基于SOM神经网络的二阶变异体约简方法研究

汇报人： 宋利

导师： 刘靖 副教授

主要内容

1.研究背景

2.相关工作

3.研究目标

4.研究方法

5.理论分析

6.实验及结果分析

1.研究背景 (1)

(1) 变异测试

- 一种面向缺陷的测试技术
- 变异算子产生变异体来模拟软件中的各种缺陷
- 满足变异测试准则的测试用例集具有更高的缺陷检测能力

(2) 高阶变异测试

- 实际缺陷是由两个或者更多个缺陷一起造成
- 高阶变异测试用以解决程序中复杂缺陷的问题
- 在模拟程序复杂缺陷方面有着重要意义

1.研究背景(2)

(3) 高阶变异体数量大量增加

- 增加执行开销

(4) 高阶变异测试的研究热点

- 减少高阶变异体数量
- 发现隐藏高阶变异体

主要内容

1.研究背景

2.相关工作

3.研究目标

4.研究方法

5.理论分析

6.实验及结果分析

2.相关工作（1）

（1）文献[1]指出高阶变异体约简的方法主要有三种不同的方法. 选择高阶变异体（基于搜索的软件工程的遗传算法）、减少变异算子的数量（last-to first的变异算子选择算法和互不同的变异算子选择算法）、减少变异位置的个数（数据流技术）。

（2）文献[2]基于动态变异体筛选和变异体包含提出了使用聚类技术实现变异体约简的方法。但该方法适用于一阶变异体，对二阶变异体，因多个变异体组合进行多次比较耗费时间而不适用。

[1] Ghiduk AS, Girgis MR, Shehata MH, Higher order mutation testing: A systematic literature review, Computer Science Review, 2017,25:29-48

[2] Ma YS, Kim SW, Mutation testing cost reduction by clustering overlapped mutants, Journal of Systems and Software 115 (2016) 18-30.

2.相关工作（2）

（3）文献[3] 基于一定的组合策略对二阶变异体在执行开销和变异充分度上进行验证和比较，得到执行开销明显降低以及二阶变异充分度和一阶变异充分度较为接近的结论。但是该组合策略不能涵盖所有的变异体组合，导致程序中有些错误组合没有被表示出来。

（4）文献[4] 指出SOM神经网络是一种通过模拟大脑神经系统自组织特征映射功能进行的无监督竞争式学习前馈网络，根据神经网络学习提取数据中的重要特征或某种内在规律对数据进行聚类。该网络具有很强的聚类能力,并且在各领域得到了广泛的应用[5]。

[3] Ghiduk AS, Reducing the number of higher-order mutants with the aid of data flow, E-Informatica Software Engineering Journal 10 (2016) 31-49.

[4] Kohonen T. The self-organizing map, Proc. IEEE 1990,pp.1464-1480.

[5] Delgado S, Higuera C, Calle-Espinosa J, Morn F, Montero F, A som prototype-based cluster analysis methodology, Expert Systems with Applications, 2017,88:14-28.

2.相关工作（3）

不足：

一方面，现有方法中在研究如何减少二阶变异体的计算开销时忽略了程序中可能的错误组合；另一方面，在实际的环境中，由于多个缺陷一起造成的情况普遍存在，因此有必要提出一种既能在充分考虑二阶变异体组合时减少二阶变异体计算开销又能发现隐藏二阶变异体的方法。

主要内容

1.研究背景

2.相关工作

3.研究目标

4.研究方法

5.理论分析

6.实验及结果分析

3.研究目标

本文针对充分考虑二阶变异体组合时减少二阶变异体执行开销（efficient）及发现隐藏二阶变异体（effective）的问题，提出了使用SOM神经网络模型聚类对二阶变异体进行约简的方法。该方法首先使用组合策略对一阶变异体进行组合形成二阶变异体；然后根据二阶变异体在测试执行过程中的中间值相似性，进行基于SOM神经网络模型的变异体聚类，达到充分考虑二阶变异体组合时减少了二阶变异体执行开销而且也发现隐藏二阶变异体的效果。

主要内容

1.研究背景

2.相关工作

3.研究目标

4.研究方法

5.理论分析

6.实验及结果分析

4.0.研究方法
—基础

- 定义1：二阶变异体的中间结果　指的是仅运行至二阶变异体中两个变异点后得到的结果，而非运行完整个程序，记为intermediate results of second-order mutants,简单记为Irs。
- 定义2：条件下二阶等价变异体　指对于相同的测试用例，一组二阶变异体的中间结果相同，但可能会被该测试用例杀死的二阶变异体，记为second-order conditionally overlapped mutants，简单记为Second_CoM。

4.0.研究方法

—示例

```
1 int my(int A,int B){
2 int C;
3 if(A>B)
4 C=C+A;
5 else
6 C=C-A;
7 C=C+B;
8 return C;
9 }
```

（3）

```
1 int my(int A,int B){
2 int C;
3 if(A<B)
4 C=C+A;
5 else
6 C=C-A;
7 C=C+B;
6 C=C-A;
7 C=C*B;
8 return c;
9 }
```

（4）

Second_CoMs = {{1、5、17、21},{2、6、18、22},{3、7、19、23},{4、8、12、16、20、24、28},{9、13、25},{10、11、14、15、26、27}}

ID为1、5、17、21的二阶变异体的中间结果为-8;

ID为2、6、18、22的二阶变异体的中间结果为-6;

ID为3、7、19、23的二阶变异体的中间结果为-2;

ID为4、8、12、16、20、24、28的二阶变异体的中间结果为0;

ID为9、13、25的二阶变异体的中间结果为8;

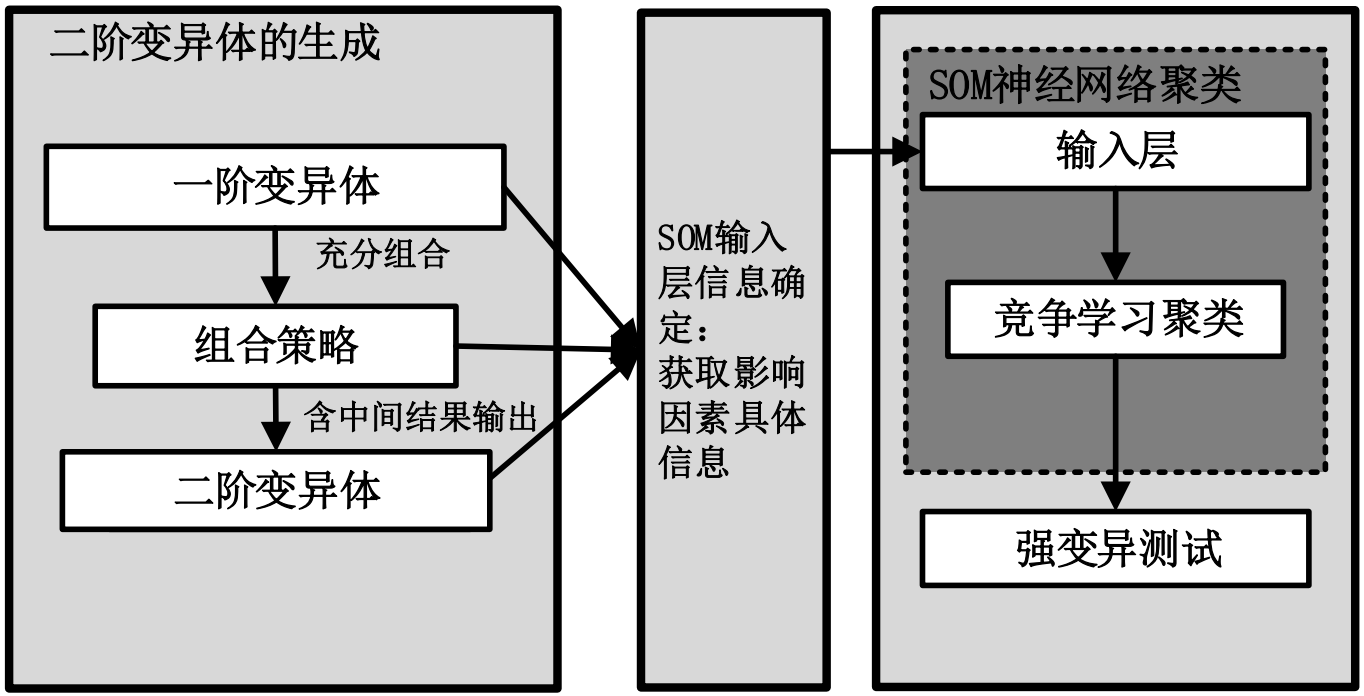
ID为10、11、14、15、26、27的二阶变异体的中间结果为2.

测试数据信息（A=4,B=2）

ID	Line	Combinations	Mutanted code	*Irs*
1	3-7	ROR_1-AOR_1	$A < B\ C = C * B$	-8
2	3-7	ROR_1-AOR_2	$A < B\ C = C - B$	-6
3	3-7	ROR_1-AOR_3	$A < B\ C = C \div B$	-2
4	3-7	ROR_1-AOR_4	$A < B\ C = C\%B$	0
5	3-7	ROR_2-AOR_1	$A \leq B\ C = C * B$	-8
6	3-7	ROR_2-AOR_2	$A \leq B\ C = C - B$	-6
7	3-7	ROR_2-AOR_3	$A \leq B\ C = C \div B$	-2
8	3-7	ROR_2-AOR_4	$A \leq B\ C = C\%B$	0
9	3-7	ROR_3-AOR_1	$A \geq B\ C = C * B$	8
10	3-7	ROR_3-AOR_2	$A \geq B\ C = C - B$	2
11	3-7	ROR_3-AOR_3	$A \geq B\ C = C \div B$	2
12	3-7	ROR_3-AOR_4	$A \geq B\ C = C\%B$	0
13	3-7	ROR_4-AOR_1	$A \neq B\ C = C * B$	8
14	3-7	ROR_4-AOR_2	$A \neq B\ C = C - B$	2
15	3-7	ROR_4-AOR_3	$A \neq B\ C = C \div B$	2
16	3-7	ROR_4-AOR_4	$A \neq B\ C = C\%B$	0
17	3-7	ROR_5-AOR_1	$A == B\ C = C * B$	-8
18	3-7	ROR_5-AOR_2	$A == B\ C = C - B$	-6
19	3-7	ROR_5-AOR_3	$A == B\ C = C \div B$	-2
20	3-7	ROR_5-AOR_4	$A == B\ C = C\%B$	0
21	3-7	ROR_6-AOR_1	False $C = C * B$	-8
22	3-7	ROR_6-AOR_2	False $C = C - B$	-6
23	3-7	ROR_6-AOR_3	False $C = C \div B$	-2
24	3-7	ROR_6-AOR_4	False $C = C\%B$	0
25	3-7	ROR_7-AOR_1	True $C = C * B$	8
26	3-7	ROR_7-AOR_2	True $C = C - B$	2
27	3-7	ROR_7-AOR_3	True $C = C \div B$	2
28	3-7	ROR_7-AOR_4	True $C = C\%B$	0

4.研究方法

—方法框架


二阶变异体的生成
一阶变异体
充分组合
组合策略
含中间结果输出
二阶变异体
SOM输入层信息确定：获取影响因素具体信息
SOM神经网络聚类
输入层
竞争学习聚类
强变异测试

4.1.二阶变异体的产生

使用变异测试工具*muJava*产生一阶变异体。

对任意两个不同位置处的任意两个一阶变异体进行组合以充分模拟程序中的复杂缺陷。 组合策略

对两个一阶变异体逐行读取，然后根据一阶变异体中变异点行数和变异算子组合等信息替换掉对应的语句形成二阶变异体。

为了二阶变异体在执行测试例时能够及时获取二阶变异体中间结果，本文对*muJava*工具的源码进行了修改。源码修改

下一步

4.1.二阶变异体的产生

Algorithm 1 Combination strategy

```
1:  sum ← 0
2:  br ← mutation_log;
3:  for i = 0 → (s = br.readLine())! = null do
4:      st ← s.split(" : ")
5:      numLine[i] ← st[1]
6:      M_1st[i] ← st[0]
7:  end for
8:  for i = 0 → (s = br.readLine())! = null do
9:      j ← i + 1
10:     while numLine[i].equals(numLine[j]) do
11:         j ← j + 1
12:     end while
13:     while numLine[j]! = null do
14:         file ← FileM_1st[i] + "And" + M_1st[j])
15:         file.mkdirs()
16:         f1 ← FileReader(M_1st[i] + "//name.java")
17:         f2 ← FileReader(M_1st[j] + "//name.java")
18:         f3 ← FileWriter(M_1st[i] + "And" + M_1st[j] +
            "//name.java")
19:         FullFileName ← M_1st[i]+"And"+M_1st[j]+
            "//name.java"
20:         com ← newCombination()
21:         com.write(f1, f2, f3)
22:         com.compile(FullFileName)
23:         sum ← sum + 1
24:         j ← j + 1
25:     end while
26: end for
27: br.close() =0
```

根据一阶变异体的日志中“：”的第二列，获取到一阶变异控制不同行不同位置的变异体进行组合

根据一阶变异体的日志中“：”的第二列，确定组合形成到二阶变异

在二阶变异体形成的同时对二阶变异体进行编译。因为测试用例进行测试的时候需要有.class文件

返回

4.1.二阶变异体的产生

获取变异点变异后表达式

Algorithm 2 AOR mutation operator

1: $Irs \leftarrow m.getLeft() + m.op$

2: $out.print("statement")$

3: $out.print("System.out.println(" -$

4: $CodeChangeLog.writeLog(class\_name +$
$MutationSystem.LOG\_IDENTIFIER +$
$mutated\_line + method\_signature + log\_str) = 0$

将源程序非变异点语句读取到新生成的.java文件中

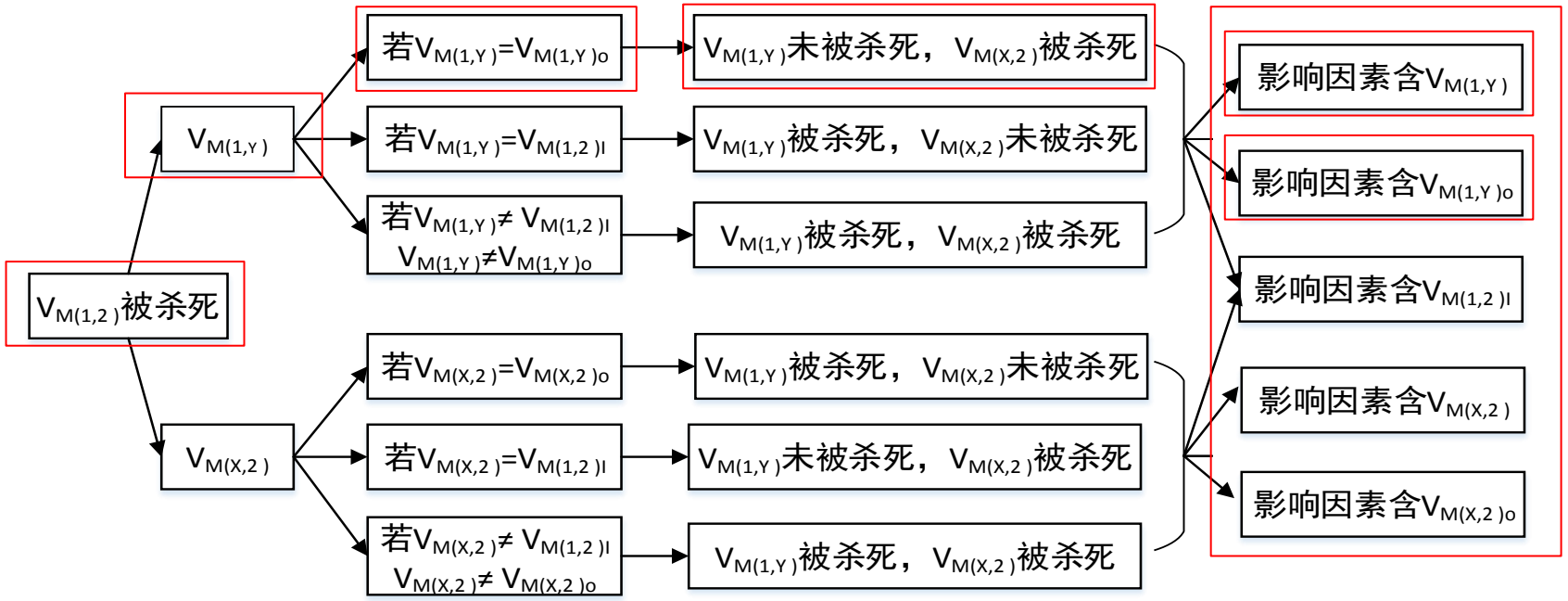
使变异点变化后的结果在变异体中有输出

将变异点如位置和变异算子等信息写入变异体日志log中

返回

4.2.SOM模型输入层确定（1）

复杂分析：影响二阶变异体的中间结果被杀死的原因



如果第一变异体的执行结果(V_(M_((1,Y))))和第一变异体的预期结果(V_(M_((1,Y)) o))相同（因为第二变异体的执行结果起的作用，导致二阶被杀死），则可知造成二阶变异体的中间结果被杀死的原因是第一变异体没有被杀死，第二变异体被杀死。

4.2.SOM模型输入层确定（2）

简单分析：直观的影响因素

```
...
1  if(a>=0||b<=0||c<=0)
2        return INVALID;
3  tri = 0;
4  if (a == b)
5        tri = tri - 1;
6  if (a == c)
7        tri = tri + 2;
8  if (b == c)
9        tri = tri + 3;
10 if (tri == 0)
11       if (a + b < c || a + c < b || b + c < a)
12                 return INVALID;
13       else
14                 return SCALENE;
15 if (tri > 3)
16       return EQUILATERAL;
...
```

变异后为真时

第二变异体变异结果对二阶变异体最终结果无影响

因素含：

测试用例、变异点位置、变异算子、两个变异体的距离.

4.2.SOM模型输入层确定（3）

简单分析：影响隐藏二阶变异体的因素

隐藏二阶变异体：

二阶变异体中单独的第一变异体和
第二变异体均被杀死，
但两者组合后的二阶变异体没有被杀死的二阶变异体.

因素中必含两个变异点分别变异前的预期结果、
两个变异点分别变化后的结果、
两个变异点同时发生变异后的结果

4.3.SOM神经网络设计

```
#1、对数据集和权值矩阵进行归一化处理
dataSet, old_dataSet = normalize(dataSet)
com_weight = normalize_weight(com_weight)
#遍历归一化后的数据集
for data in dataSet:
#2、得到获胜神经元
  n , m = getWinner(data , com_weight)
#得到神经元的N邻域
  neibor = getNeibor(n , m , N_neibor , com_weight)
  for x in neibor:
    j_n=x[0];j_m=x[1];N=x[2]
     #3、权值调整
    com_weight[j_n][j_m]=com_weight[j_n][j_m]+eta(t,N)*
                          (data - com_weight[j_n][j_m])
```

1、初始化权值：随机小数；
初始化迭代次数：1000；
初始化学习率：0.9；
对输入向量和权值做归一化处理 ；

2、获胜神经元获得方式：输入样本与权值向量的欧几里得距离，距离最小的神经元赢得竞争。

3、更新权值:对获胜的神经元拓扑邻域内的神经元进行更新
4、更新学习率η，判断是否收敛。如果学习率η<=ηmin或达到预设的迭代次数时结束

主要内容

1.研究背景

2.相关工作

3.研究目标

4.研究方法

5.理论分析

6.实验及结果分析

5.理论分析(1)

采用Shin等人[6]提出的变异测试的数学理论框架，对本文所提方法的核心理论进行分析和证明。

1、如果两个程序的运行结果不同，测试差分d记为1，否则记为0。

$$\mathrm{d}(t, px, py) = \begin{cases} 1, & px \text{ 和} py \text{运行结果不同} \\ 0, & px \text{ 和} py \text{运行结果相同} \end{cases}$$

[6] D. Shin and D. H. Bae, "A Theoretical Framework for Understanding Mutation-Based Testing Methods," 2016 IEEE International Conference on Software Testing, Verification and Validation (ICST), Chicago, IL, 2016, pp. 299-308.

5.理论分析(1)

2、簇M1中的所有变异体的中间结果之间相似性：

$$\forall M1_{xi} \in M1, \exists t \in TS, \neg d(t, M1_{x1}, M1_{xi}) = 1 (i = 2,3,..,m)$$

3、变异体之间的差异性：

$$\forall M1_{xi} \in M_{all}, \forall Mj_{xk} \in (M_{all} - M1), \exists t \in TS,$$

$$\begin{cases} d(t, M1_{x1}, M1_{xi}) = 0, & i = 2,3,\dots,m. \\ d(t, M1_{x1}, Mj_{xk}) = 1, & j = 2,3,..,N; k = 1,2,\dots,y \end{cases}$$

5.理论分析(1)

4、二阶变异体的约简原理推导:

$$\forall \mathrm{mi} \in M_{all}, \exists t \in TS, d(t, m1, mi) = 1$$

$$\Rightarrow \forall \mathrm{mi} \in \sum_{\mathrm{i=2}}^{\mathrm{n}} \mathrm{Mi}, \exists \mathrm{t} \in \mathrm{TS}, \mathrm{d}(\mathrm{t}, \mathrm{m1}, \mathrm{mi}) = 1$$

$$\approx \forall \mathrm{mi} \in \sum_{i=2}^{n} \mathrm{Mi}_{x1}, \exists t \in TS, d(t, \mathrm{M1}_{x1}, mi) = 1$$

⇒执行全部变异体与执行约简后变异体的运行结果相同.

5.理论分析(2)

1、某一测试用例下隐藏二阶变异体的数学模型为：

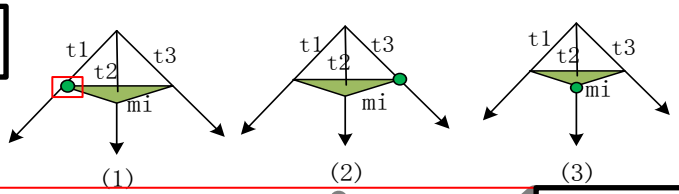
$$\exists \mathrm{mi} \in M_{all}, \forall t \in TS, \text{使得：} \begin{cases} d(t, po, mi(1, po)) = 1 \\ d(t, po, mi(po, 2)) = 1 \\ d(t, po, mi(1,2)) = 0 \end{cases}$$

2、所有测试例下的隐藏二阶变异体：

$$\exists \mathrm{mi} \in M_{all}, \forall \vec{t} \in TS, \text{使得：} \begin{cases} \vec{d}(\vec{t}, po, mi(1, po)) = \langle 1,0, \ldots, 1 \rangle \\ \vec{d}(\vec{t}, po, mi(po, 2)) = \langle 1,0, \ldots, 1 \rangle \\ \vec{d}(\vec{t}, po, mi(1,2)) = \langle 0,0, \ldots, 0 \rangle \end{cases}$$

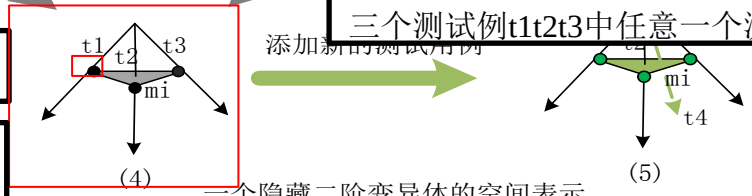
5.理论分析(2)

变异体被测试用例杀死



某二阶变异体在只有一个变异点被杀死时,该二阶变异体能够被三个测试例t1t2t3中任意一个测试例杀死;

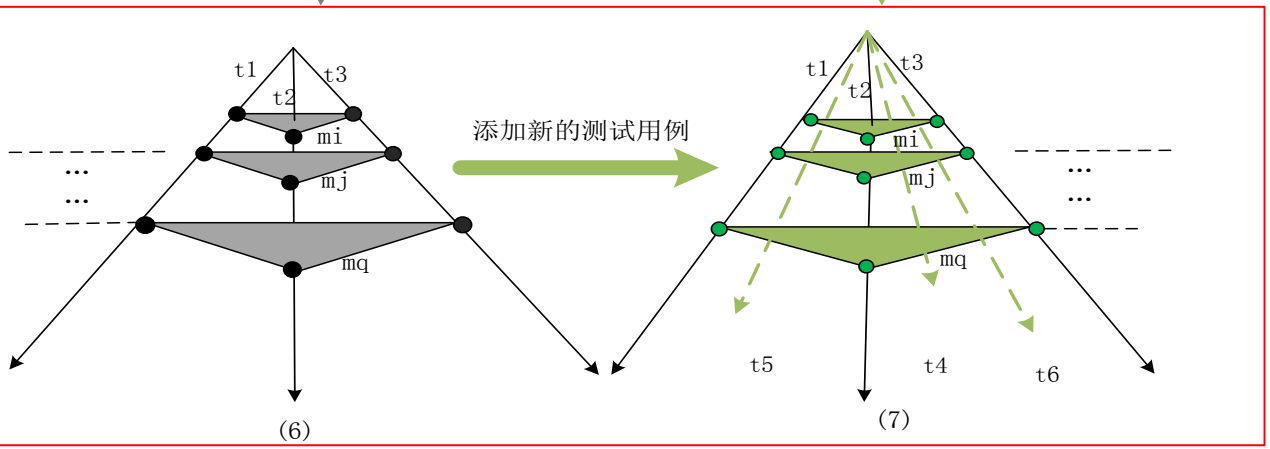
变异体没有被杀死



一个隐藏二阶变异体的空间表示

如果两个变异点均发生变异后,
该二阶变异体反而不能被三个测试用例杀死

推广到二阶变异体中有多个隐藏二阶变异体,可以针对不同隐藏二阶变异体设计新的测试用例,使得该隐藏二阶变异体被杀死,用以提高测试组件的质量.



所有隐藏二阶变异体的空间表示　　为所有隐藏二阶变异体添加新的测试组件的空间表示

5.理论分析

通过使用数学理论框架对本文所提方法进行的分析和理论证明，使得本文所提方法的核心思想更加的确定和易理解。更重要的是,使用框架中的元素对本文中的方法进行了推导证明，证明了方法的正确性。

主要内容

1.研究背景

2.相关工作

3.研究目标

4.研究方法

5.理论分析

6.实验及结果分析

6.实验及结果分析

程序

基准程序----有效性，三角形(Triangle) 、 找中间数(Mid)

实用程序----可应用性，贪吃蛇（Snake）、在线购物（Shopping）

变异算子列表

变异算子	全称	包含的操作符	示例
AOR	Arithmetic Operator Replacement	+,-,*,/,%	a+b => a-b
AOIS	Arithmetic Operator Insertion (Short-cut)	++,--	a => a++
AOIU	Arithmetic Operator Insertion (Unary)	-	a => -a
LOI	Logical Operator Insertion	~	b==a => b==~a
ROR	Rational Operator Replacement	>,<,>=,<=,≠	a>b => a<b
COI	Conditional Operator Insertion	!	b<a => !(b<a)
COR	Conditional Operator Replacement	&, |,	a&&b => a||b

6.实验及结果分析

(1) 所提方法能否减少二阶变异体个数?
(2) 所提方法对变异充分度是否影响?
(3) 所提方法能否降低变异测试时间开销?
(4) 所提方法能否发现隐藏二阶变异体?

6.实验及结果分析

(1) 所提方法能否减少二阶变异体个数?

与文献方法[3](记为已有方法)的二阶变异体个数进行比较， 具体如下:

Mid 约简后的二阶变异体

方法	测试例	约简后二阶变异体个数	减少率（%）
已有方法	t1,t2,t3	20	99.86
本文方法	t1	17	99.93
	t2	19	99.92
	t3	15	99.94

Triangle约简后的二阶变异体

方法	测试例	约简后二阶变异体个数	减少率（%）
已有方法	t1,t2,t3	52	99.89
本文方法	t1	19	99.97
	t2	25	99.96
	t3	23	99.96

与已有方法比,本文方法执行了更少的二阶变异体,

如以Mid为例,已有方法减少了99.86%,而本文方法的减少率至少为99.92%.

表明基于SOM神经网络的二阶变异体聚类能够有效减少二阶变异体个数.

6.实验及结果分析

(1) 所提方法能否减少二阶变异体个数?

为了验证结论是否有效，本文对本文方法和已有方法的筛选后减少率做了差异显著性分析(无重复双因素分析，其中a=0.05)。

基准程序的筛选后减少率的差异显著性分析

差异源	F	P-value	F crit
Mid筛选前后减少率组间	147	0.006743	18.51282
Triangle筛选前后减少率组间	484	0.00206	18.51282

说明两组数据具备显著性差异的可能性大于99%

表明本文方法和已有方法的减少率差异极为显著

6.实验及结果分析

(2) 所提方法对变异充分度是否影响?

首先进行一阶和二阶变异充分度比较，接着和文献方法进行变异充分度的比较,来反映本文方法的优劣性。

基准程序的一阶和二阶的变异充分度

程序	方法	一阶变异充分度	二阶变异充分度
Triangle	已有方法	92.9	91.7
	本文方法	92.9	92.5
Mid	已有方法	100	100.0
	本文方法	100	100.0

本文二阶变异充分度计算方法为:

假设100个非等价二阶变异体,聚类后共有20个簇.

若10个簇的代表性二阶变异体被杀死,

而10个簇中共含有60个二阶变异体,则变异充分度为60/100=60%.

的越好,

6.实验及结果分析

(2) 所提方法对变异充分度是否影响?

使用规模较大的开源项目Snake（2160行）和小组团队开发程序shopping（1560行）进行可应用性分析的实验。

实用程序的一阶和二阶变异体个数

程序	核心类	一阶变异体	二阶变异体
Snake	Obstacle,Foodset,CreatNode,SnakeNode,Move	131	10414
	Windows	15	
Shopping	ProductAction, UserAction, AdminAction	91	12126
	ProductEntity, UserEntity, AdminEntity	84	

实用程序的一阶和二阶的变异充分度

程序	一阶变异体个数	筛选前二阶变异体个数	一阶变异充分度	二阶变异分度
Snake	146	10414	13.68	13.73
Shopping	175	12126	14.76	14.69

6.实验及结果分析

(2) 所提方法对变异充分度是否影响?

实用程序的一阶和二阶变异充分度的显著性差异

差异源	F	P-value	F crit
组间	0.27778	0.894863	161.4476

说明两组数据具备显著性差异的可能性小于95%.

表明本文方法的筛选后二阶变异充分度较一阶变异充分度更为接近

6.实验及结果分析

(3-4) 所提方法能否降低变异测试时间开销和发现隐藏二阶变异体?

因为发现隐藏二阶变异体的实验需知道全部二阶变异体执行过测试例的信息，信息可从聚类后根据符合该特征的簇中获取。基于该特征，将发现隐藏二阶变异体的实验时间和约简变异体的时间开销同时分析。

平均时间消耗

方法	二阶变异体生成时间	聚类时间	测试例运行时间	发现隐藏二阶变异体时间	总时间
文献法	无	无	无	39685.000	39685.000
本文方法	756.826	38010.099	0.067	80.900	38847.892

聚类花费时间较长，但聚类后的二阶变异体运行测试例的时间很短；

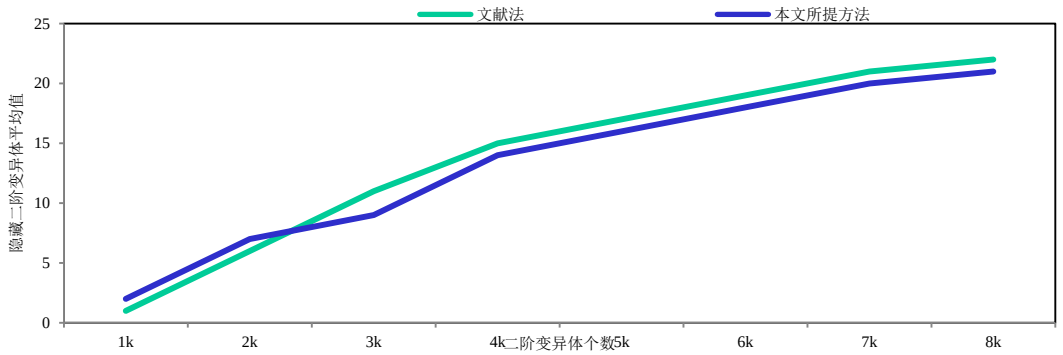
所提方法在发现隐藏二阶变异体功能上，较该文献方法中的时间开销减少了一些。

说明文本方法在能够减少程序执行开销且时间较少。

6.实验及结果分析

(3-4) 所提方法能否降低变异测试时间开销和发现隐藏二阶变异体?

文献方法和本文方法获取到的隐藏二阶变异体的平均数



本文方法寻找的到隐藏二阶变异体个数的平均数在总数不是很大时优于该文献；当数据量增大时与该文献方法几乎一样，稍微偏低，原因为数量较大时，根据聚类后的二阶变异体，提取出第一变异体被杀死、第二变异体被杀死，但是二者共同作用时未被杀死的二阶变异体数有少许误差。

总结与展望

总结

- 本文方法中的组合策略充分模拟了软件中的复杂错误，根据对二阶变异体执行过程中的中间值进行基于SOM神经网络的聚类，保证了二阶变异充分度与一阶变异充分度较为接近的情况下减少了二阶变异测试的执行开销；发现了隐藏二阶变异体。

展望

- 二阶变异测试是高阶变异测试的基础，所以该方法可以推广到其他的高阶变异测试。当推广到不同的阶数时,需要另外增加特征属性以达到更好的聚类效果。
- 设计更多可能的变异算子，如枚举。

参考文献

[1] Ghiduk AS, Girgis MR, Shehata MH, Higher order mutation testing: A systematic literature review, Computer Science Review, 2017,25:29-48

[2] Ma YS, Kim SW, Mutation testing cost reduction by clustering overlapped mutants, Journal of Systems and Software 115 (2016) 18-30.

[3] Ghiduk AS, Reducing the number of higher-order mutants with the aid of data flow, E-Informatica Software Engineering Journal 10 (2016) 31-49.

[4] Kohonen T. The self-organizing map, Proc. IEEE 1990,pp.1464-1480.

[5] Delgado S, Higuera C, Calle-Espinosa J, Morn F, Montero F, A som prototype-based cluster analysis methodology, Expert Systems with Applications, 2017,88:14-28.

[6] D. Shin and D. H. Bae, "A Theoretical Framework for Understanding Mutation-Based Testing Methods," 2016 IEEE International Conference on Software Testing, Verification and Validation (ICST), Chicago, IL, 2016, pp. 299-308.

参考文献

[7]Papadakis M, Malevris N. Mutation based test case generation via a path selection strategy. Information and Software Technology,2012,54(9):915-312.

[8] 党向盈,巩敦卫,姚香娟. 基于统计分析的弱变异测试可执行路径生成[J]. 计算机学报,2016,39(11):2355-2371.

[9] Jia Y, Harman M. Constructing subtle faults using higher order mutation testing. In: Cordy JR, Zhang L, eds. Proc. of the 8th Int'l Working Conf. on Source Code Analysis and Manipulation. Washington: IEEE Computer Society, 2008. 249-258.

[10]Kim,S.W.,Ma,Y.S.,Kwon,Y.R. Combining weak and strong mutation for a noninterpretive Java mutation system.J.Softw.Test.Verif.Reliab. 2013,23(8),647–668.

谢谢！