基于划分的自适应随机测试综述

李志博 李清宝 于 磊 侯雪梅
(中国人民解放军战略支援部队信息工程大学 郑州 450001)
(数学工程与先进计算国家重点实验室 郑州 450001)

摘 要 随机测试是一种广泛应用于实践的基础测试方法。自适应随机测试(ART)是对随机测试的改进,其检错有效性优于随机测试。首先,分析了具有较高检错有效性但时间开销较大的经典ART算法;其次,重点综述了能降低时间开销的基于划分的ART算法,并对各种划分策略和测试用例生成算法进行了分析和对比;同时,分析了影响ART算法有效性的关键因素以及高维输入域空间中算法有效性低下的问题,梳理了算法有效性度量指标以及测试用例分布度量指标;最后,论述了ART算法中存在的问题及面临的挑战。

关键词 软件测试,随机测试,自适应随机测试,基于划分的自适应随机测试
中图法分类号 TP311.5 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2019.03.003

Survey on Adaptive Random Testing by Partitioning

LI Zhi-bo LI Qing-bao YU Lei HOU Xue-mei
(PLA Strategic Support Force Information Engineering University, Zhengzhou 450001, China)
(State Key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou 450001, China)

Abstract As a fundamental software testing technique, random testing (RT) has been widely used in practice. Adaptive random testing (ART), an enhancement of RT, performs better than original RT in terms of fault detection capability. Firstly, this paper analyzed the classical ART algorithm with high detection effectiveness and large time overhead. Secondly, it summarized the ART algorithms by partitioning to reduce the time cost, analyzed and compared various partition strategies and test case generation algorithms. Meanwhile, this paper analyzed the problems of the key factors affecting the effectiveness of ART algorithm and leading to low efficiency of algorithm in high dimensional input domain. Finally, it discussed the problems and challenges in the ART algorithm.

Keywords Software testing, Random testing(RT), Adaptive random testing(ART), Adaptive random testing by partitioning

1 引言

随着软件规模的不断扩大以及复杂性的不断提高,软件质量问题受到更多的关注。软件测试作为软件质量保证的重要活动,在软件开发中的地位愈发重要[1]。

被测程序(Software Under Test,SUT)通常有较大的输入域空间,因此选择一部分可以有效识别软件失效的测试输入作为测试用例(Test Case,TC)具有重要意义[2]。测试用例生成技术指导如何生成高效的测试用例,如符号执行的测试用例生成方法[3]、有限状态机的测试用例生成技术[4-6]、基于智能搜索的测试用例生成技术[7]等。其中,随机测试(Random Testing,RT)[8]是一种简单、易于实现的测试方法,它不涉及到复杂的软件需求或内部程序的结构信息,只需在输入域内随机地选择测试用例。随机测试没有利用被测软件的任何信息,且存在生成的测试用例冗余度高、覆盖度低以及测试用例生成的盲目性等缺点,曾被Myers[9]认为可能是最糟糕的测试方法。但由于随机测试具有简单、易于实现、成本低、无偏性、速度快等优点,其通常与其他测试方法结合使用,被广泛应用于各个领域的软件测试和可靠性评估中[10-11]。同时,所有测试方法生成的测试用例理论上几乎均可由随机测试生成,因此随机测试具有了发现一切错误的潜能[12-15]。

经实验研究发现[16-17],引发失效的输入易于聚集在连续的区域。依此结论,Chen等[18]提出了自适应随机测试(Adaptive Random Testing,ART)。与随机测试不利用任何信息随机产生测试用例相比,ART算法通过利用已执行成功测试用例的信息,以及引发失效的输入易于集中的特点,使生成

到稿日期:2018-01-14 返修日期:2018-03-10 本文受国家自然科学基金(61402525),国家社会科学基金(15AJG012),国家"核高基"科技重大专项(2013JH00103)资助。

李志博(1982-),女,博士生,讲师,主要研究方向为软件测试、测试结果正确性验证,E-mail:lizhibo1019@163.com;李清宝(1967-),男,博士,教授,主要研究方向为信息安全、软件测试,E-mail:qingbao_li@163.com(通信作者);于 磊(1973-),男,博士,副教授,主要研究方向为软件测试;侯雪梅(1981-),女,副教授,主要研究方向为软件测试。

的测试用例尽可能均匀地分布在输入域空间内。实验表明[19]，ART 的检错性能较 RT 有明显改善，发现第一个错误所需的测试用例数量更少。各种各样基于 ART 思想的算法随之被提出，例如，基于距离的 ART 算法(D-ART)[19]、基于限制的 ART 算法(RRT)[20]等。虽然 D-ART 和 RRT 的检错有效性非常好，但是它们的计算开销非常大。

子域测试(Subdomain Testing)同样是一种重要的软件测试方法。在子域测试[21]中，被测软件将输入域划分成若干不相交或重叠的区域，并从其中的一个或多个域中生成测试用例。如果划分的子域不相交，则称为划分测试(Partition Testing)。将自适应随机测试的思想与划分测试相结合，即基于划分的 ART 算法[22]。它通过划分多个输入子域来识别低密度测试用例区域，以便在这些区域内生成下一个测试用例，从而使整个输入域空间内的测试用例均匀分布。基于划分的 ART 算法减少了计算开销，提高了运行效率。大多数基于划分的 ART 算法的有效性优于 RT 算法，但不如 D-ART 和 RRT 等算法。

本文重点对各种基于划分的经典 ART 算法的实现原理做介绍并进行对比分析。

国内至今尚无有关自适应随机算法的综述，本文介绍了自适应随机测试算法的原理，分析了经典的 D-ART 算法和 RRT 算法，它们具有较高的检错有效性及较大的时间开销；重点综述了能降低时间开销的划分 ART 算法，并根据划分子域的策略与测试用例生成策略的不同详细对比分析了当前主流的划分 ART 算法；同时分析了影响 ART 算法有效性的关键因素以及高维输入域空间中算法有效性低下的问题，梳理了算法有效性度量指标以及测试用例分布度量指标；最后论述了 ART 算法中存在的问题及面临的挑战。

2 失效模式

Chan 于 1996 年对失效模式进行了定义[23]：失效模式是指引发程序产生失效的数据在输入域内分布的形状、大小等信息。失效包含 3 种典型的失效模式：点状失效模式、条状失效模式和块状失效模式。以二维输入域空间为例，图 1 给出引发 3 种失效的程序形态，图 2 给出其对应的失效模式的几何形状。

图 2(a)为块状失效模式，表示引起失效的输入聚集在输入域中的一块区域；图 2(b)为条状失效模式，表示引起失效的输入在输入域中是一个狭长的条状区域；图 2(c)为点状失效模式，表示引起失效的输入的分布比较分散，或呈现出多个小聚集区域。图 1 中的 3 个示例分别对应图 2 中的 3 种失效模式。

文献[23]中指出 3 种经典失效模式中，块状失效模式和条状失效模式是最常见的失效模式。

通过对失效模式进行研究发现：引发失效的输入往往形成连续的失效区域，不能引发失效的输入也应该形成一个连续的非失效区域[16,24]。换言之，当某测试用例不能引发软件程序中的失效时，那么与该 TC 相邻的测试用例亦有较小的概率能够触发软件程序中的失效。因此，基于该理论，提高随机测试发现失效的能力的最有效途径便是让测试用例尽可能更加广泛而均匀地分布于程序的整个输入域空间内[8]。

```
Example 1
Integer X,Y,Z;
Input X,Y;
IF(X>0 AND X<10)
  AND(Y>0 AND Y<10)
THEN
  Z=X;
//correct statement is Z=
  2 * X;
ELSE
  Z=2 * Y;
OUTPUT Z;
```

(a)块状失效代码

```
Example 2
Integer X,Y,Z;
Input X,Y;
IF(Y<=0)
//correct statement is IF
  (Y<0)
THEN
  Z=X-Y;
ELSE
  Z=X+Y;
OUTPUT Z;
```

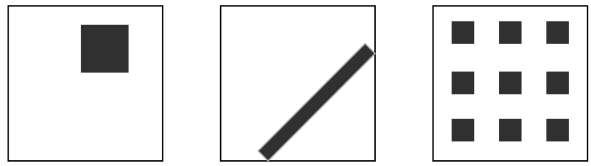
(b)条状失效代码

```
Example 3
Integer X,Y,Z;
Input X,Y;
IF(X mod 2=0 AND Y mod 2=0)
THEN
  Z=X-Y;
//correct statement is Z=X+Y;
ELSE
  Z=X * Y;
OUTPUT Z;
```

(c)点状失效代码

图 1 引发程序失效的代码示例图

Fig. 1 Failure-causing codes



(a)块状 (b)条状 (c)点状

图 2 块状、条状和点状失效模式示意图

Fig. 2 Block, strip and point failure patterns

3 自适应随机测试

基于失效的连续性原理，Chen 等[19]提出了一种增强型随机测试方法——自适应随机测试(ART)。ART 算法的思想是生成的测试用例应尽可能更加均匀地分布在整个输入域空间内。实验结果显示[25-26]，当失效域为连续型时，ART 的有效性相比 RT 有非常显著的提高；而当失效区域为离散型时，几乎没什么改进效果。

3.1 基于距离的自适应随机测试

基于距离的自适应随机测试(Distance-based Adaptive Random Testing，D-ART)算法是最早提出的 ART 算法，在 2004 年由 Chen 等第一次正式提出[18-19]。基于距离的自适应随机测试首先生成一定数目的候选测试用例，然后选择候选测试用例集中最好的候选测试用例作为下一个被执行的测试用例(计算候选测试用例中的每一个测试用例与已执行的测试用例间的距离，最后将其中最短距离最大的个体作为下一个被执行的测试用例)。D-ART 是有效性最好的 ART 测试方法之一，固定候选集(Fixed Sized Candidate Set ART，FSCS-ART)算法是 D-ART 中最经典的算法之一。

在 FSCS 算法中,首先随机生成一个测试用例,然后生成 k 个候选测试用例(候选集 $C=\{c_1,c_2,\cdots,c_k\}$),并计算每个候选测试用例 c_i($1\leqslant i\leqslant k$)与已执行测试用例集($E=\{E_1,E_2,\cdots,E_q\}$)内所有测试用例的最短距离,选择最短距离中最大的候选测试用例作为下一个执行的测试用例(如式(1)所示)。如果 TC 执行后未检测出错误,则将 TC 加入已执行测试用例集 E;再次执行上述生成下一个测试用例的过程,直至发现错误。

$$\min_{i=1}^{|E|} dist(c_j,E_i)\geqslant\min_{i=1}^{|E|} dist(c_m,E_i),\forall c_m\in C \tag{1}$$

其中,$dist(a,b)$表示 a 和 b 两点之间的距离。

但是该算法存在过多的距离计算,因此带来了大量的时间消耗,产生 n 个测试用例的算法时间复杂度为 $O(n^2)$;同时,处于输入域边界附近的点有更高的概率被选为下一个测试用例,也就是边界处更易于生成测试用例,即边界效应问题;而且随着输入域空间维度的增加,有效性明显降低,甚至不及 RT[18-19]。

3.2　基于限制的自适应随机测试

2002 年,Chan 等[20]提出基于限制的自适应随机测试(Restricted Random Testing,RRT)算法,它没有采用基于距离的自适应随机测试中的候选测试用例集的思想,而是定义每个已执行测试用例周围的排除域,在所有排除域之外随机生成下一个测试用例。即在所有已执行的测试用例周围划分一个半径为 $r=\sqrt{A\cdot R/(\pi\cdot n)}$ 的排除区域(其中,A 代表输入域的大小,n 代表用来测试程序但并没有引发程序失效的测试用例的数量,R 代表排除区域的覆盖率,即所有排除区域占输入域的比例),选择在所有排除区域外的第一个测试用例作为下一次执行的测试用例[27-28]。

Mayer 等曾专门对 RRT 算法的开销进行了实验分析[29],结果表明,选择 n 个测试用例所需时间为 $O(n^2\log n)$。从实验数据可以看出,RRT 算法发现错误的能力只略优于 FSCS,在相同维度的输入域空间中,不同失效率对 RRT 检错有效性的影响相对较小,但边界效应和庞大的运算开销却是 RRT 和 FSCS 都存在的问题。

4　基于划分的自适应随机测试

D-ART 算法和 RRT 算法虽然选择下一个测试用例的策略不同,但都是从整个输入域随机生成测试用例。假定输入域中的所有位置生成测试用例的概率相同,但当生成一部分测试用例后,输入域空间内测试用例的分布已变得不均匀,所以生成下一个测试用例的概率也应不相等了(测试用例稀疏的区域应该有更高的概率生成下一个测试用例)。

考虑到输入域生成测试用例的概率在不断变化,Chen 等于 2004 年提出了基于动态划分的 ART 算法[22,30](ART through Dynamic Partitioning,DP-ART)。该算法首先划分输入域,然后在测试用例稀疏区域生成下一个测试用例。这种方法通过将输入域划分为子域的方式区分不同测试用例密度的区域,使得测试用例稀疏区域有更大的概率产生测试用例,从而达到测试用例的均匀分布。

文献[22]提出了两种动态划分的 ART 算法:随机划分 ART 算法和对等划分 ART 算法。基于划分的 ART 虽然提高了算法的效率,但还是无法克服 RT 自身的盲目性,其有效性不及 DRT 和 RRT。

4.1　基于随机划分的 ART

基于随机划分的 ART(ART by Random Partition)[22]的思路是每次都以最近执行过的测试用例划分输入域空间,然后在最大的子域空间中再次随机生成下一个测试用例。重复此过程,直到发现缺陷为止。

如图 3 所示,以二维输入域空间为例,首先在输入域空间中随机生成一个测试用例 $tc1$(如图 3(a)所示),然后根据 $tc1$ 所在的位置,沿 x 轴方向和 y 轴方向划分输入域,得到 4 个子域,选择其中最大的子域 1 作为下一个测试用例生成域,随机生成测试用例 $tc2$(如图 3(b)所示),再沿 x 轴和 y 轴方向划分子域 1,此时当前输入域空间共被划分为 7 个子域,选择其中最大的子域再次随机生成下一个测试用例,直到发现缺陷或测试资源耗尽时算法终止。

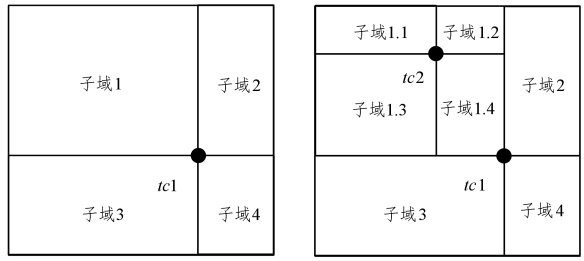


图 3　基于随机划分的 ART 算法

Fig. 3　ART by random partitioning

算法 1　随机划分 ART 算法

1. 设置候选测试集 C{{(i_{min},j_{min})(i_{max},j_{max})}}。
2. 从 C 中选择一个最大面积的域 T{(i_{min},j_{min})(i_{max},j_{max})},并将 T 从 C 中移除。如果最大面积域不止一个,则随机选择一个。
3. 从 T 中随机选择一个测试点(x,y),即 i≤x≤s 并且 j≤y≤t。
4. 如果测试点(x,y)触发失效,则发现 bug,算法停止。
5. 否则,将当前测试域{(i,j)(s,t)}划分成 4 个测试域{(i,j)(x,y)},{(i,y)(x,t)},{(x,j)(s,y)},{(x,y)(s,t)},并将 4 个域添加到 C 中。返回步骤 2。

4.2　基于对等划分的 ART

基于对等划分的 ART(ART by Bisection,ART-Bi)[22]算法每次将输入域划分为相等大小的子域,在没有测试用例的子域中随机产生测试用例,直至所有子域均包含测试用例时再次进行下一轮划分。随着层次划分的不断深入,生成的测试用例都分布在大小相同的子域中,从而实现测试用例的均匀分布。

如图 4 所示,以二维输入域空间为例,首先随机生成一个测试用例 $tc1$,若未检测到错误,则沿着边长较长的方向将输入域对等划分成两个子域,已执行测试用例 $tc1$ 所在子域不再生成测试用例,另一个子域中随机生成测试用例 $tc2$,若执行后未发现错误,则根据当前子域边长,将较长的边等分,得到 4 个子域,将没有包含已生成测试用例的子域作为测试用

例生成域，再次随机生成测试用例。依次类推，直到发现软件错误或资源耗尽时，算法终止。

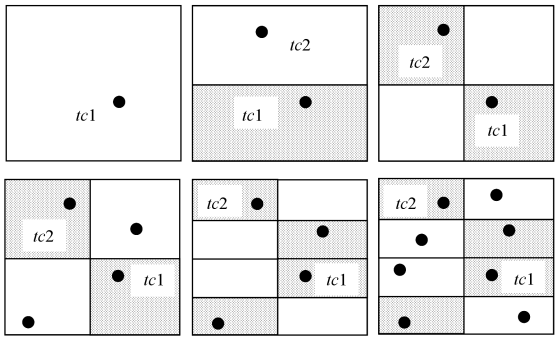


图 4 基于对等划分的 ART 算法

Fig. 4 ART by bisection

算法 2 对等划分 ART 算法

1. 假设一个待测试域为 $L_{untested}=\{(i_{min},j_{min})(i_{max},j_{max})\}$，初始为整个输入域。
2. 假设已测试域 L_{tested} 为空。
3. 如果 $L_{untested}$ 非空，从 $L_{untested}$ 中随机选择一个测试域 $T=\{(i_{min},j_{min})(i_{max},j_{max})\}$，并从 $L_{untested}$ 中删除 T；否则跳到步骤 7。
4. 在 T 中随机选择一个点(x,y)，$i\leqslant x\leqslant s$ 并且 $j\leqslant y\leqslant t$。
5. 如果测试点(x,y)触发失效，则发现缺陷，算法停止；否则，将测试域 T 加入 L_{tested}，转向步骤 3。
6. 设临时列表 L_{temp} 为空。
7. 对于 $L_{tested}\{(i,j)(s,t)\}$ 中的每个测试域，如果 $s-i\geqslant t-j$，令 $p:=\frac{s-i}{2}$，并将测试域划分成两个子域：$\{(i,j)(s,p)\}$，$\{(i,p)(s,t)\}$；否则，令 $q:=\frac{t-j}{2}$，将测试域划分成两个子域：$\{(i,j)(s,q)\}$，$\{(i,q)(s,t)\}$。
8. 对于这两个测试域，若测试域存在一个测试用例，则将该测试域放入 L_{temp}；否则将该测试域放入 $L_{untested}$ 中。
9. 将 L_{temp} 重命名为 L_{tested}，转向步骤 3。

Mayer[31]通过对基于对等划分的 ART 算法[22]进行改进，每次将输入域划分为等大小的子域后，下一个测试用例的选择都是从未生成测试用例的子域中生成，提出了基于受限子域的对等划分 ART 算法(ART by Bisection with Restriction，ART-Bi. Res)。改进的算法[31]再次缩小了选定的测试用例生成的子域，从而使生成的测试用例在子域的中心附近更集中，避免了对等划分算法[22]相邻子域内生成的测试用例可能会集中在公共边界附近的情况，使测试用例分布更加均匀。实验结果表明[31]，算法的有效性得到明显提高，与 D-ART 和 RRT 的有效性相近。

4.3 基于位置的 D-ART

2004 年，Chen 等在基于随机划分的 ART 的基础上，提出一种基于位置的 D-ART(ART by Localization，ART-Loc)[32]，使生成的测试用例更均匀。Localization 一方面限制测试用例从部分输入域选择，另一方面重新考虑下一个待生成的测试用例与测试用例生成域附近已执行测试用例的距离。该算法在随机划分 ART[22]的基础上考虑了待生成测试用例区域顶点位置测试用例已执行的情况，通过计算与该已执行的顶点测试用例的距离，确定测试用例生成区选出哪个候选测试用例作为下一个待执行测试用例；考虑了离生成测试用例区域最近的已执行测试用例的信息，提高了随机划分 ART 算法的有效性。

如图 5 所示，以二维输入域空间为例，将 D-ART 算法应用于基于位置的 ART 算法。首先通过执行随机划分算法，随机生成一个测试用例 $tc1$，并以此点划分 4 个子域，选择最大的子域 1 作为下一个测试用例的生成域。与随机划分算法直接在子域 1 中随机产生下一个测试用例再次迭代划分不同，该算法将作为划分点的测试用例 $tc1$ 作为已执行测试用例，在子域 1 中应用 D-ART 算法，生成多个候选测试用例($c1$，$c2$，$c3$)，再根据距离公式选择距离 $tc1$ 最远的候选测试用例 $c1$ 作为下一个测试用例 $tc2$，再次以测试用例 $tc2$ 作为划分点迭代划分子域，直至发现错误或测试资源耗尽时算法终止。

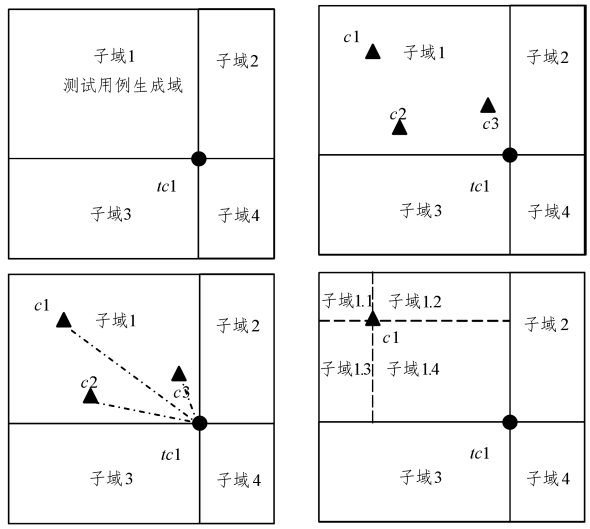


图 5 基于位置的 D-ART 算法

Fig. 5 D-ART algorithm based on localization

4.4 扩大输入域的 ART

2006 年，Mayer 等通过分析 D-ART 算法、RRT 算法生成的测试用例的空间分布，提出了扩大输入域的 ART 算法(ART with Enlarged Input Domain，ED-ART)[33]。对于固定的失效率，失效域的边长(每个维度上的长度)随着维度的增大而变长。算法原理为：增大原输入域空间为 D'，在 D' 中使用 ART 算法生成测试用例，但只执行原输入域空间内的测试用例(如图 6(a)所示)。这样，使得 ART 算法边界效应集中的边界测试用例都在扩大的非原输入域内产生，从而实现原输入域内测试用例的均匀分布。ED-ART 通过缓解测试用例生成的边界效应，提高了原始 ART 算法的检错有效性。

实验结果显示，在同一维度下，检错有效性(F-measure 值)在不同失效率下保持不变(为常量)。

如图 6(b)所示，假定在二维输入域空间中，原输入域 D 扩展为 D'，在扩展输入域 D' 中使用 D-ART 算法生成测试用例。ED-ART 与 D-ART 算法的不同之处是，在扩展输入域

D'中,所有满足最短距离的候选测试用例都会被放入成功测试用例集,只有落入原输入域 D 的测试用例才会执行。如图 6 所示,当前已生成测试用例 $T\{tc1,tc2,tc3\}$(其中,$tc2$ 不在原输入域 D 中,但属于已成功生成的测试用例,未被执行,会参与以后测试用例距离的比较)随机生成候选测试用例 $C\{c1,c2,c3\}$,将每个候选测试用例 ci 与 T 中的测试用例进行距离计算,选择最好的测试用例作为下一个测试用例。以此迭代进行,直至发现错误或测试资源耗尽时算法终止。

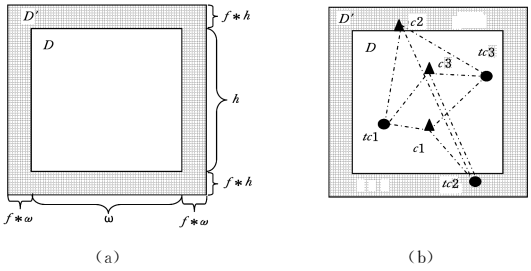


图 6　基于扩展输入域的 D-ART 算法

Fig. 6　D-ART with enlarged input domain

2012 年,结合 ART-Loc 算法[32]和 ED-ART 算法[33]的思想,Sabor 等提出以距离计算和扩大的输入域空间为基础实现 ART[34](ART Through Dynamic Partitioning by Localization with Restriction and Enlarged Input Domain, ED-ART-DP. Loc. Res),该算法解决了大部分 ART 都存在的边界效应问题。仿真结果显示,其检错能力通常优于基于位置的 D-ART 算法[32]。

4.5　基于迭代划分的 ART 算法

2006 年,Chen 等提出基于迭代划分的 ART 算法(ART Through Iterative Partitioning, IP-ART)[35]。算法思想为:随机生成首个测试用例,若未能检测出错误,则使用对等划分的方式划分原输入域($p\times p$),将划分后的子域分成 3 类:包含执行测试用例的域(occupied cells),不包含任何测试用例但与包含执行测试用例的域相邻的域(adjacent cells),其余的输入域为测试用例生成域(candidate cells),此域为随机测试用例生成域。若 candidate cells 不存在,则舍弃原划分域($p\times p$),重新将输入域等分为$(p+1)\times(p+1)$个子域,此情况下需将先前生成的测试用例映射到新划分的子域中,并将新划分的子域分为 3 类,若不存在测试用例生成域,则再次进行对等划分。实验显示,该算法的检错能力相比动态划分有所提高(在保持检错有效性的情况下,能减少计算开销)。

其中,将先前生成的所有测试用例映射到当前的新划分子域中会带来计算开销。假定输入域空间的维度为 k,检测出首个失效所需测试用例的数量为 n,则算法的时间复杂度为:$O(3^{k+1}\cdot n^{1+1/k})$。可以看出,随着维度 k 的增大,3^{k+1}会增大,从而导致计算开销增大[38]。

如图 7 所示,假定在二维输入域空间中,随机生成首个测试用例 $tc1$,当前输入域为包含执行测试用例的域,无测试用例生成域。令 $p=2$,将输入域划分成 2 * 2 的子域,将 $tc1$ 映射到子域中,包含 1 个 occupied cells、3 个 adjacent cells,无 candidate cells。令 $p=3$,将输入域划分成 3 * 3 的子域,将 $tc1$ 映射到子域中,包含 1 个 occupied cells、3 个 adjacent cells 和 5 个 candidate cells;在 candidate cells 中随机生成下一个测试用例 $tc2$,更新对应的子域类型,当前为 2 个 occupied cells、5 个 adjacent cells 和 2 个 candidate cells。在 candidate cells 中随机生成下一个测试用例,以此类推,直到发现错误或测试资源耗尽时算法终止。

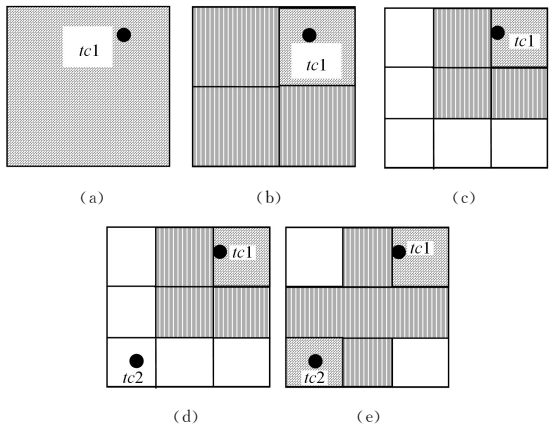


图 7　基于迭代划分的 ART 算法

Fig. 7　ART through iterative partitioning

Mayer 分析,IP-ART 算法每次迭代都是将原划分域($p\times p$)舍弃,重新将输入域等分为$(p+1)\times(p+1)$个子域,不能根据输入域的形状调整划分的子域,并对此提出了非归一化的迭代划分算法(Non-Normalized Adaptive Random Testing through Iterative Partitioning, NIP-ART)[36]:当舍弃原划分域($p\times p$)后,根据当前子域的高度和宽度,选择较长者进行划分,若第一维度的边较长,则重新划分输入域为$(p+1)\times p$ 个子域,否则为 $p\times(p+1)$个子域。IP-ART 算法的有效性与输入域的形状有关,各维度上尺寸比值的差别越大,有效性越低。改进的 NIP-ART 在每轮次迭代中只增加一个维度上的划分数量,尽可能地维持子域空间的边长相近,减慢了每轮迭代时划分子域数量的增长速度,使得 NIP-ART 算法的有效性对输入域的形状不敏感,保持了较高的有效性[36]。

4.6　基于平衡的 ART 算法

2007 年,Chen 等为了提高高维输入域空间中 ART 的有效性,提出了基于平衡的 ART 算法(ART by Balancing, ART-Bal)[37]。在输入域中使用 D-ART 生成候选测试用例,距离的判断改为选择与已执行测试用例形成的质心与输入域质心最近的一个候选测试用例。当输入域内存在两个及以上已执行测试用例时,重新划分输入域。在每个子域中,再次根据质心最近原则选择候选测试用例。该算法对高维空间下的检错能力有所改进。

在 k 维输入空间中,令 $S=\{s_1,s_2,\cdots,s_m\}$为测试用例集,其中 $m\geqslant 1$ 且 $s_i=(s_{i_1},s_{i_2},\cdots,s_{i_k})(1\leqslant i\leqslant m)$。$S$ 的质心 $R=(r_1,r_2,\cdots,r_k)$为:

$$r_j=\frac{1}{m}\sum_{i=1}^{m}s_{i_j},1\leqslant j\leqslant k \tag{2}$$

输入域也被认为是一个密度均匀的系统,其质心位于输入域的中心。如果测试用例均匀分布,那么测试用例的质心

应该接近于输入域的质心。

若在一个输入域内使用平衡策略，可能会指导测试用例的生成在质心点过于集中。为避免此类情况，ART-Bal 算法采用划分多个输入域空间的方法，每次使用平衡策略时，仅限于子输入域空间内的两个点（测试用例），超出两个域，则再次划分子输入域空间，从而避免平衡策略导致的测试用例中心区域密集的问题。

如图 8 所示，假定在二维输入域空间中，划分域中的理想质心位置在图中用“☺”表示，原输入域空间中随机生成测试用例 $tc1$，执行 $tc1$ 时未发现软件错误。随机生成候选测试用例集 $C=\{c1,c2,c3\}$，计算候选测试用例集 C 中每个测试用例 ci 与子域中已执行测试用例 $tc1$ 的质心点，选取与该域中理想质心点距离最近的候选测试用例 $c2$ 作为下一个测试用例 $tc2$，执行 $tc2$，若未发现错误，此时该域中已执行测试用例数已达到 2 个，需重新划分输入域空间为 2 个子域，每个子域中各有一个已执行测试用例，计算每个子域中的理想质心点。在每个子域中继续依此算法生成测试用例并执行，直至发现错误或资源耗尽时算法终止。

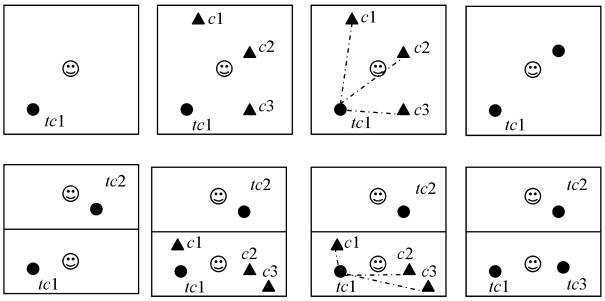


图 8 基于平衡的 ART 算法

Fig. 8 ART by balancing

4.7 基于静态划分的 ART 算法

2015 年，基于静态划分的 ART 算法（ART By Static Partitioning，ART-SP）[38] 被提出，其将输入域静态划分成多个子域，通过标注不同的区域，使测试用例生成域具有不同的概率。按测试用例的分布情况对不同子域进行标注区分，分别为白色格子、点状格子、斜线格子和竖线格子。点状格子域（不必产生测试用例的区域）表示包含一个已经执行过的测试用例；白色区域（测试用例生成的高概率区域）表示不包含任何已执行测试用例，并且不与点状格子相邻；斜线格子区域（测试用例的生成概率小于白色）表示不包含任何已执行测试用例但仅与一个点状格子域相邻；竖线格子域（测试用例的生成概率更低）表示与两个或多个点状格子域区域相邻。如果所有子域均为点状格子域，并且还未检测到失效，则重新将所有子域标成白色，算法重新开始。

如图 9 所示，以二维输入域空间为例，将输入域空间划分成 5×5 的子域，所有 25 个单元在初始时均为白格子（测试用例生成概率最高的区域），随机选择其中一个单元格，并在其中随机生成一个测试用例，同时将该单元格标为点状格子，将与该单元格相邻的所有单元格（8 个）标为斜线格子，执行测试用例 $tc1$，若未发现错误，则在当前剩余的白色单元格（16 个）中随机选择一个单元格，并在其中随机生成下一个测试用例 $tc2$，同时将该单元格标为点状格子，更新 $tc2$ 相邻单元格为斜线格子（初始为白色）或竖线格子（初始为斜线格子）。以此类推，在剩余白色单元格中继续生成下一个测试用例，如果无白色单元格，则将斜线单元格作为测试用例生成域，若无斜线单元格，则选竖线单元格作为测试用例生成域，直至 5×5 的每个子域包含一个测试用例，若未检测到错误，则清空 5×5 的子域，算法重新开始，直至发现错误或资源耗尽时算法终止。

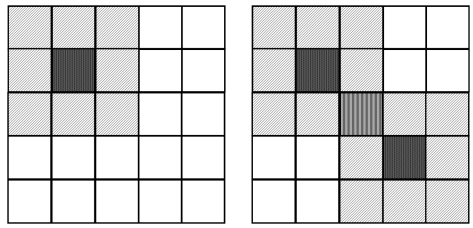


图 9 基于静态划分的 ART 算法

Fig. 9 ART algorithm by static partitioning

基于静态划分的 ART 算法简单，运行效率高，但其有效性低[38]。

假定检测到第一个错误所需生成的测试用例数量为 n，由于不需要与已执行测试用例进行距离计算，因此时间复杂度为 $O(n)$[38]。

因为每次只保证 P^n（n 为输入域空间的维度）个测试用例分布均匀，下一轮迭代时，对以前生成的测试用例采用遗忘策略。由于无法保证每轮次迭代内的 P^n 个测试用例与不同轮次迭代内的测试用例呈均匀分布，因此其有效性较低。静态划分 ART 算法的有效性远不及 D-ART 算法，且高失效率时的有效性高于低失效率时的有效性（与 D-ART 相反）。

4.8 各种基于划分的 ART 算法的对比

经典的 D-ART 与 RRT 算法的有效性几乎接近理论上限——随机测试有效性的 50%，但计算开销均较大，尤其是在低失效率或高维度输入域空间时。

基于迭代的 ART 算法、基于平衡的 ART 算法的有效性较高，接近 D-ART 与 RRT 算法的有效性。

动态划分的 ART 算法（包括对等划分算法和随机划分算法）的有效性略低，而基于位置的 ART 算法针对随机划分算法进行了改进，提高了有效性。

基于扩大输入域的 ART 算法，针对 D-ART 算法易于在边界附近产生测试用例而导致边界附近测试用例生成密度较高的边界效应问题，通过扩大输入域空间，将边界处密集的测试用例划分到输入域空间外部，同时保证了 ART 算法均匀生成测试用例的特性，因此检错有效性很高（高于原 ART 算法、D-ART 或 RRT 算法）。

基于静态划分的算法的有效性更低，由于每轮迭代在 $p\times p$ 单元中各生成一个测试用例后重新清空所有已执行测试用例，因此算法简单，但每次最多只参考最近的 $p\times p$ 个已执行测试用例，所以有效性较低，效率较高。

基于优先级的迭代划分测试方法的效率和有效性均较高，但由于每次测试用例来源于子域中心点，因此测试用例的随机性不强。

表 1 对各种基于划分的 ART 算法进行了详细对比。

表 1　各种基于划分的 ART 算法的 F-ratio 对比

Table 1　F-ratio comparison of various ART algorithms by partitioning

ART	Year	Ref.	TC Generation	SubDomain		F-ratio(2-domain, block pattern, 0.001)
				num	*size*	
ART-RP	2004	[22]	random	increase	unequal	0.795
ART-Bi	2004	[22]	random	increase	equal	0.740
ART-Loc	2004	[32]	D-ART RRT	increase	unequal	0.695 0.692
ART-Bi. Res	2005	[31]	random	increase	equal	0.679
ED-ART	2006	[33]	D-ART RRT	fixed	enlarged	0.581
IP-ART	2006	[35]	random	increase	equal	0.610
ART-Bal	2007	[37]	random	increase	equal	0.689
ED-ART-DP. Loc. Res	2013	[34]	D-ART	increase	unequal	0.657
ART-SP	2015	[38]	random	fixed	equal	0.816

表 1 中的 TC Generation 代表基于划分的算法在选定的测试用例生成子域空间中生成测试用例的方法,一般有随机生成(random)、D-ART 或 RRT 等几种生成方式(IPT-PS 较特殊,其在子域中心点生成测试用例)。SubDomain 表示划分的子域信息,包括两部分,一个是 *num*,表示子域的数量,在算法执行过程中,划分的子域数量不断增多(increase)或者是子域数量保持不变(fixed);另一个是 *size*,表示子域的大小是否相同,equal 表示划分为等大小子域;F-ratio 为算法有效性度量指标(见表 1),在仿真环境为二维输入域空间块状失效模式下,失效率为 0.001 时的 F-ratio 值。由于算法实现上存在一些差异,外加仿真环境的影响,F-ratio 值会存在些许误差,尤其是 F-ratio 值相近的一些算法。

5　ART 算法的关键问题分析

5.1　影响 ART 算法检错有效性的因素

在什么情况下 ART 算法的检错有效性优于 RT 算法值得研究。文献[39-40]研究了引发失效的输入数据的分布与 ART 算法检错有效性之间的关系。仿真实验表明,输入域的维度、失效率、失效域的形状和紧致度均影响 ART 算法的有效性。

(1)输入域的维度

输入域空间维度越大,ART 算法的检错有效性越低。

(2)失效率

失效率越低,ART 算法的检错有效性越高。

(3)失效域的紧致度

不同失效域形状的紧致度不同,代表了失效域的聚集程度不同。关于形状紧致度的度量研究有很多[41-43],其中一种方法将失效域面积(体积)与具有相同边长的圆(球)的比值作为紧致度。在二维空间中,紧致度$=\frac{4A\pi}{P^2}$;在三维空间中,紧致度$=\frac{6A\sqrt{\pi}}{\sqrt{P^3}}$。其中,$A$ 表示失效域的面积或体积,P 表示失效域的周长或表面积。紧致度的取值范围为(0,1),值越大越紧致。

失效域越紧致,ART 算法的检错有效性越高。

(4)失效域的数量(等大小失效域)

随着失效域个数的增加,ART 的有效性降低。在二维或三维空间中,当失效域个数增加到 20 左右后,ART 的有效性达到稳定值,几乎与随机测试效果相同。

(5)(如果存在)主失效域的大小

主失效域越大,ART 算法的有效性越高。这说明存在一个明显的主失效域比失效域的数量(第(4)点提到的)对 ART 算法的有效性影响大。

另外,输入域的大小、输入域的类型对 ART 算法的 F-measure 值几乎没有影响[44]。

5.2　高维空间有效性低下问题的分析

很多 ART 算法在低维度空间的有效性与 D-ART 相似,且在高维输入域空间时与 D-ART 算法一样会遇到有效性急速下降的问题。

通过分析得出,高维度输入域空间下 ART 算法有效性普遍低下的原因[37]如下:在高维空间中,假设输入域空间为边长为 1 的超立方体,失效域同样也是超立方体。假设 k 为输入域的维度,θ 为失效率,失效域的边长 ω 随着维度的增加而增大(如表 2 所列),$\omega=\sqrt[k]{\theta}$。在高维输入域空间中,从输入域中部产生的测试用例比从整个输入域中产生的测试用例有更高的检测失效的概率[40],尤其在高失效率时更为明显。

表 2　k 维输入域空间的边长为 1 时不同失效率下超立方体失效模式的边长取值

Table 2　Width of hypercube failure pattern in k dimensional hypercube input domain when size is 1

ω	Failure rate θ			
	0.01	0.005	0.001	0.0005
$k=2$	0.100	0.071	0.032	0.022
$k=3$	0.215	0.171	0.100	0.079
$k=4$	0.316	0.266	0.178	0.150
$k=5$	0.398	0.347	0.251	0.219
$k=6$	0.464	0.414	0.316	0.282
$k=7$	0.518	0.469	0.373	0.338

6　ART 算法的相关度量指标

6.1　有效性度量指标

测试用例生成算法的有效性是指,算法生成的测试用例集可以检测到更多的失效,或有更高的概率检测到一个失效,或检测到第一个失效所用的测试用例更少。

测试算法有效性的常用度量方法有 P-measure,E-measure和 F-measure。P-measure 是指对于指定测试用例集合,找到至少一个导致程序失效的测试用例的概率;E-measure

是指对于一个测试用例集合,所期望找到的导致程序失效的测试用例的数量。它们都曾用于划分测试、随机测试的分析[21,45-46]。后来,Chen 等[23]提出了一种新的有效性度量方法 F-measure,它表示找到第一个导致程序失效的测试用例时所使用的测试用例的数量。文献[19]中使用 F-measure 对 ART 算法与 RT 算法的有效性进行了对比,后续 ART 算法的改进[20]也大都采用此度量指标。作为评价算法有效性的标准,P-measure 值越大或者 E-measure 值越大,则说明算法的有效性越好;而 F-measure 值越小,说明算法的有效性越好。与前两种度量方法相比,F-measure 所提供的测试结果更能直观、自然地显示出算法效率的高低;而且,在具体实践中当一个测试用例导致程序失效时,测试也就停止了。因此,不论从直观感受还是从实用的观点来看,F-measure 都比前两种度量方法更具有吸引力。ART 算法一般采用 F-measure 对算法进行评估。

Merkel[47]对单一失效域(失效域大小和形状已知,位置未知)的软件测试有效性的上限进行了理论分析。文中指出最优的测试用例选择方法的 F-measure 比随机测试的 F-measure 的降低幅度不会超过 50%。Chen 等[44]将该上限推广到任意不同数量的失效域中。

在程序测试过程中,能够使程序失效的输入被称为引发失效输入(Failure-causing Inputs)。可以用来产生测试用例(一个或一组输入数据)的数据空间的取值范围被称为输入域(Inputs Domain)。引发失效的输入组成的输入域被称为失效域(Failure Domain)。用 D 表示程序的输入域,d 表示输入域 D 的大小,m 表示失效域的大小,由于 m 所代表的输入域包含在 D 中,因此 $d \geqslant m \geqslant 0$。定义失效域占整个输入域的比例为失效率 θ,用公式表示为:

$$\theta = \frac{m}{d} \times 100\% \tag{3}$$

随机测试算法在实验中的 F-measure 期望值[20] F_{RT} 为:

$$F_{RT} = \frac{1}{\theta} \tag{4}$$

实验统计得到 ART 算法的 F-measure 期望值 F_{ART},用 F_{ratio} 来表示 ART 算法相比 RT 有效性提高的比例,即 Fratio 取值越小,ART 算法相对 RT 算法的检错有效性改进越好。

$$F_{ratio} = \frac{F_{ART}}{F_{RT}} \tag{5}$$

假设测试用例集的大小为 n,则随机测试算法的 E-measure 值[47]为:

$$E_{RT} = n\theta \tag{6}$$

随机测试算法的 P-measure[47]为:

$$P_{RT} = 1-(1-\theta)^n \tag{7}$$

6.2 测试用例分布的度量指标

ART 的原则是测试用例选择的随机性与测试用例的均匀分布。测试用例分布的均匀程度包括以下具体度量指标。

假设 a,b 为集合 E 中的元素,$dist(a,b)$ 表示 a,b 之间的距离,$neighbour(p,E)$ 表示集合 E 中距离 p 最近的邻居。

(1)差异 Discrepancy[40]

$$M_{Discrepancy} = \max_{i=0,\cdots,m} \left| \frac{|E_i|}{|E|} - \frac{|D_i|}{|D|} \right| \tag{8}$$

输入域 D 被分成 m 个子域 $D_1, D_2, \cdots, D_m$,每个子域上对应的测试用例集为 $E_1, E_2, \cdots, E_m$。

$M_{Discrepancy}$ 表示区域 D 内是否具有等密度的点。如果 $M_{Discrepancy}$ 的取值接近 0,则可认为集合 E 是等分布的。

(2)分散 Dispersion[40]

$$M_{Dispersion} = \max_{i=1,\cdots,|E|} dist(e_i, neighbour(e_i, E)) \tag{9}$$

其中,$e_i \in E$。

$M_{Dispersion}$ 表示集合 E 中任意点具有的最大球形(三维中)空间(只有该点自己)。$M_{Dispersion}$ 的值越小,则可认为集合 E 中各点间具有各加近似相等的距离,即分布更均匀。

(3)边界/中心测试用例比率[40]

$$M_{Edge:Centre} = \frac{|E_{edge}|}{|E_{centre}|} \tag{10}$$

E_{edge} 与 E_{centre} 分别表示位于 D_{edge} 与 D_{centre} 内两个不相交的子集,且 $E = E_{edge} \cup E_{centre}$。

$M_{Edge:Centre}$ 取值接近于 1 时,表示没有边界或中心区域的偏好,则更利于集合 E 的均匀分布。

(4)最近邻居的平均距离[40]

$$M_{AveDist} = \frac{\sum_{i=1,\cdots,|E|} dist(e_i, neighbour(e_i, E))}{|E|} \tag{11}$$

其中,$e_i \in E$。

$M_{AveDist}$ 表示最近邻居点的平均距离。($M_{Dispersion} - M_{AveDist}$)的值越小,表示集合 E 的分布越均匀。

虽然使用上述测试用例分布度量指标可以精确地确定测试用例的分布情况,但计算复杂,很多文献[36,48-49]都采用空间分布(Spatial Distribution)来对 ART 算法的测试用例分布情况进行分析。

为了分析算法生成测试用例的空间分布情况,通常不考虑输入域空间内的失效域,即假设输入域空间内无失效域(失效率为 0),生成测试用例的算法不会因为检测到错误而终止。在每次测试用例生成过程中,记录第 n 个测试用例生成的空间位置($n=1,5,10,20,50,100,\cdots$)。分析不同测试用例序数的空间分布是否达到测试用例的均匀分布。

7 ART 算法的机遇和挑战

2011 年,Arcuri 等[50]发表了一篇讨论适应性随机测试效果的论文。Arcuri 等开展了一项大规模的实验,其中涉及 3727 个变异程序和近 10 万亿个测试用例。实验结果表明,在实际应用中,ART 的效率非常低,即使是对于一些琐碎的小问题,ART 的效率也比随机测试的效率更低。若要将 ART 投入实际应用,必须解决 ART 目前所面临的三大难题:不可接受的时间消耗,巨大的内存消耗,以及测试结果的正确性判定也就是自动的 Oracle 方法。

综合 Arcuri 的分析结论,将当前 ART 算法所面临的机遇与挑战分为以下几个方面。

(1)现有的一些 ART 实现方法中大量计算带来的时间开销[50]。

经典 ART 算法(如 D-ART 算法、RRT 算法等)虽然在低失效率时具有较高的检错有效性,但存在大量的计算开销,难以直接应用于实际项目中。与此同时,很多提高 ART 算法效率的改进算法被提出,如各种基于划分的 ART 算法[22,32,35]、镜像策略的 ART 算法[51-53]、遗忘策略的 ART 算法。这些算法虽然在一定程度上提高了算法效率,但同时也损失了部分有效性。其中,一些基于划分的 ART 算法用空间换时间,增加了空间开销。

(2)现有的 ART 实现方法中缺乏自动的 Oracle 技术[50];难以构造程序的预期输出,从而无法判定程序的执行结果是否正确。

测试 Oracle 问题不仅仅是 ART 算法面临的难题,同样是各种测试方法研究中共同的问题,它是软件测试中的两个关键问题之一[40]。很多研究缓解测试 Oracle 问题的方法,比如蜕变测试[64](不需要测试 Oracle 即可以测试程序),使用蜕变关系验证输出结果的正确性。文献[65]将蜕变测试应用于 D-ART 算法,在一定程度上缓解了测试 Oracle 问题,同时将源测试用例与衍生测试用例同时放入候选测试用例,计算源测试用例、衍生测试用例与已执行测试用例集的距离,应用 D-ART 算法使得测试用例的分布更均匀。

(3)高维输入域空间下有效性低下的问题。

将 ART 算法应用于低维输入域空间中时,其检错有效性较随机测试有明显提高,但随着输入域空间维度的增加,有效性降低,甚至不及随机测试。

5.2 节中对高维空间中有效性低下的问题进行了分析,随着输入域维度的增加,失效域边长增大,失效域更易集中在输入域的中部,而 ART 算法则是尽量均匀地分布测试用例,有些 ART 算法还存在边界效应(边界处产生的测试用例较中部更密集),这也是 ART 算法在高维输入域空间内有效性下降的原因。Mayer[69]提出在输入域中部生成测试用例的密度高于边缘区域的 ID-D-ART 算法,且通过仿真实验表明,在高维输入域空间中,算法的有效性大幅提高。

(4)非数值型输入数据间距离的度量问题。

应用非数值输入域的 ART 算法的关键在于非数值输入数据间的差异度度量,即距离的计算。文献[54]提出了 linear-order ART 算法,通过将非数值型输入进行分类并从类别中进行选择,实现输入数据间的差异性度量。实验结果显示,linear-order ART 算法的有效性优于 RT 算法,同时 linear-order ART 算法的开销接近于 RT 算法。

对于面向对象的程序,不同的输入涉及到多个类以及对象间通过方法调用进行的交互,对测试用例之间的距离进行度量具有更大的挑战,同样影响 ART 算法在面向对象程序中的适应性。Ciupa[55]提出了针对同一个类的不同对象间的距离进行度量的方法,将 ART 算法应用在面向对象的程序中,实现工具为 ARTOO[56]。Lin[57]提出了 divergence-oriented ART 算法对其进行改进,开发了名为 ARTGen 的工具。文献[58]提出了针对多个类应用 ART 算法的相似性度量(Similarity Metric),并在开源项目实验中验证了其有效性优于 RT 与已有 ART 算法。

(5)自动化工具的实现。

自 ART 算法被提出以来,十余年间各种类型的 ART 算法相继涌现,如基于距离的 ART[18-19]、基于限制的 ART[20]、基于网格的 ART[67-68]、准随机测试[66]、基于划分的 ART[22,32,35]、镜像策略的 ART[51-53]等,学术研究领域的成果丰硕。而 ART 算法的成果在实践中的应用依赖于自动化测试工具的实现。现有的自动化随机测试工具有 RANDOOP[60],QuickCheck[61-62],MoreBugs[63]等,基于 ART 算法的测试工具有 ARTOO[56],ARTGen[57]等。下一步工作是将更多具有良好性能改进的 ART 算法进行成果转化。

结束语　随机测试是经典的黑盒测试方法,也是具有潜力的测试策略之一。自适应随机测试作为随机测试的改进算法,具有随机测试无偏、无需程序任何信息、发现一切缺陷的潜力等优点,且具有比随机测试更高的检错有效性。但目前的 ART 算法很难在有效性和效率上同时满足要求,有效性较高的 D-ART 和 RRT 算法,其时间开销较大,难以投入实际应用。将自适应随机的思想与划分测试思想结合,目的是提高 ART 算法的效率。但划分自适应随机测试算法是牺牲部分有效性来提高效率。如何使 ART 算法满足简单、易实现、高有效性、高效率、低空间开销的要求,同时扩展其非数值型应用领域,提高高维输入域空间下算法的有效性,并确定有效的程序执行结果的正确性验证方法,都是 ART 算法进一步研究的方向。

参 考 文 献

[1] BERTOLINO A. Software testing research: Achievements, challenges, dreams[C]// Proceedings of Future of Software Engineering. 2007: 85-103.

[2] HUANG R, XIE X, CHEN T Y, et al. Adaptive Random Test Case Generation for Combinatorial Testing[C]// Proceedings of Computer Software and Applications Conference (COMPSAC). IEEE, 2012: 52-61.

[3] GRIESMAYER A, AICHERNIG B, JOHNSEN E B, et al. Dynamic Symbolic Execution for Testing Distributed Objects[C]// International Conference on Tests and Proofs. 2009: 105-120.

[4] CHOW T S. Testing software design modeled by finite-state machines[M]// Conformance testing methodologies and architectures for OSI protocols. IEEE Computer Society Press, 1995: 391-400.

[5] FUJIWARA S, BOCHMANN G V, KHENDEK F, et al. Test selection based on finite state models[J]. IEEE Transactions on Software Engineering, 1991, 17(6): 591-603.

[6] WHITTAKER J A, THOMASON M G. A markov chain model for statistical software testing[J]. IEEE Transactions on Software Engineering, 1994, 20(10): 812-824.

[7] ARIFIANI S, ROCHIMAH S. Generating test data using ant Colony Optimization (ACO) algorithm and UML state machine diagram in gray box testing approach[C]// Proceedings of Technology of Information & Communication. 2017: 217-222.

[8] WHITE L J, COHEN E I. A Domain Strategy for Computer Program Testing[J]. IEEE Transactions on Software Enginee-

ring,1980,6(3):247-257.

[9] MYERS G J,SANDIER C,BADGETT T. The art of software testing[M]. Wiley,2011.

[10] ORSO A,ROTHERMEL G. Software Testing:a Research Travelogue (2000-2014)[C]// Proceedings of 14th Future of Software Engineering(FOSE). 2014:117-132.

[11] GIRARD E,RAULT J C. A programming technique for software reliability[C]// Proceedings of the IEEE Symposium on Computer Software Reliability. 1973:44-50.

[12] CHEN T Y,KUO F C,ZHOU Z Q. On Favourable Conditions for Adaptive Random Testing[J]. International Journal of Software Engineering and Knowledge Engineering, 2007, 17(6): 805-825.

[13] LOO P S, TSAI W K. Random Testing Revisited[J]. Information and Software Technology,1988,30(7):402-417.

[14] ZHOU Z Q. Using Coverage Information to Guide Test Case Selection in Adaptive Random Testing[C]// Proceedings of 34th Annual IEEE Computer Software and Applications Conference Workshops. 2010:208-213.

[15] XIONG N. Random Testing with varied probability[D]. Hefei: University of Science and Technology of China,2013. (in Chinese)
熊能.变概率的随机测试[D].合肥:中国科学技术大学,2013.

[16] AMMANN P E,KNIGHT J C. Data diversity:An approach to software falut tolerance [J]. IEEE Transactions on Computers, 1998,37(4):418-425.

[17] FINELLI G B. Nasa sofware failure characterization experiments [J]. Reliability Engineering and System Safety,1991,32 (1-2):155-169.

[18] CHEN T Y,LEUNG H,MAK K. Adaptive Random Testing [C]// Proceedings of the IEEE International Conference on Quality Software(QSIC). IEEE,2008.

[19] 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,2004:57-76.

[20] CHAN K P,CHEN T Y,TOWEY D P. Restricted Random Testing[C]// Proceedings of 7th Events in Care Screening Questionnaire(ECSQ). Berlin,Germany,2002:321-330.

[21] GUTJAHR W J. Partition testing vs. random testing:The influence of uncertainty[J]. IEEE Transactions on Software Engineering,1999,25(5):661-674.

[22] CHEN T Y,MERKEL R G,EDDY G,et al. Adaptive RANDOM TESTING THROugh Dynamic Partitioning[C]// Proceedings of the Forth IEEE International Conference on Quality Software. Washington,2004:79-86.

[23] CHAN F T,CHEN T Y,MAK I K,et al. Proportional Sampling Strategy:Guidelines for Software Testing Practitioners[J]. Information and Software Technology,1996,38(12):775-782.

[24] BISHOP P G. The variation of software survival times for different operational input profiles[C]// Proceedings of the 23rd International Symposium on Fault-Tolerant Computing (FTCS-23). IEEE Computer Society Press,1993:98-107.

[25] CHEN T Y,KUO F C,MERKEL R. On the statistical properties of the F-measure[C]// Proceeding of the Fourth International Conference on Quality SOFTWARE(QSIC 2004). 2004: 146-153.

[26] CHEN T Y,KUO F C,MERKEL R. On the statistical properties of testing effectiveness measures[J]. Journal of Systems & Software,2006,79(5):591-601.

[27] CHAN K P,CHEN T Y,TOWEY D. Normalized Restricted Random Testing[M]// Reliable Software Technologies. DBLP, 2003:368-381.

[28] CHAN K P,CHEN T Y,TOWEY D P. Restricted Random Testing:Adaptive Random Testing by Exclusion[J]. International Journal of Software Engineering and Knowledge Engineering, World Scientific Publishing Company,2006,16(4):553-584.

[29] MAYER J,SCHNECKENBURGER C. An Empirical Analysis and Comparison of Random Testing Techniques[C]// Proceedings of the 2006 ACM/IEEE International Symposium on Empirical Software Engineering (ISESE'06). 2006:105-114.

[30] CHEN T Y,KUO F C,LIU H. Enhancing Adaptive Random Testing Through Partitioning by Edge and Centre[C]// Proceedings of the Australian Software Engineering Conference. Washington,2007:265-273.

[31] MAYER J. Adaptive random testing by bisection with restriction[C]// International Conference on Formal Methods and Software Engineering. Springer-Verlag,2005:251-263.

[32] CHEN T Y,HUANG D H. Adaptive random testing by localization[C]// Proceedings of the Software Engineering Conference,2004. IEEE,2004:292-298.

[33] MAYER J,SCHNECKENBURGER C. Adaptive random testing with enlarged input domain[C]// Proceedings of the Sixth international conference on quality software (QSIC'06). IEEE. 2006:251-258.

[34] SABOR K K. MOHSENZADEH M. Adaptive Random Testing Through Dynamic Partitioning By Localization with Distance and Enlarged Input Domain[J]. International Journal of Innovative Technology & Exploring Engineering,2012,1(6):147-155.

[35] CHEN T Y,HUANG D H,ZHOU Z Q. Adaptive Random Testing Through Iterative Partitioning[C]// Proceedings of the Eleventh Conference on Software Technologics. Berlin, 2006: 155-166.

[36] MAYER J,CHEN T Y,HUANG D H. Adaptive Random Testing through Iterative Partitioning Revisited [J]. Journal of Information Science and Engineering,2011,27(15):22-29.

[37] CHEN T Y,HAO D H,KUO F C. Adaptive Random Testing by Balancing[C]// Proceedings of the Second International Workshop on Random Testing, Colocated With the Twenty-second IEEE/ACM International Conference on Automated Software Engineering. New York,USA,2007:2-9.

[38] KOROSH K S,STUART T. Adaptive Random Testing by Static Partitioning[C]// Proceedings of the 2015 IEEE/ACM 10th International Workshop on Automation of Software Test. 2015: 28-32.

[39] CHEN T Y,KUO F C,ZHOU Z Q. On the Relationships Between the Distribution of Failure-causing Inputs and Effective-

ness of Adaptive Random Testing[C]// Proceedings of the 2005 Software Engineering and Knowledge Engineering (SEKE). 2005:306-311.

[40] KUO F C. On adaptive random testing[D]. Swinburne University of Technology,2006.

[41] T. R. D. of the Texas Legislative Council. Data for 2001 Redistricting in Texas. the Texas Legislative Council,Austin,Texas,2001[OL]. http://www.tlc.state.tx.us/pubspol/red2001data.pdf.

[42] VARACHIU C M,VARACHIU N. A fuzzy paradigm approach for the cognitive process of categorization[C]// Proceedings of the 1st IEEE International Conference on Cognitive Informatics (ICCI'02). IEEE Computer Society Press,2002:229-232.

[43] YOUNG H P. Measuring the compactness of legislative districts [J]. Legislative Studies Quarterly,1988,13(1):105-115.

[44] CHEN T Y,MERKEL R G. An Upper Bound on Software Testing Effectiveness[J]. ACM Transactions on Software Engineering and Methodology,2008,17(3):1-27.

[45] CHEN T Y,YU Y T. On the relationship between partition and random testing [J]. IEEE Transactions on Software Engineering,1994,20(12):977-980.

[46] WEYUKER E J,JENG B. Analysing partition testing strategies [J]. IEEE Transactions on Software Engineering,1991,17(7):703-711.

[47] MERKEL R. Analysis and enhancements of adaptive random testing[D]. Swinburn University of Technology,2005.

[48] LIU H,XIE X D,YANG J,et al. Adaptive Random Testing by Exclusion through Test Profile[C]// Proceedings of the 10th International Conference on Quality Software. 2010:92-101.

[49] MAYER J. Adaptive Random Testing with Randomly Translated Failure Region [C]// Proceedings of the First International Workshop on Random Testing. 2006:70-77.

[50] ARCURI A,BRIAND L. Adaptive Random Testing:An Illusion of Effectiveness? [C]// Proceedings of the International Symposium on Software Testing & Analysis. ACM,2011:265-275.

[51] KUO F C,An indepth study of mirror adaptive random testing [C]// Proceedings of the 9th International Conference on Quality Software (QSIC'09). 2009:51-58.

[52] CHEN T Y,KUO F C,MERKEL R,et al. Mirror adaptive random testing [J]. Information and Software Technology,2004,46(15):1001-1010.

[53] HOU S F,YU L,LI Z B,et al. Based on the Random Vector Mirror Method Improve the ART Algorithm [J]. Journal of Computer-Aided Design & Computer Graphics,2017,29(9):1750-1759. (in Chinese)
侯韶凡,于磊,李志博,等. 基于随机向量镜像策略改进 ART 算法[J]. 计算机辅助设计与图形学学报. 2017,29(9):1750-1759.

[54] BARUS A C,CHEN T Y,KUO F C,et al. A Cost-Effective Random Testing Method for Programs with Non-Numeric Inputs[J]. IEEE Transactions on Computers,2016,65(12):3509-3523.

[55] CIUPA I,LEITNER A,ORIOL M,et al. Object distance and its application to adaptive random testing of object-oriented programs[C]// Proceedings of the 1st Int. Workshop Random Testing. 2006:55-63.

[56] CIUPA I,LEITNER A,ORIOL M,et al. ARTOO:adaptive random testing for object-oriented software [C] // International Conference on Software Engineering. DBLP,2008:71-80.

[57] LIN Y,TANG X,CHEN Y,et al. A Divergence-Oriented Approach to Adaptive Random Testing of Java Programs [C] // Ieee/acm International Conference on Automated Software Engineering. IEEE,2009:221-232.

[58] CHEN J F,KUO F C,CHEN T Y,et al. A Similarity Metric for the Inputs of OO Programs and Its Application in Adaptive Random Testing[J]. IEEE Transactions on Reliability,2017,66(2):373-402.

[59] VISHAWJYOTI,GANDHID P. A survey on prospects of automated software test case generation methods[C]// International Conference on Computing for Sustainable Global Development. IEEE,2016:3867-3871.

[60] PACHECO C,ERNST M D. Randoop:Feedback-Directed Random Testing for Java[C]// Companion to the Acm Sigplan Conference on Object—oriented Programming System & Application Companion. 2007:815-816.

[61] CLAESSEN K,HUGHES J. QuickCheck:a lightweight tool for random testing of Haskell programs[C]// Acm Sigplan International Conference on Functional Programming. ACM,2000:268-279.

[62] ARTS T,HUGHES J,NORELL U,et al. Testing AUTOSAR software with QuickCheck [C] // IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops. IEEE,2015:1-4.

[63] HUGHES J,NORELL U,SMALLBONE N,et al. Find More Bugs with QuickCheck! [C] // Proceedings of the 2016 11th IEEE/ACM International Workshop in Automation of Software Test. IEEE,2016:71-77.

[64] BARUS A C,CHEN T Y,KUO F C. The Impact of Source Test Case Selection on the Effectiveness of Metamorphic Testing [C]// Proceedings of the 2016 1st International Workshop on Metamorphic Testing. IEEE,2016:5-11.

[65] HUI Z W,HUANG S. MD-ART:a test case generation method without test oracle problem [C] // International Workshop on Specification, Comprehension, Testing, and Debugging of Concurrent Programs. ACM,2016:27-34.

[66] LIU H,CHEN T Y. Randomized Quasi-Random Testing[J]. IEEE Transactions on Computers,2016,65(6):1896-1909.

[67] MAYER J. Lattice-based Adaptive Random Testing [C]// Proceedings of the Twenty IEEE/ACM International Conference on Automated Software Engineering. 2005:333-336.

[68] CHEN T Y,HUANG D H,KUO F C,et al. Enhance Lattice-based Adaptive Random Testing[C]// Proceeding of the Software Automated Conference (SAC). Honolulu, Hawaii, 2009:422-429.

[69] MAYER J. Towards Effective Adaptive Random Testing for Higher-Dimensional Input Domains [C] // Proceedings of the Conference on Genetic & Evolutionary Computation. 2006:1955-1956.