面向数值程序的自适应随机测试复现技术——实验报告

1 概述

随机测试是一种朴素的生成测试数据的方法，当前已被大多数流行软件广泛作为测试手段。 然而，与被测程序的整个输入空间相比，人工生成的测试数据集的大小非常有限，因此随机测试生成的测试输入还不是真正均匀分布的。为了克服这个问题，研究者提出了一种称为自适应随机测试 (ART) 的改进方法，以产生更多分散的测试输入。实验结果表明，ART可以比传统随机测试更快地发现潜在故障。

“潮玩伴”小组探究了15篇论文中提出的数值程序ART算法，将这些算法以java语言实现，并基于原有框架进行重构与增补，实现了可进行错误空间模拟评估、真实程序评估的数值程序ART算法测试与评估框架。小组成员与负责内容如下：

姓名	学号	负责
葛家辰	201250105	框架重构、TPP_ART、DA
吴筱权	201250102	B_ART、IP_ART、
陈广华	201250103	SIMD_FSCS_ART、MD_RRT
邓僚僚	181250025	KDFC-ART、RRT、

项目创新点包含：

- 配置简单字段即可直接运行。
- 自行设计的输入空间划分辅助类。

2 文件结构

2.1 src/art

存放不同的ART算法，实现的算法均继承**AbstractART抽象类**的以下方法：

- main：单独执行该算法的入口方法。
- bestCandidate：描述如何找到下一个测试用例。
- testEfficiency：使用bestCandidate方法生成测试用例。

Abstract抽象类作为算法框架，提供了与测试框架交互的接口，描述了算法如何在测试框架的指导下运行。其中的重要方法有：

- runWithFaultZone：描述算法如何使用错误空间模型模拟的程序输入空间运行。
- runWithNumericProgram：描述算法如何使用真实数值程序的输入空间运行。

2.2 src/entry

存放运行该测试框架的两个“入口”：

- FaultZoneEntry：使用错误空间的测试入口。
- RealCodeEntry：使用真实数值程序的入口。

这两个入口均继承自**AbstractEntry抽象类**。AbstractEntry包含了一些两种入口均需要的方法与字段，例如：

- testEfficiency：测试算法效率的通用方法。
- storeResult：使用util包中的StoreResults类将测试结果输出至指定路径的文件中。

2.3 src/faultZone

存放三种错误空间模型，用于模拟输入空间：

- FaultZone_Block：错误空间被抽象为一个正方体（三维输入空间中）。
- FaultZone_Strip：错误空间被抽象为一个类圆柱体（三维输入空间中）。
- FaultZone_Point_Square：错误空间被抽象为多个小正方体（三维输入空间中）。

2.4 src/realCodes

存放实际的数值程序，我们使用其中的部分进行算法评估，包含两个文件夹：

- original：数值程序文件，包括不同参数、不同用途的数值程序。
- mutant：包含original文件夹中数值程序的变异程序，按原程序名称储存于对应的文件夹中。

我们还实现了两个辅助类，用于更简便地运行数值程序：

- RealCodesDriver：用于驱动数值程序运行。
- RealCodesRunner：**将数值程序在运行时转化为独立线程**，方便控制单次运行时间，发现无限循环等错误。

2.5 src/util

包含多种本项目需要的实用类：

- Dimension：表示单个维度的范围，一个维度对应一个输入参数。
- DomainBoundary：表示输入空间，由多个Dimension组成，即多个输入参数。

- Parameters：存放本项目统一的参数，比如单次运行次数、文件保存路径等。
- StoreResults：保存测试用例于指定的文件。
- TestCase：根据指定DomainBoundary生成的测试用例，还包含一些生成与比较测试用例需要的方法。
- Node：作用于部分算法的测试用例KD树节点类。
- PartitionTree：我们发现有相当一部分的ART算法使用了输入空间的划分技术，因此我们自行设计了**树状记录输入空间划分的PartitionTree类**，用于记录划分区域的范围。该类在实际使用中构成了一棵多叉树，树的父节点的范围是其子节点范围的并集，方便递归地查找与记录当前测试用例所在的划分。

2.6 out/faultZone

存放错误空间模型的测试结果文件。

2.7 out/realCode

存放真实数值程序的测试结果文件。

3 执行流程

3.1 错误空间测试

3.1.1 效率测试

我们调用testEfficiency函数，通过反射根据AbstractEntry中定义的算法名称字段新增对应实例，多次反复调用算法的testEfficiency函数，使算法执行bestCandidate方法生成规定数量的测试用例，并记录所用时长（默认为1000次），最终取时长平均值进行比较。

3.1.2 有效性测试

我们对算法在0.2、0.1、0.05三种错误率（错误率越大，错误空间越大）、所有三种模型的错误空间中依次执行并记录找到错误需要的测试用例数，在此过程中算法将不断执行bestCandidate方法。将测试用例数乘以错误率可以得到F-measure数值进行算法的相互比较。

3.2 真实数值程序测试

3.2.1 效率测试

同 3.1.1。

3.2.2 有效性测试

我们通过RealCodesDriver中的getDomainBoundary方法，获取对应名称数值程序的输入空间，将其作为初始化参数运行算法。在运行时，算法反复执行bestCandidate方法获取最佳候选测试用例，使用RealCodes包装过的RealCodesDriver的isCorrect方法，以反射方式调用、比较源数值程序与变异数值程序的返回值，若结果相同则继续生成新的测试用例进行测试，若**结果不同、用例生成次数过多、运行超时或报错**，我们认为变异程序分别发生了**结果错误、无法找到错误、无限循环、错误触发报错**的错误情况，停止算法循环并记录结果。

4 PartitionTree

我们设计了用于记录输入空间划分的PartitionTree类，部分算法通过该类记录、产生多层划分，极大地方便了划分类ART算法的工作。

PartitionTree的主要思想是，每个节点保存一种划分的范围。初始时整体的输入空间范围作为PartitionTree的树根节点，当程序要求树根节点进行划分时，该类将以给定的划分中心点为中心，在指定数量且随机的维度上将根节点“切割”成多个“立方体”子节点并保存在树根节点的子节点列表中，当二级节点需要划分时可以按如上方法再次分割生成多个三级节点。以多叉树结构保存划分可以减少寻找测试用例归属划分的时间支出，并以极高的效率存储划分的结果与过程。

重要方法介绍如下：

- partition：接收需要划分的维度的数量与划分中心点参数，“垂直”或“水平”地划分指定的节点。
- getPartitionedDomainBoundaries：partition调用的核心方法，将节点的DomainBoundary类字段切割成多个部分并返回，使得partition方法能够直接根据多个小范围的DomainBoundary类属性生成对应的子节点。
- getPartitionTreeNode：找到包含指定测试用例的最小划分。
- getLeaveTreeNodes：返回树中所有的叶节点，即当前输入空间真正的划分，交由算法做其他处理。

5 测试结果

5.1 结果文件结构

例：

```
4.251
1.118
2.572
UNFOUND
...
1.136
2.62
```

> UNFOUND
> UNFOUND
> EFFICIENCY: 0.164
> TOTAL TEST NUM: 51
> ERROR NOT FOUND TEST NUM: 23
> ERROR FOUND RATE: 0.5490196078431373

前多行的数字与UNFOUND代表算法作用于多个特定错误空间模型或数值程序，是否能找到错误（在规定尝试次数内未找到的标注UNFOUND），以及平均需要几次尝试找到错误（若以faultZone方式测试，则结果会预先乘以错误率，也即错误空间大小）。

EFFICIENCY数值指算法按指定次数生成测试用例的耗时数值，即效率测试数值。

TOTAL TEST NUM数值为算法作用于多少个错误空间模型或数值程序，也即EFFICIENCY前的行数。

ERROR NOT FOUND TEST NUM数值为UNFOUND次数。

ERROR FOUND RATE指UNFOUND次数占所有错误空间模型或数值程序的比例，即TOTAL TEST NUM数值除以ERROR NOT FOUND TEST NUM数值。

5.2 测试参数

对于效率测试，我们对每一种ART算法均进行错误率为0.05的Point_Square错误空间模型测试。每次效率测试进行1000轮，每轮生成100个测试用例，记录每轮所花费时间，取平均值作为结果。测试要求生成三维、范围为-1000至1000的测试用例。

对于有效性测试，我们使用两种方式进行测试：

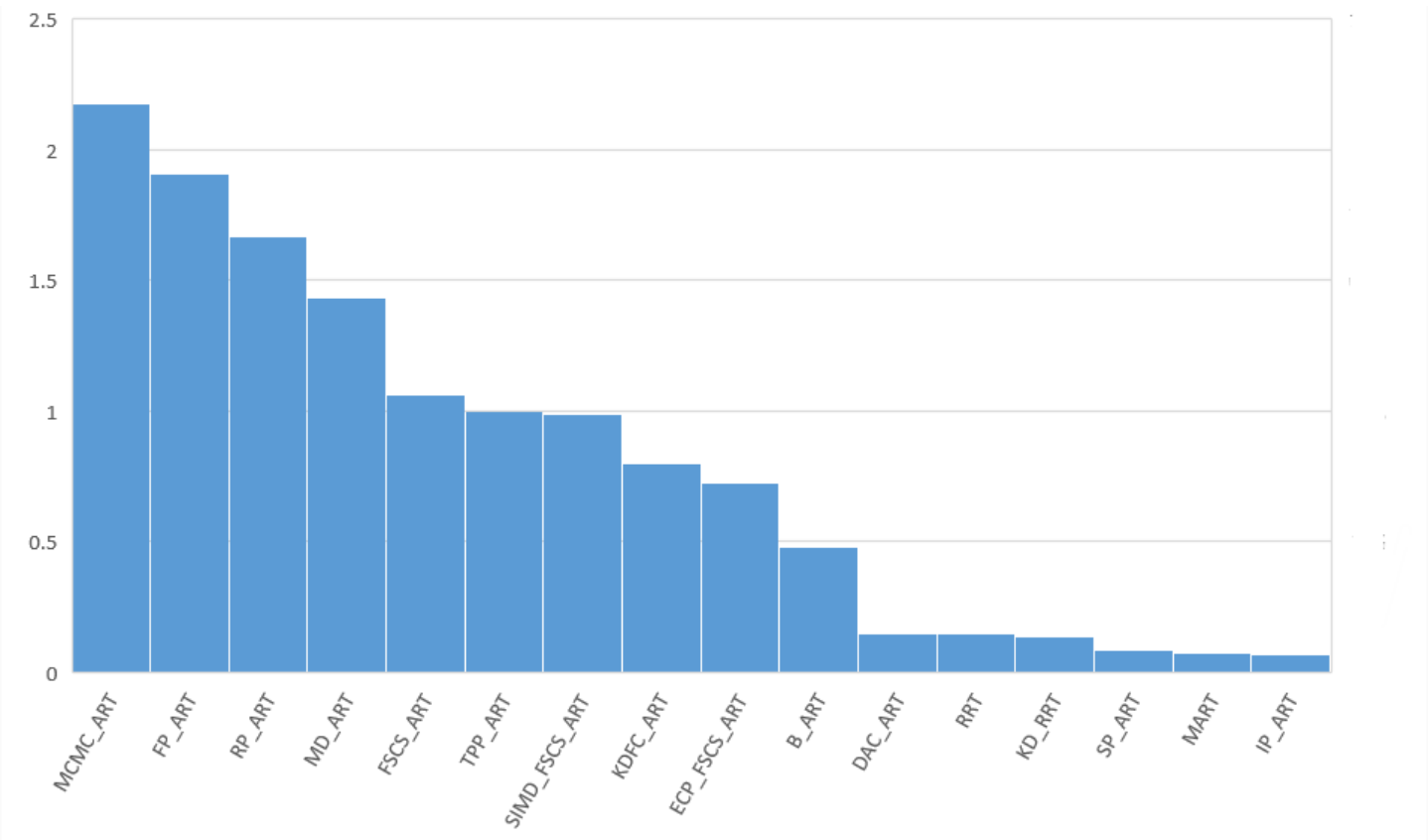
- **错误空间模型测试**：对每一种ART算法均进行九种错误空间模型测试：错误率（即错误空间大小）为0.2/0.1/0.05的Block/Strip/Point_Square模型测试。每个模型的测试进行1000轮，记录每轮找到错误需要的测试用例数，取平均值作为结果。测试要求生成三维、范围为-1000至1000的测试用例。特殊情况处理如下：
  - 测试用例数达到1000时，我们强制停止并标记该模型测试未找到错误（UNFOUND）。
- **真实数值程序测试**：由于框架完整度、时间等限制，我们仅使用realCodes文件夹内名称为Bessj、Remainder的真实程序及其变异程序进行测试。每个真实数值程序取与之对应的50个左右变异测试进行结果对比，每个对比进行1000轮，记录每轮触发原程序与编译程序返回值不同的情况需要的测试用例数，取平均值作为结果。输入空间根据不同数值程序的要求做不同处理（例如Bessj程序中某一参数小于2时会报错，因此该参数对应的输入维度下限为2）。特殊情况处理如下：
  - 测试用例数达到1000时，我们强制停止并标记该模型测试未找到错误（UNFOUND）。
  - 单次测试用例验证超过5s时，我们认为测试用例发现了无法停止的循环，强制结束测试并直接取本次测试用例数作为结果；
  - 单次测试用例验证变异程序报错时，我们认为测试用例触发了变异程序的错误报错，强制结束测试并直接取本次测试用例数作为结果。

5.3 测试步骤

1. 修改AbstractEntry类中的testingART字段为当前准备测试的ART算法名称。
2. 若进行真实数值程序测试，修改RealCodeEntry类中的testingCodeName为当前准备测试的真实数值程序名称。
3. 运行RealCodeEntry或FaultZoneEntry的main函数进行测试。测试结果将默认导出至out文件夹。
4. （可选）修改util/Parameters类中的字段修改测试控制相关参数。

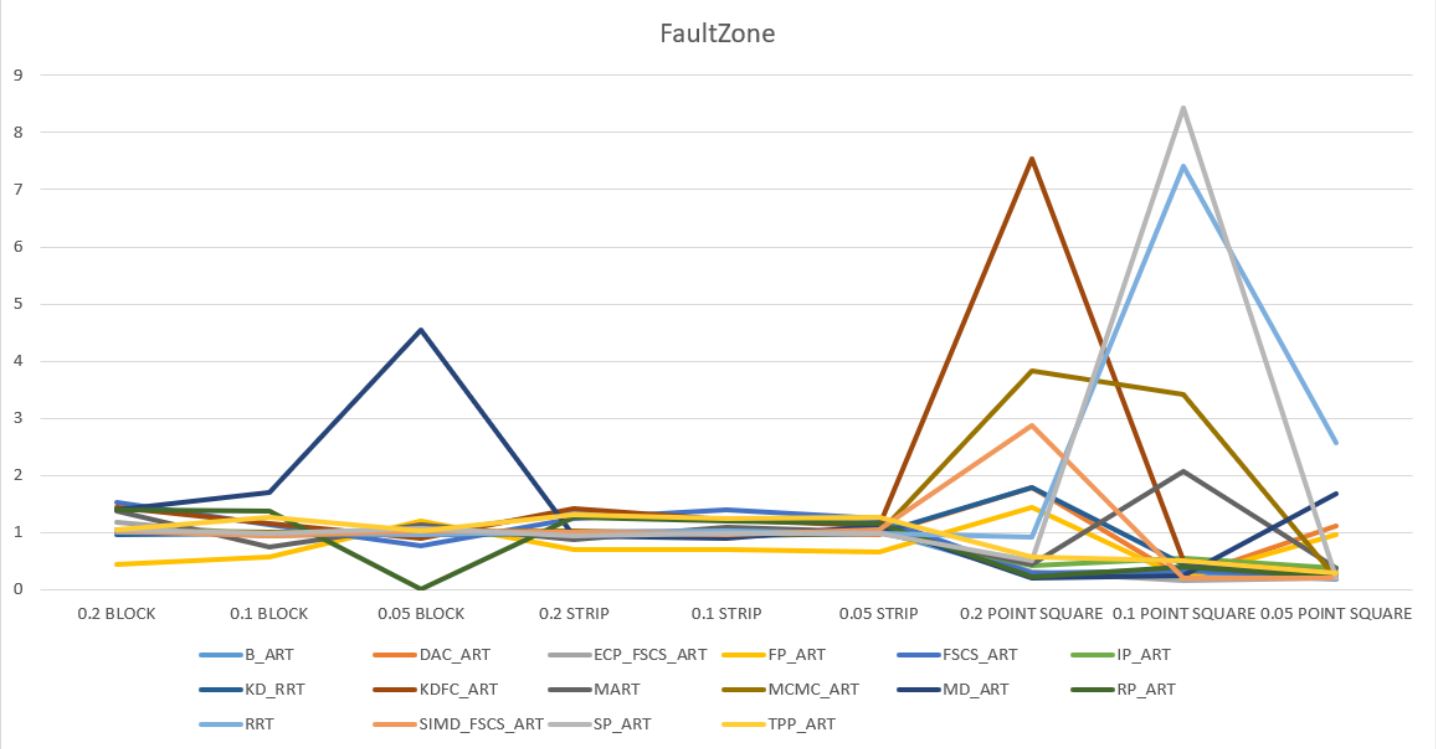
5.4 结果分析

5.4.1 效率



以FSCS_ART为基准，过半的算法的效率优于基准数值。我们根据算法的特征推测，效率劣于基准的可能原因有：算法过于随机、输入空间划分效率过低等。

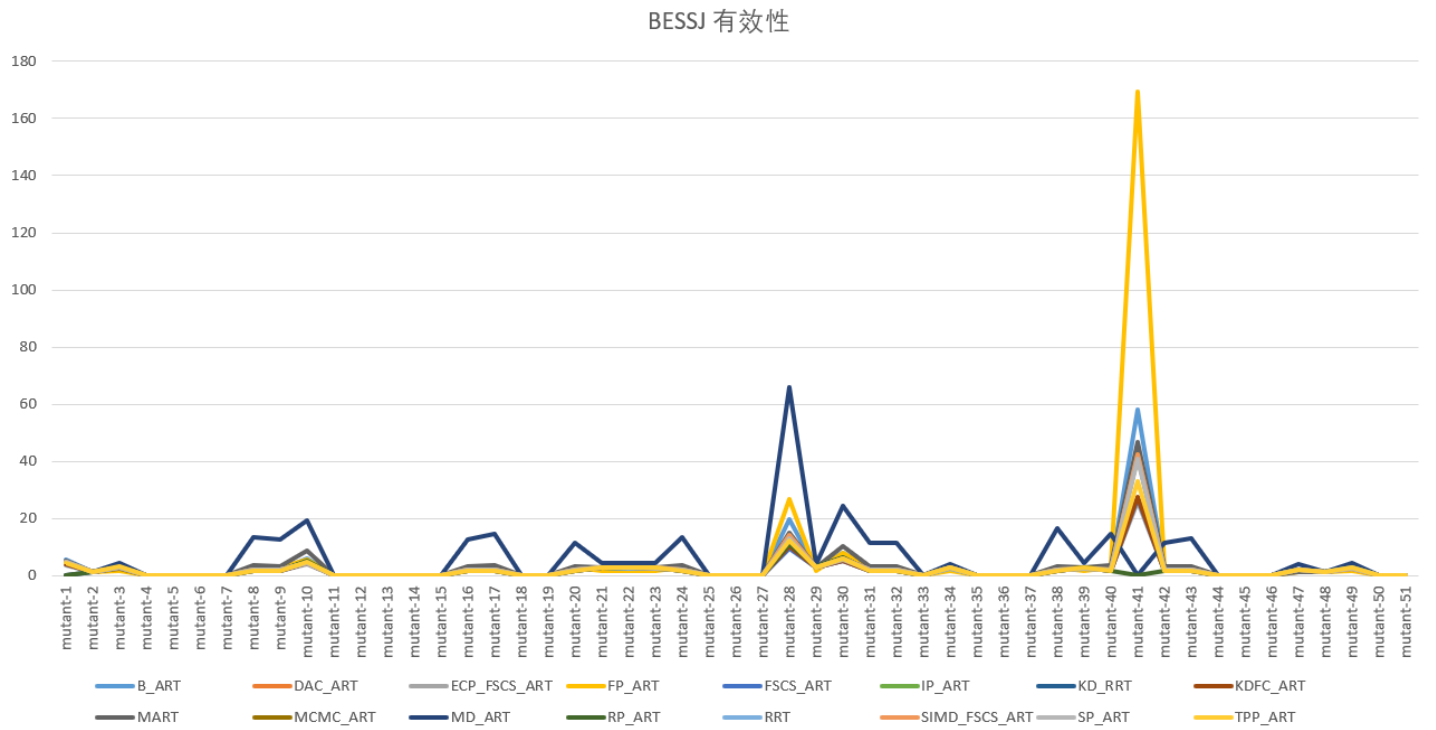
5.4.2 错误空间有效性



以FSCS_ART为基准，我们可以发现大部分算法在块形、条形错误空间模拟测试中与基准从差别不大甚至更优。有相当一部分的算法在点块错误空间模拟测试中产生了较大波动，我们推测这是由于点块错误空间在生成时错误块并非在输入空间中均匀分布，而是非重叠地分布，这会导致部分算法的特性无法较好地应用在其上。

5.4.3 真实数值程序有效性

文档仅以Bessj程序为例。



大部分算法都检出了最多的错误程序（28个），平均使用的测试用例数也大多低于5。部分算法在特定测试用例中会产生比其他算法更大的波动（如图中mutant-41的TPP_ART测试）。这可能是由于某些

算法对特定错误情况（如边界条件）难以触及，例如趋向输入空间中心的算法。

6 错误记录

- 某些真实数值程序的测试过程无法正常结束。
  - 解决方案：经过单步调试发现在进入发生变异错误的数值程序后执行陷入了循环。因此我们添加了继承自Runnable的RealCodesRunner类，包装RealCodesDriver类调用数值程序的方法，使用Future相关方法结束运行超过5s的测试进程。
- 某些真实数值程序的反射invoke方法报错无法进行。
  - 解决方案：发现报错内容为java.lang.reflect.InvocationTargetException，通过检索发现可能是由于内存消耗过大导致。由于发生错误时运行的真实数值程序的参数为int数组，而测试框架默认生成的数组大小为100，故尝试缩小该标准为10，该错误不再报告。
- 解决以上问题时仍报错：java.lang.reflect.InvocationTargetException。（未解决）
  - 分析：检索发现有两种可能：内存消耗过大或数值转换错误。经对比发现反射时数值转换与数值程序需求参数一致。考虑到程序报错前有很长一段时间无控制台输出，故可能有两种错误可能：反射invoke方法内部触发错误，或数值程序执行出现无法处理的错误。添加try-catch语句后无法解决问题。