A Guided Genetic Algorithm for Automated Crash Reproduction

一种用于自动崩溃再现的引导式遗传算法

张国豪
(1634061005)

1

前言
(Background)

Crash（崩溃）：

Crash 和 Debug

尽管我们拥有很多实践模型和设计模式来帮助编写优秀的软件，但仍然不能保证某个软件从不崩溃。这也是软件工程中的一项挑战。

相关工作
(Related Work)

2

- 为了减少开发人员为崩溃调试做出的努力，已经有了很多种自动故障再现的解决方案。这些技术通常旨在产生触发目标故障的测试。
  例如，记录重放方法（record-replay approaches）。

有弊端

- 与之不同的技术还有，post-failure approaches.

  尝试通过利用故障后可用的数据来复制崩溃，通常存储在日志文件或外部的错误跟踪系统中。然而这些技术除了 崩溃堆栈轨迹（crash stack traces）之外，还需要其他附加的特定的输入数据，例如core dumps ，类似 input grammars或者class invariants这样的软件模型这样的附加信息对于开发人员来说是不可得到的或者不可用的。
- 所以最近的这个领域的研究是基于崩溃堆栈轨迹，作为调试信息的唯一来源。

总的来说，可分为三类：

(i) record-play approaches ;
（记录重放方法）

(ii) post-failure approaches using various data sources,
（使用各种数据源的故障后方法）

(iii) stack-trace based post-failure techniques.
（基于堆栈轨迹的故障后技术）

基于崩溃堆栈跟踪（crash stack trace）
这样的技术现在有

- STAR 一种基于向后符号执行的方法。优于早期的崩溃复制技术，如Randoop 和 BugRedux。

- MuCrash 一种使用特定变异算子更新现有测试用例的工具，从而创建了一个新的测试池

- JCHARMING 一种基于有向模型检验与程序分层

然而，这些工具都有着各种各样的限制和不足。例如STAR无法处理具有外部环境依赖性的情况（例如文件或网络输入）等。

3

EvoCrash

本文提出了一种基于进化的搜索方法，名为EvoCrash，用于崩溃再现。

- EvoCrash构建在著名的Java自动测试生成工具EvoSuite 之上
- 它允许 堆栈跟踪 去引导搜索，从而减少搜索空间
- GGA使用新颖的生成程序　来构建 运行至少一个在崩溃堆栈帧中报告的方法的　初始测体。

The contribution of the paper are:

- 一种基于崩溃复制的新颖的导向遗传算法（GGA）
- EvoCrash，一个实现GGA的Java工具
- 关于涉及不同版本的三个开源项目的50个真实世界软件崩溃的实证研究，显示EvoCrash可以复制41个案例（82%），34个（89%）可用于调试（debugging）；
- 将EvoCrash与基于崩溃堆栈跟踪的三种最先进的方法进行比较（STAR ，MuCrash 和 JCHARMING ）。

The EvoCrash Approach 算法

Crash Stack Trace Processing:（崩溃堆栈轨迹过程）

崩溃再现的最佳测试案例必须在与原始崩溃相同的位置崩溃，并产生尽可能接近原始轨迹的堆栈轨迹。

因此，在EvoCrash中，我们首先解析作为输入给出的日志文件(log file)，以提取感兴趣的崩溃堆栈帧。（the crash stack frames）

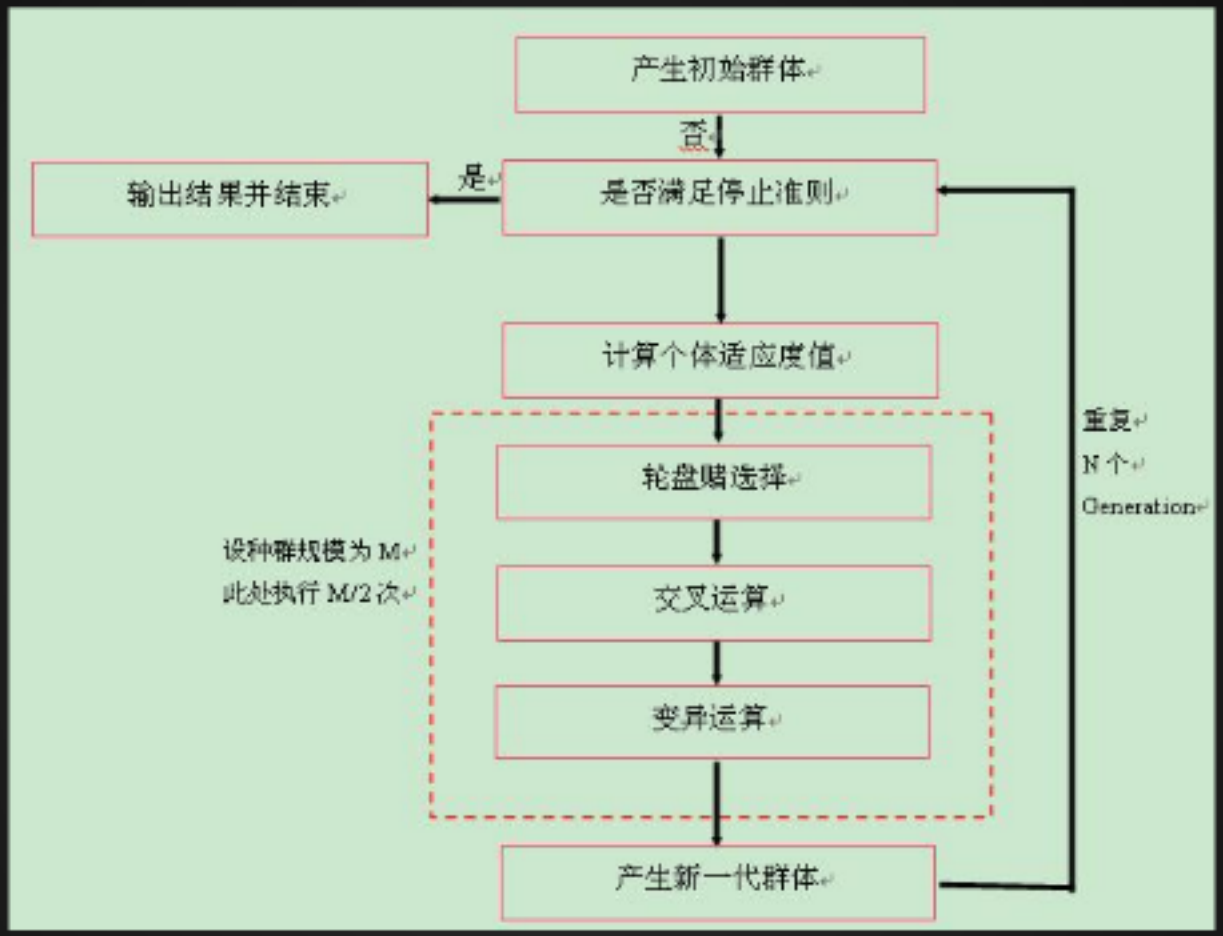
标准Java堆栈轨迹包含

（ i ）抛出的异常的类型

（ ii ）崩溃时生成的堆栈帧列表。

每个堆栈帧 (stack frame) 对应于故障 (failure) 中涉及的一种方法，因此它包含其识别生成异常的地方所需的所有信息：

（ i ）方法名称; （ ii ）class 名称，（ iii ）行号


产生初始群体
否
是
输出结果并结束
是否满足停止准则
计算个体适应度值
轮盘赌选择
设种群规模为 M
此处执行 M/2 次
交叉运算
变异运算
产生新一代群体
重复
N 个
Generation

GGA：

GGA包含标准遗传算法的所有主要步骤：

- (i) 初始化种群；
- (ii) 它使用交叉和突变对后续世代进行演化
- (iii) 根据适应度函数(fitness fuction)，在每一代中选择最适合的测试集 (the fittest test)。

主要的不同之处在于：

- (i) 使用一个新的程序去初始化生成种群；
- (ii) 新的遗传交叉算子；
- (iii) 一个新的突变算子;

4

引导式遗传算法的过程

Initial Population（初始化种群）

在传统的基于覆盖的工具中（traditional coverage-based tools），例如 EvoSuite、JTExpert，它们尽可能多地去调用目标类中的方法，尽管并不是崩溃复制的主要目标。

因此，GGA对目标故障中涉及到的方法给予更高的优先级。
In particular, our routine gives higher importance to methods contained in crash stack frames.
保证了每次测试中至少一次崩溃中的方法被插入到初始化测试中。
（Algorithm ensures that at least one method of the crash is inserted in each initial test.）

在每次迭代，创建一个空的测试 t，以填充随机的语句。

我们将一个公共方法的调用从Mcrash或C的成员类中插入。
在第一次迭代中，Mcrash中的崩溃方法以较低概率

p = 1 / size（第7行）插入，其中size是要在t中添加的语句的总数。

在随后的迭代中，当在t（15-17行）中没有插入来自Mcrash的方法时，这种概率自动增加。

因此，算法2确保在每个初始测试中插入至少一种崩溃方法。

Algorithm 2: MAKE-INITIAL-POPULATION

Input: Class under test C
Set of failing methods M_{crash}
Population size N

Result: An initial population P_0

```
1  begin
2      P_0 ⟵ ∅
3      while | P_0 | < N do
4          t ⟵ empty test case
5          size ⟵ random integer ∈ [1;MAX_SIZE]
6          // probability of inserting a method involved in the failure
7          insert_probability ⟵ 1/size
8          while (number of statements in t) < size do
9              if random_number ⩽ insert_probability then
10                 method_call ⟵ pick one element from M_crash
11                 // reset the probability of inserting a failing method
12                 insert_probability ⟵ 1/size
13             else
14                 method_call ⟵ pick one public method in C
15                 length ⟵ number of statements in t
16                 // increase the probability of inserting a failing method
17                 insert_probability ⟵ 1/(size − length + 1)
18             INSERT-METHOD-CALL(method_call, t)
19         P_0 ⟵ P_0 ⋃ t
```

Fitness function 适应度函数：

我们的适应度函数被设计为考虑三个必须保持的主要条件，以便将测试用例评估为最优，

（i）抛出异常的行（语句）必须被覆盖，

（ii）必须抛出目标异常，

（iii）生成的堆栈跟踪必须与原始堆栈跟踪类似

$$f(t) = 3 \times d_s(t) + 2 \times d_{except}(t) + d_{trace}(t) \qquad (1)$$

$$f(t) = 3 \times d_s(t) + 2 \times d_{except}(t) + d_{trace}(t) \qquad (1)$$

ds（t）表示执行目标语句的距离，即崩溃的位置;

dexcept（t）∈{0,1}是指示是否抛出目标异常的二进制值;

dtrace（t）测量生成的堆栈轨迹（如果有的话）和预期轨迹之间的距离。

$$D(S^*, S) = \sum_{i=1}^{\min\{k,n\}} \varphi\left(\text{diff}(e_i^*, e_i)\right) + |\, n - k \,| \qquad (2)$$

$$S^* = \{e_1^*, \ \ldots, e_n^*\} \quad e_i^* = (C_1^*, m_1^*, l_1^*)$$

$$S = \{e_1, \ldots, e_k\}$$

$$D(S^*, S) = \sum_{i=1}^{\min\{k,n\}} \varphi\left(\text{diff}(e_i^*, e_i)\right) + \mid n - k \mid \tag{2}$$

$$\varphi(x) = x/(x+1)$$

$$D(S^*, S) = \sum_{i=1}^{n} \min\left\{\text{diff}(e_i^*, e_j) : e_j \in S\right\} \tag{3}$$

$\text{diff}(e_i^*, e_j)$ as follows:

$$\text{diff}(e_i^*, e_i) = \begin{cases} 3 & C_i^* \neq C_i \\ 2 & C_i^* = C_i \text{ and } m_i^* \neq m_i \\ \varphi\left(\mid l_i^* - l_i \mid\right) & \text{Otherwise} \end{cases} \tag{4}$$

$$d_{trace}(t) = \varphi\left(D(S^*, S)\right) = D(S^*, S)/(D(S^*, S) + 1) \tag{5}$$

Guided Crossover（引导交叉）

Algorithm 3: GUIDED-CROSSOVER

Input: Parent tests p_1 and p_2
Set of failing methods M_{crash}

Result: Two offsprings o_1, o_2

```
1  begin
2      size_1 ⟵ | p_1 |
3      size_2 ⟵ | p_2 |
4      // select a cut point
5      μk ⟵ random number ∈ [0; 1]
6      // first offspring
7      o_1 ⟵ first μ × size_1 statements from p_1
8      o_1 ⟵ append (1 − μ) × size_2 statements from p_2
9      CORRECT(o_1)
10     if o_1 does not contain methods from M_crash then
11         o_1 ⟵ clone of p_1
12     // second offspring
13     o_2 ⟵ first μ × size_2 statements from p_2
14     o_2 ⟵ append (1 − μ) × size_1 statements from p_1
15     CORRECT(o_2)
16     if o_2 does not contain methods from M_crash then
17         o_2 ⟵ clone of p_2
```

Guided Crossover（引导交叉）

传统的单点交叉表示，它通过在两个父测试p1和p2之间随机交换语句来生成两个后代。

```
    给定随机切点 μ
    the first offspring o1
        inherits the first μ statements from parent p1,
        # p1的前 μ 个语句
        followed by | p2 | -μ statements from parent p2.
        # | p2 | -μ statements from p2
    the second offspring o2
        will contain μ statements from parent p2
        # p2的前 μ 个语句
        and | p1 | -μ statements from the parent p1.
        #   | p1 | -μ     from p1
```

不足之处

虽然父母双方p1,p2都包含了一个或者多个 从崩溃堆栈跟踪中 的 一个或多个失败的方法，交叉执行之后，这些调用的方法可能会只被移到一个子代而已。(only)

改进：

```
    (i)它选择随机切点μ（第五行）
    (ii) it recombines statements from the two parents around the cut-point
            (lines 7-8 and 12-13 of Algorithm 3).
    After this recombination, if o1 (or o2) loses the target method calls
    # 如果丢失目标方法的调用
    (a call to one of the methods reported in the crash stack trace),
    # 在崩溃堆栈跟踪中 报告的 一个方法 的调用。
    we reverse the changes and re-define o1 (or o2) as pure copy of its parent p1
                                                        (p2 for offspring o2)
    (if conditions in lines 10-11 and 16-17).
    #我们就将倒退这改变，重新定义 o1 为 父节点 p1 的纯复制
```

Guided Mutation (引导突变)

Algorithm 4: GUIDED-MUTATION

Input: Test $t = \langle s_1, \ldots, s_n \rangle$ to mutate
Set of failing methods M_{crash}

Result: Mutated test t

```
1  begin
2      n ⟵| t |
3      apply_mutation ⟵ true
4      while apply_mutation == true do
5          for i =1 to n do
6              φ ⟵ random number ∈ [0; 1]
7              if φ ⩽ 1/n then
8                  if delete probability then
9                      delete statement s_i
10                 if change probability then
11                     change statement s_i
12                 if insert probability then
13                     insert a new method call at line i
14         if t contains method from M_crash then
15             apply_mutation ⟵ false
```

Guided Mutation：

（After Crossover）交叉之后，通过添加（adding statement），更改（changing）和删除（removing）一些语句，新的测试通常会突变（mutated）。

添加statement没事，changing和removing会动到崩溃堆栈框架中的方法的相关调用。

引导突变 中的 Changing、Removing

在更改位置i（第10-11行）中的语句时，变异算子必须处理两种不同的情况：

（i）如果语句si是原始变量（例如，整数）的声明，则其原始值用另一个随机值（例如，另一个随机整数）改变;

（ii）如果si包含方法或构造函数 m，那么通过用具有相同返回类型的另一个公共方法/构造函数替换m来应用该突变，而其输入参数（对象或原始值）取自t中的先前i-1语句，设置为null（仅对于对象）或随机生成

最后，删除方法调用（算法4中的第8-9行）需要删除用作输入参数的相应变量和对象（如果这样的变量和对象不被t中的任何其他方法调用使用）。

如果测试 t 由于突变而失去目标方法调用（即，Mcrash中的方法），则重复行4-15中的循环，直到在t中重新插入一个或多个目标方法调用; 否则突变过程终止

5

实证研究
（Empirical study）

本论文研究包括三个真实世界 开源项目 的 50个bugs：

- Apache Commons Collections（ACC),
- Apache Ant（ANT),
- Apache Log4j（LOG)
- ACC是一个流行的Java库，具有25,000行代码（LOC），它提供扩展Java Collection Framework的实用程序。对于这个库，我们选择了2003年10月至2012年6月期间在Jira上公布的12个错误报告，因此涉及到五种不同的ACC版本。
- ANT是一个具有超过100,000行代码的大型Java构建工具，它支持不同的内置任务，包括编译，运行和执行Java应用程序的测试。对于ANT，我们选择了2004年4月至2012年8月期间在Bugzilla上提交的20个错误报告，并关注10个不同版本和子模块。
- LOG是一个广泛使用的Java库，具有20,000行代码，用于实现Java应用程序的日志记录实用程序。对于这个库.，我们选择了在2001年6月到2009年10月之间的时间窗口内报告的18个错误报告，并且与三个不同的LOG版本相关。

TABLE I

REAL-WORLD BUGS USED IN OUR STUDY.

Project	Bug IDs	Versions	Exception	Priority	Ref.
ACC	4, 28, 35, 48, 53, 68, 70,77, 104, 331, 277, 411	2.0 - 4.0	NullPointer (5), UnsupportedOperation (1), IndexOutOfBounds, (1) IllegalArgument(1), ArrayIndexOutOfBounds, (2) ConcurrentModification, (1) IllegalState (1),	Major (10) Minor (2)	[6] [12]
ANT	28820, 33446, 34722, 34734, 36733, 38458, 38622, 42179, 43292, 44689, 44790, 46747, 47306, 48715, 49137, 49755, 49803, 50894 51035, 53626	1.6.1 - 1.8.2	ArrayIndexOutOfBounds (2), NullPointer (17), ArrayIndexOutOfBounds (1) StringIndexOutOfBounds (1)	Critical (2) Medium (14) Medium (14),	[6] [7]
LOG	29, 43, 509, 10528, 10706, 11570, 31003, 40212, 41186, 44032, 44899, 45335, 46144, 46271, 46404, 47547, 47912, 47957	1.0.2 - 1.2	NullPointer (17), InInitializerError (1)	Critical (1) Major (4) Medium (11) Enhanc. (1) Blocker (1)	[6] [7]

Project	Bug ID	EvoCrash	STAR [6]	MuCrash [12]	JCHARMING [7]
ACC	4	Y	Y	Y	-
	28	Y	Y	Y	-
	35	Y	Y	Y	-
	48	Y	Y	Y	-
	53	**Y**	**Y**	**N**	-
	68	N	N	N	-
	70	**Y**	**N**	**N**	-
	77	NU	NU	N	-
	104	N	Y	Y	-
	331	**Y**	**N**	Y	-
	377	**Y**	**N**	Y	-
	411	Y	Y	Y	-
ANT	28820	N	N	-	-
	33446	NU	NU	-	-
	34722	**Y**	**N**	-	-
	34734	**NU**	**N**	-	-
	36733	NU	NU	-	-
	38458	Y	Y	-	-
	38622	NU	Y	-	Y
	42179	**Y**	**N**	-	-
	43292	N	Y	-	-
	44689	**Y**	**NU**	-	-
	44790	Y	Y	-	-
	46747	N	N	-	-
	47306	N	N	-	-
	48715	N	N	-	-
	49137	**Y**	**NU**	-	-
	49755	Y	Y	-	-
	49803	Y	Y	-	-
	50894	**Y**	**NU**	-	-
	51035	N	N	-	-
	53626	**Y**	**N**	-	-
LOG	29	Y	Y	-	
	43	N	N	-	
	509	**Y**	**N**	-	
	10528	**Y**	**N**	-	
	10706	**Y**	**N**	-	
	11570	Y	Y	-	
	31003	Y	Y	-	
	40212	**Y**	**NU**	-	
	41186	Y	Y	-	
	44032	**Y**	**N**	-	
	44899	**Y**	**N**	-	
	45335	**Y**	**NU**	-	
	46144	**Y**	**N**	-	
	46271	NU	Y	-	
	46404	**Y**	**N**	-	
	47547	Y	Y	-	
	47912	**Y**	**NU**	-	
	47957	**NU**	Y	-	

粗体表示可以由EvoCrash触发的错误，而其他技术不行。

带下划线的条目表示EvoCrash无法再现的错误，而有另一种技术可以。

可以看出，有22个（粗体）的情况，EvoCrash优于现有技术，

EvoCrash无法处理的2个（下划线）的情况。

EvoCrash结果如上表所示，

- EvoCrash可以成功复制我们数据集中的大多数崩溃。
- 在复制的cases中，

  EvoCrash在 ACC 中的 12个（83%）中有10个被复制，

  ANT 20个中有14个（70%），

  LOG 18个中17个（94%）。

总的来说，它可以在50个crashes（崩溃）中复制41个（82%），优于其他算法。

6

Conclusion

- 为了在调试时提高开发人员的生产力，已经提出了一些自动化崩溃复制方法。然而，现有的解决方案有一定的局限性，这些限制对于覆盖现实世界软件项目的更多崩溃案例的能力产生不利影响

- 本文介绍了EvoCrash，它是基于搜索的方法，使用崩溃堆栈轨迹中的数据 来崩溃复制。（Crash Replication）（Crash stack trace）

- EvoCrash应用了一种新颖的引导式遗传算法（GGA）以及智能适应度函数，以搜索可以触发目标崩溃的测试用例，并显示崩溃堆栈轨迹中的错误帧。

- 我们的实证评估表明，EvoCrash解决了三项尖端方法面临的主要挑战，从而在自动化崩溃再现方面表现优于其他算法。

7

Thanks!