partition testing with dynamic partitioning[C]//29th Annual International Computer Software and Applications Conference (COMPSAC'05). IEEE, 2005, 2: 113-116.

[8] K. Y. Cai. Optimal software testing and adaptive software testing in the context of software cybernetics[J]. Information and Software Technology, 2002, 44(14): 841-855.

[9] L. Zhang, B. B. Yin, J. Lv, et al. A history-based dynamic random software testing[C]//Computer Software and Applications Conference Workshops (COMPSACW), 2014 IEEE 38th International. IEEE, 2014: 31-36.

[10] Chen T Y, Merkel R G, Eddy G, et al. Adaptive Random Testing Through Dynamic Partitioning[C]//QSIC. 2004: 79-86.

[11] Duran J W, Ntafos S C. An evaluation of random testing[J]. IEEE transactions on software engineering, 1984 (4): 438-444.

[12] Ostrand T J, Balcer M J. The category-partition method for specifying and generating fuctional tests[J]. Communications of the ACM, 1988, 31(6): 676-686.

[13] Myers G J, Sandler C, Badgett T. The art of software testing[M]. John Wiley & Sons, 1979.

[14] Chen T Y, Leung H, Mak I K. Adaptive random testing[M]//Advances in Computer Science-ASIAN 2004. Higher-Level Decision Making. Springer Berlin Heidelberg, 2004: 320-329.