基于变异粒子群算法的字符串型测试数据生成

李 刚 于 磊 孙回回 张兴隆 侯韶凡
(解放军信息工程大学 郑州 450001) (数学工程与先进计算国家重点实验室 郑州 450001)

摘 要 基于搜索的算法在以路径覆盖为目标的测试数据生成中应用广泛。然而对于字符串型测试数据的生成,现有方法效率不高。为了高效地生成字符串型测试数据,提出了一种基于变异粒子群算法的字符串型测试数据自动生成方法。在随机生成初始种群后,采用粒子群算法使种群在趋近最优个体的过程中实现进化,并以一定的概率对种群中的个体进行变异操作,以避免进化过程陷入局部最优。为了有效地指导种群进化过程,对经典适应度函数中分支距离的计算方法进行改进,使其适用于含有字符串型参数的程序。实验结果表明,该方法具有较高的成功率和稳定性,且能明显提升测试数据生成效率。

关键词 测试数据生成,字符串型数据,分支距离,变异粒子群算法
中图法分类号 TP311.5 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2016.11.049

String-type Test Data Generation Based on Mutation Particle Swarm Optimization

LI Gang YU Lei SUN Hui-hui ZHANG Xing-long HOU Shao-fan
(The PLA Information Engineering University, Zhengzhou 450001, China)[1]
(State Key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou 450001, China)[2]

Abstract Search-based algorithm is widely used in test data generation for path coverage. However, current methods are not efficient enough dealing with string-type test data. In order to generate string-type test data efficiently, a novel approach based on mutation particle swarm algorithm (MPSO) was presented. In this approach, once a swarm is randomly generated, PSO is used to evolve the swarm by drawing the whole swarm towards the optimal particle, and mutation operation is carried out with certain probability to avoid falling into local optimal. For the sake of providing effective guidance for the search process, this paper improved the classical fitness function by modifying the calculation of branch distance to make it applicable for string-type test data. The experimental results show that our proposed method not only has high success rate and stability, but also can improve the efficiency of test data generation.

Keywords Test data generation, String-type test data, Branch distance, MPSO

1 引言

随着计算机软件的应用越来越广泛,软件的失效可能会造成巨大的经济损失,甚至危及人的生命安全。质量问题现已成为制约计算机软件的一个主要因素。许多计算机科学家在展望21世纪计算机科学发展方向和策略时,把提高软件质量放在优先于提高软件功能和性能的地位[1]。软件测试是保证软件质量的重要手段[2],而测试数据生成是软件测试过程中最困难且开销最大的任务之一[3]。

结构型测试是一种常用的软件测试方法,它以程序内部结构为依据,在输入域中寻找满足结构测试覆盖准则的输入数据,其中路径覆盖是一种常用的测试准则。结构型测试数据自动生成技术可以大致分为两种类型:静态方法和动态方法[4]。静态方法[5-7]分析被测程序的控制流、数据流信息,并通过符号执行等技术生成合适的测试数据,这种方法不需要执行被测程序。动态方法[8-11]通过执行程序并利用执行程序过程中得到的相关信息指导测试数据生成,常用的动态方法有随机测试、遗传算法、粒子群算法、蚁群算法等。

目前国内外对结构型测试数据自动生成方法已有许多研究成果。Kohsuke等人提出了一种反馈可控的随机测试数据生成方法,通过控制反馈信息量提升测试数据的多样性,以实现大规模实用程序的测试覆盖[10]。Shaukat等人将耦合路径作为输入,使用粒子群算法为其生成测试数据,并开发了相应的工具[11]。Moataz等人综合考虑了路径相似度和分支距离,设计了一种经典的适应度函数,并用遗传算法实现多路径测试数据生成[12]。Jiang等人利用程序切片技术简化目标路径和实际执行路径,获得了更有指导意义的适应度值,并用粒子群算法提高了测试数据的生成效率[13]。Chawla等人提出一种新的混合粒子群遗传算法用于生成面向对象程序测试数据,用遗传算法的选择、交叉、变异等操作替代粒子群算法原

到稿日期:2015-11-09 返修日期:2016-03-04 本文受国家科学自然基金项目(61402525),郑州市普通科技攻关项目(141PPTGG383)资助。
李 刚(1991-),男,硕士生,主要研究方向为软件质量工程,E-mail:cloudlg12@live.com;于 磊(1973-),男,博士,副教授,CCF高级会员,主要研究方向为软件工程、软件可靠性;孙回回(1991-),女,硕士生,主要研究方向为先进编译技术;张兴隆(1992-),男,硕士生,主要研究方向为软件质量工程;侯韶凡(1990-),女,硕士生,主要研究方向为软件质量工程。

有的更新方式[14]。Gordon 等人以整个测试套件为进化单元，使用遗传算法为被测程序生成测试数据，并讨论了初始种群设计、参数寻优等问题，还设计了相应的测试数据生成工具 Evo-Suite[15-18]。

以上研究中提出的方法都能在达到覆盖标准的基础上较高效地为被测程序生成测试数据，但它们都只涉及数值型参数的情况。然而，在实际程序中，非数值型参数如字符串的使用相当普遍，但这方面的研究却很少。Alshraideh 等人在编辑距离的基础上为 3 种字符串谓词设计代价函数，使用一种改进遗传算法生成字符串型测试数据，其改进措施是使用程序中的字符串常量构造初始输入并根据被测程序设计特定的变异操作算子[19]。赵瑞莲将不满足路径条件的字符串谓词表示成一个实值函数，利用快速下降搜索算法实施目标函数极小化，从而得到满足路径条件的测试输入[20]。McMinn 等人通过对被测程序源代码中的关键标识符进行网络查询，得到相关的字符串用于构造初始种群，并用遗传算法实现测试数据生成[21]。张晓迪提出了一种基于禁忌搜索的字符串测试数据自动生成方法[22]。Kim 等人提出了一种搜索树，可以有效地用多序列对比技术实现字符串型测试数据生成[23]。Sheeva 等人提出的字符串型测试数据生成方法不只考虑了测试覆盖率，同时也考虑了语言模型，以提高所生成数据的可读性，从而降低人工设计 Oracle 的难度[24]。

为了高效地生成字符串型测试数据，本文设计了一种变异粒子群算法，利用粒子群算法相比于其他搜索算法收敛速度快的特点[25]，并结合遗传算法中的变异操作，来克服粒子群算法容易陷入局部最优的缺陷，从而确保全局搜索能力，实现了以路径覆盖为标准的字符串型测试数据自动生成。此外，为了有效地指导种群进化过程，本文在经典适应度函数的基础上，设计了一种适用于字符串程序的适应度函数，利用程序插桩技术记录被测程序的实际执行路径和每个分支处的分支距离；对指定的程序路径，根据实际路径与目标路径的符合情况以及第一个不符合的节点处的分支距离计算该测试数据的适应度值。为验证本文方法的有效性，对几个典型的含有字符串谓词的程序进行测试，并与经典遗传算法[26]和 Alshraideh 的方法[19]进行对比后发现，文中提出的适应度函数是合理可行的，且设计的基于变异粒子群算法的面向路径字符串测试数据自动生成方法是高效稳定的。

本文的主要贡献有以下 3 点：1）设计了基于变异粒子群算法的字符串型测试数据生成方法；2）设计了含有字符串谓词的分支语句的分支距离计算方法；3）在经典适应度函数的基础上，设计了一种适用于字符串程序的适应度函数。

本文第 2 节提出了基于变异粒子群算法的字符串型测试数据自动生成方法；第 3 节介绍了适应度函数的设计；第 4 节给出了实验结果及分析；最后总结全文。

2 基于变异粒子群算法的字符串测试数据自动生成方法

本文提出的变异粒子群算法（Mutation Particle Swarm Optimization，MPSO）主要借鉴遗传算法变异操作的思想和粒子群算法以种群全局最优和个体自身最优位置为依据进行搜索的思想。面对一个被测程序，首先需要明确目标路径集合，再构造种群并使用变异粒子群算法进行搜索，以生成满足覆盖目标的测试数据。种群中的个体是根据被测程序参数类型及参数个数构造的，每个个体是一个多维向量，向量的每一维存储被测程序的一个参数值。

在执行搜索算法生成测试数据前需要对被测程序进行插桩，即在源程序中插入一些语句，以记录程序执行的路径和每个分支处的分支距离[27]。被测程序的目标路径集合就是通过控制流分析和插桩信息得到的。

在算法执行过程中，以适应度函数驱动测试数据更新与适应度函数的设计详见第 3 节。考虑到被测程序参数为单个字符串的情况，使用粒子群优化对种群中的每个个体进行更新。更新时以整个种群到达过的适应度值最大的位置和每个个体在搜索过程中到达过的适应度值最大的位置所对应的字符串为参考，使当前个体存储的字符串接近上述两个最优位置所对应的字符串。粒子群优化可以使当前字符串以较快的速度在长度上接近最优字符串；也能将一些可能会提高个体适应度值的字符引入到当前字符串中；并且在当前字符串接近最优字符串的过程中，还有可能使个体到达一些适应度值更高的位置，这种情况下需在该轮更新结束后更新最优位置。

使用粒子群优化对种群进行更新时，可能会出现种群中所有个体都与最优个体相同的情况，所以算法在完成一轮基于粒子群优化的更新后，以一定的概率对当前种群中的个体进行变异操作。此变异操作可以使种群跳出搜索过程中的局部最优解，继续搜索，以保证能找到符合覆盖目标的测试数据。

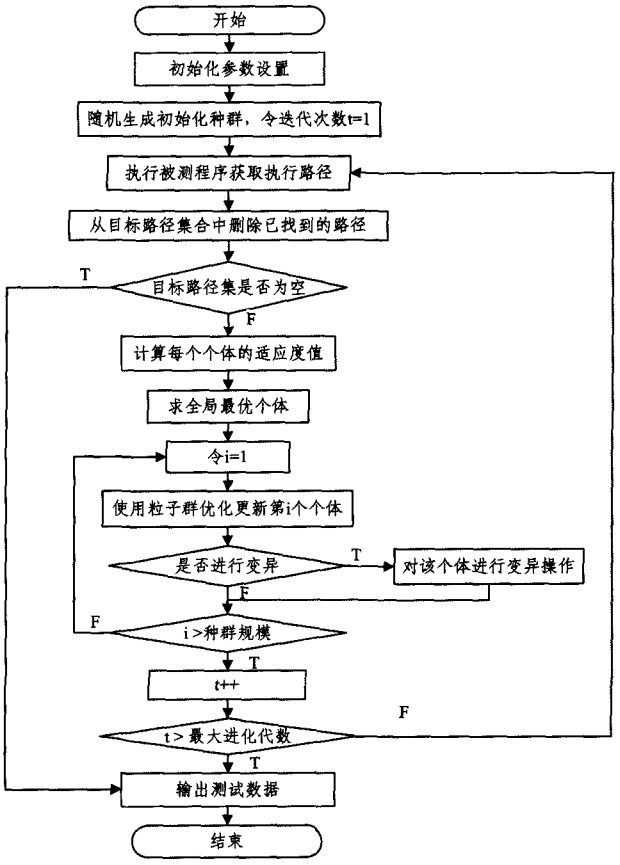

图 1 基于变异粒子群算法的字符串测试数据生成流程

图 1 描述了基于变异粒子群算法的字符串测试数据自动生成的流程。首先初始化设置种群规模、最大进化代数、插入概率、删除概率、替换概率、变异概率等算法相关参数，并使用

随机算法生成初始种群。随后分别以当前种群的每个个体存储的参数作为程序参数，执行插桩后的被测程序，并记录每个个体的执行路径及执行过程中在各分支处的分支距离。从目标路径集合中删除已找到的路径，并以剩余的目标路径作为目标路径集合计算每个个体的适应度值，并更新个体的自身最优位置。判断目标路径集合是否为空，若为空则停止搜索并输出测试数据；否则，更新全局最优位置。使用粒子群优化更新种群中的每个个体，并以一定的概率对个体进行变异操作，得到新的种群。最后判断是否已达到最大进化代数，若是则停止搜索并输出测试数据，算法终止；否则返回继续执行搜索。

算法流程中，使用粒子群优化更新个体即更新个体中的每个参数，更新操作包括插入、删除和替换。个体更新与变异的算法的伪代码如图 2 所示。

```
1. //功能:对当前字符串 str_p 进行更新、变异
2. //输入:当前字符串 str_p,全局最优字符串 str_gb,自身最优字符
       串 str_pb
3. //输出:更新后的字符串
4. procedure Update_Algorithm(str_p, str_gb, str_pb)
5.   flag = FALSE
6.   if str_gb.length>str_p.length then
7.       对 str_p 进行插入操作
8.   else if str_gb.length<str_p.length then
9.       对 str_p 进行删除操作
10.  else
11.      len=str_p.length
12.      for j:=0 to len do begin
13.          if str_p.charAt(j)!=str_pb.charAt(j) then
14.              根据 str_pb 对 str_p 进行替换操作
15.              flag=TRUE
16.          endif
17.      endfor
18.      if flag==FALSE then
19.          for j:=0 to len do begin
20.              if str_p.charAt(j)!=str_gb.charAt(j) then
21.                  根据 str_gb 对 str_p 进行替换操作
22.              endif
23.          endfor
24.      endif
25.  endif
26.  if random()<chanceOfMutation then
27.      对 str_p 进行变异操作
28.  endif
29. end Update_Algorithm
```

图 2　更新与变异操作的伪代码

如果当前字符串长度比种群全局最优个体的字符串长度小，则在当前字符串中任意位置插入一个随机字符。如当前字符串为“TAG”，全局最优个体的字符串为“TARGT”，则经过插入操作后，当前字符串可能会变为“TAEG”。如果当前字符串长度比种群全局最优个体的字符串长度大，则随机删除当前字符串中的一个字符。如当前字符串为“TARGATE”，全局最优个体的字符串为“TARGAT”，则经过删除操作后，当前字符串可能会变成“TARGTE”。如果当前字符串与全局最优个体的字符串长度相同，则从第一位开始逐位比较当前字符串与自身最优字符串。若第 j 位不同，则以自身最优字符串的第 j 位替换当前字符串的第 j 位，得到下一代的字符串；若当前字符串与自身最优字符串完全相同，则从第一位开始逐位比较当前字符串与全局最优字符串，替换操作和与自身最优字符串比较时相同。如当前字符串为“TAEGAT”，自身最优字符串为“TARGET”，则经过替换操作后，当前字符串会变成“TARGAT”。

为了确保算法不会陷于局部最优解，搜索过程还以一定的概率对个体进行变异操作。若确定对个体进行变异操作，则等概率地对个体进行插入、删除或者替换。其中插入、删除操作与更新操作中的插入、删除相同；替换操作在字符串中随机选取一个字符，对其 ASCII 码进行加 1 或减 1。如当前字符串为“TARGAT”，则经过变异的替换操作后，当前字符串可能会变成“TARGBT”。

3　适应度函数

在使用搜索算法生成测试数据时，用适应度函数来表示种群中个体优秀程度的数值，是搜索算法与实际问题的结合点。

对于路径覆盖，适应度函数可以考虑层接近度和分支距离[12]，分别用 V 和 D 表示，适应度函数通常就是层接近度与分支距离的函数，即 $fitness=f(V,D)$。层接近度表示个体实际执行路径与目标路径的匹配程度，一种可行的方法是求两条路径从第一个节点开始相同的节点数；分支距离是指对于一条目标路径，将个体所代表的测试数据执行程序得到的实际路径与其对比，在两条路径的第一个不匹配节点处，程序对该分支条件的偏移程度。例如：对于分支条件 $if(str.length()==5)$，若此时 $str=$“abc”，则程序实际执行的是此分支结构的 else 分支，即对应表达式为 $if(!str.length()==5)$，程序在此处的分支距离为 $abs(str.length()-5)=2$。

为了实现对路径覆盖的指导，将反映实际路径与目标路径相似程度的层接近度作为适应度函数的主要参考依据。与此同时，加入分支距离来区分执行路径相同个体间的优劣程度，偏离程度越大的个体，分支距离越大，适应度值越小。加入对分支距离的考虑，可以有效地加速进化过程。

3.1　适应度函数设计

对于种群中的任意个体，本文设计的适应度函数如下：

$$fitness=\max(fit_1,fit_2,\cdots,fit_n) \tag{1}$$

$$fit_i=\sum_{j=0}^{div}(j+1)+1.001^{-D\cdot m} \tag{2}$$

其中，n 为目标路径集合的规模；div 为该个体执行路径与第 i 条目标路径从第一个节点开始，连续相同的节点数；$1.001^{-D\cdot m}$ 表示对分支距离进行规约化，使其取值范围为(0，1)，m 为用于放大分支距离的区分度的系数；fit_i 表示个体对于第 i 条路径的适应度值，当个体对于某条目标路径的适应度值最大时，说明个体的执行路径与该目标路径的匹配程度最高，此时该个体要通过进化使执行路径接近该目标路径的难度相对较小。因此以该个体对于 n 条目标路径适应度值中的最大值作为个体对目标路径集合的适应度值，种群将会优先对适应度值最大的路径进行搜索。

对于参数为字符串的被测程序，求解其测试数据的适应度值时，层接近度的求法与数值型程序相同，主要区别在于分支距离的求解。下面详细介绍包含字符串的分支语句的分支距离的计算方式。

3.2 字符串分支语句的分支距离

Java 程序中经常出现的与字符串相关的简单条件和它们对应的分支距离计算公式如表 1 所列。

表 1 字符串分支语句的分支距离计算公式

表达式	分支距离
a. length()==x	k
!(a. length()==x)	abs(a. length())−x)
a. length()<b. legnth()	b. length()−a. length()
!(a. length()<b. length())	a. length()−b. length()
a. compareTo(b)>0	a. compareTo(b)
!(a. compareTo(b)>0)	b. compareTo(a)+k
a. equals(b)	k
!(a. equals(b))	$ED(b,a)+(1+1.001^{-CD(b,a)})$
A‖B	φ(A) +φ(B)
¬(A‖B)	min(φ(¬A),φ(¬B))
A&&B	min(φ(A),φ(B))
¬(A&&B)	φ(¬A) +φ(¬B)

本文只讨论 if-else 结构的分支。对于简单条件，当程序第 i 个分支处的 if 语句中的表达式为真时，在程序执行路径节点列表中插入一个值为$(i,0)$的节点，同时分支距离列表插入表 1 中 if 语句表达式为 FALSE 时的分支距离。若 if 语句中表达式为假，即程序执行进入 else 分支时，在程序执行路径节点列表中插入一个值为$(i,1)$的节点，同时分支距离列表插入表 1 中 else 对应表达式为 FALSE 时的分支距离。

对于 *a. length()==x*，*a. length()<b. legnth()*，*a. compareTo(b)>0* 等与数值相关的表达式，分支距离计算公式与数值型测试数据生成时类似。k 为常数，$k>0$，通常取值为 1。k 在分支距离中单独出现时表示对应表达式很容易实现，非单独出现时是为了使分支距离不为 0。

对于 *a. equals(b)*类型的表达式，为了生成与目标相同的字符串，分支距离考虑编辑距离[28]和字符距离[19]，可以充分反映将当前字符串转换为目标字符串的难度，计算公式如表 1 所列。其中 $ED(b,a)$表示 b,a 的编辑距离；$CD(b,a)$表示 b,a 的字符距离；$1.001^{-CD(b,a)}$把字符距离规约到$(0,1)$。

对于复合条件，$\varphi(A)$等价于$\neg A?0:D(A)$，$D(A)$为表达式 A 的分支距离。当$\neg A$ 为 TRUE 时，$\varphi(A)=0$；当$\neg A$ 为 FALSE 时，$\varphi(A)=D(A)$。$\varphi(\neg\neg A)$同理。

4 实验与分析

4.1 实验设置

实验在 Windows XP Professional SP3 操作系统下的 My Eclipse8.5 环境中开展，处理器为 Intel(R) Core(TM)2 Quad CPU 2.83GHz，内存为 1.85GB。

实验中 MPSO 的参数设置如表 2 所列。

表 2 实验参数设置

参数名称	MPSO
变异概率	0.30
插入概率	0.33
删除概率	0.33
替换概率	0.33
种群规模	100
最大进化代数	100000

4.2 测试程序

实验选取 5 个典型被测程序，这些程序中均含有字符串谓词，具体描述如表 3 所列，其中前 4 个被测程序来源于文献[19]，第 5 个被测程序来源于文献[22]。表 3 各列分别给出了被测程序的名字、参数个数及类型、代码行数、代码中谓词表达式的数量以及目标路径数。其中参数类型的角标表示该类型参数的个数，$String_2$ 即表示程序输入中有 2 个 String 类型参数。

表 3 被测程序

Name	Parameters	LOC	PE	Paths
Calc	$[String_1]$	70	11	12
Cookie	$[String_3]$	31	5	6
DataParse	$[String_2]$	59	13	24
FileSuffix	$[String_2]$	55	9	11
Find	$[String_1]$	6	1	2

4.3 结果与分析

为验证本文提出的方法的有效性，对上述 5 个被测程序进行实验，并与经典遗传算法、Alshraideh 的方法[19]进行对比。其中经典遗传算法是字符串型测试数据生成研究中常用的对比方法，标记为 GA，本文方法获得的测试结果标记为 MPSO，Alshraideh 的方法获得的测试结果标记为 Alshraideh 方法。

4.3.1 效率测试

为说明本文方法的效率，对表 3 中的前 4 个被测程序进行 1000 次实验，并计算平均运行时间，测试结果如图 3 所示。

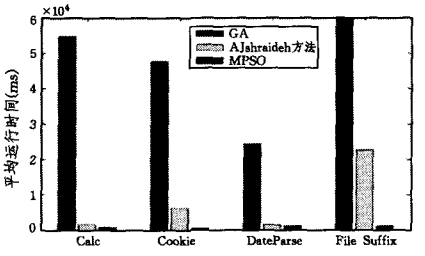

图 3 被测程序平均运行时间

从图 3 可以看出，对于 4 个被测程序，MPSO 在平均运行时间上远优于 GA。这是由于 MPSO 收敛速度远快于 GA，所需进化迭代次数较少，而 MPSO 与 GA 每一次迭代搜索过程的所需时间相近，因此 MPSO 平均运行时间比 GA 少。MPSO 在收敛速度快的基础上，还具有全局搜索能力，保证总能在最大进化代数内找到满足所有目标路径的测试数据。而 GA 收敛速度较慢，对于被测程序 Calc，Cookie 和 FileSuffix，GA 甚至还存在无法为所有目标路径生成测试数据的情况。

对于 Cookie 和 FileSuffix，MPSO 的时间开销相比于 Alshraideh 方法有明显提升，达到了一个数量级。其原因是在 Cookie 和 FileSuffix 中，作为参数的字符串具有一定的对应关系，逻辑关系复杂，本文提出的方法能够高效地为其生成测试数据，而 Alshraideh 方法对于复杂逻辑关系的搜索能力较弱，无法高效地为其生成测试数据。对于 Calc 和 DateParse，MPSO 的时间开销仅略优于 Alshraideh 方法。其原因是 Calc 和 DateParse 中的参数相互独立，逻辑关系较简单，Alshraideh 方法和本文方法都能够通过较少的搜索操作生成所需测试数据，所以时间开销较小。

综上所述，MPSO 的效率优于 GA 和 Alshraideh 方法，而且对于逻辑关系复杂的程序，其效率上的提升更明显。

4.3.2 字符串长度对效率的影响

为了观察字符串长度对算法效率的影响，对表 3 中的 Find 程序进行实验，实验结果如表 4 所列。

表 4　Find 程序实验结果

字符串长度	平均运行时间		
	GA	Alshraideh 方法	MPSO
1	7.35	5.43	5.62
2	14.06	12.11	10.00
3	44.59	30.27	15.86
4	63.91	51.88	24.53
5	208.13	103.47	35.31
6	957.65	289.71	54.84
7	3519.85	798.72	62.19
8	11706.43 (11)	1874.34	75.16
9	26027.00 (823)	4023.18	105.47
10	—— (1000)	8892.35	137.5

由表 4 可以看出，随着目标字符串长度的增加，MPSO 相对于 GA 和 Alshraideh 方法的优势更加明显。目标字符串长度增加到 8 和 9 时，GA 生成指定字符串的成功率由 100%下降到 99.89%和 17.6%；当目标字符串长度增加到 10 时，GA 生成指定字符串的成功率已降到了 0%，而此时 MPSO 仍能在很短的时间内生成指定字符串。当解空间过大时，GA 由于收敛速度慢，难以在有限时间内找到可行解。

4.3.3　算法稳定性分析

以图 4 表示 MPSO，GA 和 Alshraideh 方法在图 3 所示实验中的进化代数分布情况，图中的胡须（盒子的垂直延伸线）两端分别表示最大观测值与最小观测值。

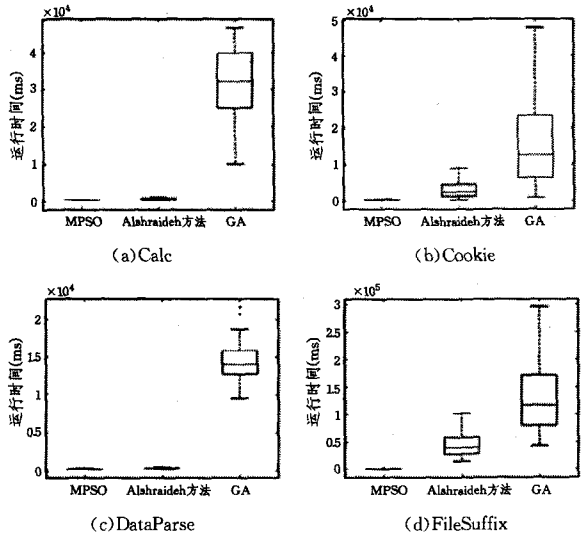

图 4　被测程序运行时间分布图

对于所有被测程序，MPSO 运行时间的中位数均小于 GA 和 Alshraideh 方法的对应值。采用下四分位数至上四分位数、最小观测值到最大观测值的区间范围两个指标来衡量运行时间的波动范围。波动范围越大，表示算法的稳定性越差。对于 5 个被测程序，MPSO 在这两个指标上均优于 GA 和 Alshraideh 方法，说明 MPSO 的运行时间波动范围小于 GA 和 Alshraideh 方法。由此可以得出结论，相对于 GA 和 Alshraideh 方法，MPSO 具有更高的稳定性。

结束语　软件测试的目的是以尽可能少的代价发现尽可能多的软件缺陷，测试数据生成是软件测试过程中的一个关键环节。基于搜索算法的路径测试数据生成是一种广泛使用的测试数据生成方法。然而，现有的测试数据生成方法大多不考虑字符串型参数或生成效率较低。本文讨论了字符串分支语句分支距离的计算方法，在此基础上借鉴粒子群算法根据个体自身最优和种群全局最优进行搜索的思想和遗传算法用变异操作实现全局搜索的思想，提出了一种基于变异粒子群算法的字符串型测试数据自动生成方法。为了有效地指导种群进化过程，在经典适应度函数的基础上，设计了一种适用于字符串程序的适应度函数。最后针对一些典型被测程序进行了实验，并与现有算法进行了对比，实验验证了本文方法的高效性和稳定性。此外，本文还分析了字符串长度对算法效率的影响。

目前，本文方法仅关注结构型程序，没有考虑面向对象程序中包含字符串型参数的情况。而现实程序中，有很多是用面向对象技术实现的，所以下一步应考虑将字符串型测试数据自动生成技术应用到面向对象程序的测试数据生成中。

参 考 文 献

[1] 朱鸿，金凌紫. 软件质量保障与测试[M]. 北京：科学出版社，1997

[2] Lyu M R, Rangarajan S, Moorsel A P. Optimal Allocation of Test Resources for Software Reliability Growth Modeling in Software Development[J]. IEEE Transactions on Reliability, 2002,51(2):183-192

[3] Offutt A J, Jin Z Y, Pan J. The Dynamic Domain Reduction Approach to Test Data Generation[J]. Software Practice and Experience, 1999,29(2):167-193

[4] Phil M. Search-based Software Test Data Generation: A Survey [J]. Software Testing, Verification and Reliability, 2004,14(2): 105-156

[5] Jensen C S, Prasad M R, Moller A. Automated Testing with Targeted Event Sequence Generation[C]// Proceedings of the 2013 International Symposium on Software Testing and Analysis. Lugano, Switzerland, ACM, 2013:66-77

[6] Roberto B, Matthieu C, Roberta G, et al. Symbolic Path-Oriented Test Data Generation for Floating-Point Programs[C]// Proceedings of the 6th International Conference on Software Testing, Verification and Validation. Luemberg, IEEE, 2013:1-10

[7] Wang Xiao-yin, Zhang Ling-ming, Philip T. Experience Report-How is Dynamic Symbolic Execution Different from Manual Testing[C]//Proceedings of the 2015 International Symposium on Software Testing and Analysis. Baltimore. USA, ACM, 2015: 199-210

[8] Yi Min-jie. The Research of Path-Oriented Test Data Generation Based on a Mixed Ant Colony System Algorithm and Genetic Algorithm[C]// Procee-dings of the 8th International Conference on Wireless Communications, Networking, and Mobile Computing (WiCOM). Shanghai, China, IEEE, 2012:1-4

[9] Boussaa M, Barais O, Sunye G, et al. A Novelty Search-based Test Data Generator for Object-oriented Programs[C]// Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation. Madrid, Spain, ACM, 2015:1359-1360

[10] Yatoh K, Sakamoto K, Ishikawa F, et al. Feedback-Controlled Random Test Generation[C]//Proceedings of the 2015 International Symposium on Software Testing and Analysis. Baltimore, USA, ACM, 2015:316-326

（下转第 279 页）

[13] Lian X, Chen L, Huang Z. Keyword Search over Probabilistic RDF Graphs[J]. IEEE Trans on Knowledge and Data Engineering, 2014, 27(5): 1246-1260

[14] Zheng Zhi-yun, Liu Bo, Li Lun, et al. Research of Keyword Search Model over RDF Data Graph [J]. Computer Science, 2015, 42(7): 234-239(in Chinese)
郑志蕴, 刘博, 李伦, 等. 基于关键词的 RDF 数据图查询模型研究[J]. 计算机科学, 2015, 42(7): 234-239

[15] Haye J, Gutiërrez C. Bipartite graphs as intermediate model for RDF [C]// Proc. of the 3rd International Semantic Web Conference, Lecture Notes in Computer Science. 2004: 47-61

[16] Li Hui-ying, Qu Yu-zhong. KREAG: Keyword query approach over RDF data based on entity-triple association graph [J]. Chinese Journal of Computers, 2011, 34(5): 825-835(in Chinese)
李慧颖, 瞿裕忠. KREAG: 基于实体三元组关联图的 RDF 数据关键词查询方法[J]. 计算机学报, 2011, 34(5): 825-835)

[17] Jia Shu-fang, Li Lei. Chinese query expansion based on user log clustering [C]// Proc. of IEEE International Conference on Network Infrastructure and Digital Content. 2009: 446-451

[18] Liu C H, Qi R H, Liu Q. Query expansion terms based on positive and negative association rules [C]// Proc. of International Conference on Information Science and Technology. IEEE, 2013: 802-808

[19] Pal D, Mitra M, Datta K. Improving query expansion using WordNet [J]. Journal of the Association for Information Science and Technology, 2014, 65(12): 2469-2478

[20] Semantic Web Technology Evaluation Ontology[OL]. http://lsdis.cs.uga.edu/Projects/SemDis/Swetodblp

(上接第 256 页)

[11] Khan S A, Nadeem A. Automated Test Data Generation for Coupling Based Integration Testing of Object Oriented Programs Using Particle Swarm Optimization (PSO)[C]// Proceedings of the 7th International Conference on Genetic and Evolutionary Computing. Prague, Czech Republic, Springer, 2013: 115-124

[12] Ahmed M A, Hermadi I. GA-based Multiple Paths Test Data Generator [J]. Computers & Operations Research, 2008, 35(10): 3107-3124

[13] Jiang S J, Yi D D, Ju X L, et al. An Approach for Test Data Generation Using Program Slicing and Particle Swarm Optimization [J]. Neural Computing and Applications, 2014, 25(7/8): 2047-2055

[14] Pritanka C, Inderveer C, Ajay R. A Novel Strategy for Automatic Test Data Generation Using Soft Computing Technique[J]. Frontiers of Computer Science, 2015, 9(3): 346-363

[15] Fraser G, Arcuri A. Whole Test Suite Generation [J]. IEEE Transactions on Software Engineering, 2013, 39(2): 276-291

[16] Fraser G, Arcuri A. EvoSuite: Automatic Test Suite Generation for Objected-Oriented Software[C]// Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering. Szeged, Hungray, 2011: 416-419

[17] Arcuri A, Fraser G. On Parameter Tuning in Search Based Software Engineering [C]// Proceedings of the 3rd International Symposium on Search Based Software Engineering (SSBSE). Szeged, Hungary, 2011: 33-47

[18] Fraser G, Arcuri A. The Seed is Strong-Seeding Strategies in Search-Based Software Testing[C]// Proceedings of the 5th International Conference on Software Testing, Verification and Validation (ICST). Montreal, Canada, 2012: 121-130

[19] Mohammad A, Leonardo B. Search-based Software Test Data Generation for String Data Using Program-Specific Search Operators[J]. Software Testing Verification and Reliability, 2006, 16(3): 175-203

[20] Zhao R L. Searc-Based Automatic Path Test Generation for Character String Data [J]. Journal of Computer-Aided Design & Computer Graphics, 2008, 20(5): 671-677(in Chinese)
赵瑞莲. 基于搜索的面向路径字符串测试数据自动生成方法[J]. 计算机辅助设计与图形学学报, 2008, 20(5): 671-677

[21] Phil M, Muzammil S, Mark S. Search-Based Test Input Generation for String Data Types Using the Results of Web Queries [C]// Proceedings of the 5th International Conference on Software Testing, Verification & Validation. Washington DC, USA, IEEE, 2012: 141-150

[22] Zhang X D. Automatic String Test Data Generation Based on Tabu Search [D]. Beijing: Beijing University of Chemical Technology, 2013(in Chinese)
张晓迪. 基于禁忌搜索的字符串型测试数据自动生成[D]. 北京: 北京化工大学, 2013

[23] Kim S H, Kim K, Cho H G. A New String Search Tree with Multiple Alignment[C]// Proceedings of the 12th International Conference on Computer and Information Technology. Chengdu, China, 2012: 456-463

[24] Sheeva A, Phil M, Mark S. Evolving Readable String Test Inputs Using a Natural Language Model to Reduce Human Oracle Cost[C]// Proceedings of the 6th International Conference on Software Testing, Verification and Validation. Luembourg, 2013: 352-361

[25] Mao C Y, Yu X X, Xue Y Z. Algorithm Design and Empirical Analysis for Particle Swarm Optimization-Based Test Data Generation [J]. Journal of Computer Research and Development, 2014, 51(4): 824-837(in Chinese)
毛澄映, 喻新欣, 薛云志. 基于粒子群优化的测试数据生成及其实证分析[J]. 计算机研究与发展, 2014, 51(4): 824-837

[26] Whitley D. The GENITOR Algorithm and Selective Pressure: Why Rank Based Allocation of Reproductive Trials is Best[C]// Proceedings of the 3rd International Conference on Genetic Algorithms, Fairfax, VA. San Francisco, USA, Morgan Kaufmann, 1989: 116-121

[27] Huang J C. Program Instrumentation and Software Testing[J]. Computer Journal, 1978, 11(4): 25-32

[28] Gonzalo N. A Guided Tour to Approximate String Matching [J]. ACM Computing Surveys, 2001, 33(1): 31-88