计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2021.20190817
58(1): 83-97, 2021

基于故障检测上下文的等价变异体识别算法

于 畅 王雅文 林 欢 宫云战
(网络与交换技术国家重点实验室(北京邮电大学) 北京 100876)
(shuoxunyc@bupt.edu.cn)

Fault Detection Context Based Equivalent Mutant Identification Algorithm

Yu Chang, Wang Yawen, Lin Huan, and Gong Yunzhan
(*State Key Laboratory of Networking and Switching Technology* (*Beijing University of Posts and Telecommunications*), *Beijing* 100876)

Abstract Although studied for almost forty years, the mutation testing has been prevented from being widely applied in industrial practice by the problem of equivalent mutants. To overcome the problem, a algorithm of using fault detection context to predict the equivalence of mutants is proposed. It makes use of static analysis technique to extract feature information about the program context around mutated program, which is called its fault detection context. Then the context information is translated into a document model, which describes the feature of mutant using natural language. The representation learning network is further used to encode fault context features. Finally, machine learning model is used to predict the equivalence of each mutant with respect to its fault detection context. An empirical study on 118000 mutants from 22 C programs is performed to validate the proposed method. The results show that the method achieves 91% of precision and 82% of recall in classifying mutants as equivalent, while 77% of precision and 78% of recall are achieved in cross-project validation. It implies the fault detection context based technique can dramatically improve the efficiency and effectiveness of equivalent mutants detection, which effectively facilitates the efficiency for mutation testing process.

Key words mutation testing; equivalent mutant; fault detection context; machine learning; static analysis

摘 要 等价变异体识别一直是阻碍变异测试在工业界得以广泛应用的一个关键难题.为此提出了一种基于故障检测上下文的等价变异体识别算法.该算法通过静态分析技术抽取程序中与故障检测条件相关的代码上下文信息,以构造故障检测上下文;接着,故障检测上下文被转换为文档模型,经过一个文档表示学习网络进行编码;最后通过机器学习模型将变异体分类为等价或非等价变异.在包含了22个C程序和118000个变异体样本的训练集上,该算法取得91%的分类精准度和82%的召回率;同时在跨项目交叉验证中,机器学习模型取得了77%的精准度和78%的召回率.该结果表明基于故障检测上下文的识别技术能够有效地提高等价变异体分类的精准性和泛用性,为提高变异测试技术的有效性提供了技术支持.

收稿日期:2019-11-28;**修回日期**:2020-07-13
基金项目:国家自然科学基金项目(U1736110,61702044)
This work was supported by the National Natural Science Foundation of China(U1736110, 61702044).
通信作者:王雅文(wangyawen@bupt.edu.cn)

关键词 变异测试;等价变异体;故障检测上下文;机器学习;静态分析

中图法分类号 TP311

变异测试(mutation testing)是一种基于故障的软件测试分析方法[1],它通过向被测软件注入一组人工故障,以模拟软件开发过程中引入的代码缺陷[2].这些故障被进一步用于评估测试充分性[3-5],并辅助测试人员开发测试用例以提高测试质量[6-8].相关研究[9-10]表明,相较结构化覆盖准则[11],该技术具有更强的检错能力.

在变异测试中,通过修改被测程序源代码的语法结构注入故障,被修改后的故障程序称为变异体,而代码修改规则称为变异算子.例如,关系运算符替换算子(relational operator replacement, ROR)对关系表达式 $x<y$ 进行替换,生成包括 $x\leqslant y$,$x>y$,$x\geqslant y$,$x==y$ 以及 $x\neq y$ 共 5 个变异体.在变异测试中,如果 1 个变异体 m 与被测程序 p 在测试输入 x 上的输出不同,则称变异体 m 被测试输入 x 杀死.

尽管变异测试能为被测程序生成大量变异体,然而并非所有变异体对测试数据质量的提高都有帮助[12-13].Ammann 等人[14-18]的研究表明:现有的变异测试工具会生成超过 80%的无效变异体.这些无效变异体不仅会增加测试成本和执行时间,同时还会降低变异分析的有效性,降低测试质量.其中一类对变异测试影响较大的无效变异体是等价变异体[19].

等价变异体指的是与原程序保持语义等价的变异体程序[20].这类变异体无法被任何的测试输入杀死,既不能有效地模拟故障缺陷,也无法改进测试数据充分性.因此在测试前,需要将程序中的等价变异体识别并移除.图 1 展示了一个示例程序以说明变异体的等价性:mutant-1 和 mutant-2 将运算符“<”替换为“≤”生成故障程序.其中 mutant-1 是非等价变异体:它会导致数组下标 k 越界从而造成程序异常退出.而 mutant-2 是等价变异体:该变异体总是输出数组的最小值,与原程序的输出完全一致,因而 mutant-2 无法被任意的测试用例杀死.

```
int get_min(int list[]){
    int min=Integer.Maximal;
    int length=list.Length;
    for(int k=0;k<length;k++)
    /* mutant-1:k≤length */
        if(list[k]<min)
        /* mutant-2:list[k]≤min */
            min=list[k];
    return min;
}
```

Fig. 1 Equivalent mutant and non-equivalent mutant

图 1 等价变异体与非等价变异体示例

长期以来,等价变异体识别是研究领域和工业界所面临的一个主要难题.一方面,通过人工审查判断变异体的等价性平均需要 6~15 min[21-23].另一方面,现有技术[24-32]无法有效识别等价变异体,存在 2 方面不足:

1) 准确度低.现有技术使用不完整的故障特征来推断变异体的等价性,导致识别技术产生大量的错分类样例[24-28].

2) 扩展性差.以往的研究方法依赖于人工设计的推理规则,这些规则只能识别少量的等价性模式,如不可达路径或数据流模式等.对于实践中复杂多样的等价性模式,这类技术的可扩展性将受到限制[33].

为了能提高识别的精准性和有效性,本文提出了基于故障检测上下文的等价变异体识别算法.该算法通过抽取变异体的故障检测上下文作为特征,并通过机器学习对复杂特征进行分析,从而实现等价变异体的自动分类和识别.故障检测上下文包含了与故障检测过程相关的程序切片.使用机器学习技术实现等价变异识别是基于 3 方面的考虑:

1) 等价变异体识别存在数据量大、故障特征复杂的问题.传统的逻辑推理技术对复杂故障的分析能力有限;而统计学习技术为复杂特征的大数据信息提供了有效的分析手段.

2) 统计学习的分类准确度会随着训练样本的增加而不断改进,从而增强了等价变异体识别的有效性和方法的扩展性.

3) 现有的机器学习研究提供了大量成熟的机器学习模型库,为本文快速和有效地实现等价变异体分类提供了技术上的保障.

我们将本文提出的方法应用于 22 个 C 程序的共计 118 000 个变异体进行评估.实验结果表明:

1) 在 5-折交叉验证中,基于故障检测上下文的等价变异识别算法取得 93%的预测准确率(accuracy).

2) 在跨工程间验证中,基于故障检测上下文的技术取得 77%以上的分类精准度(precision)和 78%的召回率(recall).

1　相关工作

长期以来,等价变异识别问题不仅是阻碍变异测试被工业界广泛应用的关键原因;同时也是变异测试领域的主要难题.造成该问题未被攻克的原因有:

1) 等价变异体数量巨大.根据 Yao 等人[30]的实验研究数据,等价变异体的数量占变异体总量的10%～15%.

2) 人工等价性检测耗时.Schuler 等人[21]的研究报告指出,人工判断变异体等价性平均需要 6～15 min.

3) 自动识别等价变异异常困难.研究者们已经证明,等价变异检测是不可解问题.这意味着对任一被测程序 p 和变异体 m,不存在算法 A 能准确地判断 m 的等价性[20].

一方面,通过人工手段从大量样本中确认等价变异体是不切实际的;另一方面,由于等价变异体识别问题的不可判定性,使得到目前为止,尚不存在能有效检测等价变异体的技术工具.因此,在实践中,研究者们只能近似地解决该问题:要么为特定变异算子设计相对精准的等价变异体识别规则,要么设计一个面向通用故障类型的近似算法.前者可以取得较高的检测精度,但是可扩展性较差;而后者虽然适用于大部分实践中的被测软件,但是精准度较差.为了开发有效的等价变异体识别算法,Madeyski 等人[19]为该算法提出了 2 方面的需求:

1) 精确性需求.要求通过算法 A 检测出的变异体均为等价变异体.令 D 为算法 A 识别的等价变异体集,E 为实际的等价变异体集.精确性要求算法 A 具有高精确率 P,其中 P 的定义为

$$P=\frac{|D\cap E|}{|D|}. \tag{1}$$

2) 可用性需求.要求被测程序的大部分等价变异体都被算法 A 检测出来.换言之,要求算法 A 具有高召回率 R,其中 R 的定义为

$$R=\frac{|D\cap E|}{|E|}. \tag{2}$$

Madeyski 等人[19]对 30 年来等价变异体相关工作进行了系统性的分类和总结,将现有的等价变异体识别技术分为 5 组,分别是:

1) 基于编译优化的等价变异检测方法[27-28];

2) 基于程序约束的等价变异检测方法[24-26];

3) 基于程序影响的等价变异体选择方法[21-22];

4) 基于层次关系的等价变异体优化方法[15-16];

5) 基于变异算子的等价变异体选择方法[29-30].

其中,基于编译优化和程序约束的等价变异体检测算法通过形式化验证和程序证明技术,严格论证程序间的等价关系,具有较高的精准性.然而,由于形式验证通常依赖于人工设计的推理规则,使得这类方法通常适用于识别复杂度较低的等价变异体;对于复杂的等价性模式,该算法的召回率较低.

此外,基于程序影响和变异算子的等价变异体选择算法通过经验研究和统计分析等手段,从大量的样例中挖掘等价变异体的算子和程序影响模式.然而,现有技术依赖于人工特征建模,提取的特征相对单一,难以确保挖掘到的等价变异体特征是否适用于更复杂的被测程序,其精准度和泛用性受到限制[19].

表 1 总结了现有等价变异体识别算法的精确率、召回率及其存在的缺陷.可以发现,现有的等价变异检测技术无法同时取得较高的精确率和召回率.为了提高等价变异体识别的精准性和有效性,本文将提出基于故障检测上下文的等价变异体识别算法.

Table 1　Existing Equivalent Mutant Detection Algorithms

表 1　现有的等价变异体识别算法

算法	精确率/%	召回率/%	缺陷
基于编译优化	100	10-30	召回率低
基于约束求解	100	8-15	
基于程序影响	15	45	精确率低 扩展性差
基于层次关系			
基于编译算子	5-15	70	

2　故障检测上下文

2.1　故障检测上下文与等价变异体

在软件测试中,故障检测条件描述了测试用例为杀死变异体需要满足的约束,包括[20]:

1) 覆盖约束.故障所在的语句被测试用例覆盖.

2) 触发约束.故障语句的执行触发运行时错误.

3) 传播约束.运行时错误传播并改变程序输出.

给定变异体 m 和被测程序 p,令 C_{R},C_{I},C_{P} 表示变异体 m 的覆盖、触发和传播约束,则变异体 m 的检测条件可以描述为

$$C_{\mathrm{R}}\wedge C_{\mathrm{I}}\wedge C_{\mathrm{P}}. \tag{3}$$

基于约束的等价变异体识别算法[25]通过验证该约束条件的可满足性来判断变异等价性.在实践

中,为了提取故障检测条件,开发者需要通过静态分析工具,从源码中提取与检测条件相关的代码来推断变异体的等价性.本文将与检测条件相关的程序切片称为故障检测上下文(fault detection context, FDC).FDC 为判断故障的可检测性提供了证据信息,是识别等价变异体的前提条件[26].

图 2 展示了图 1 中 mutant-1 和 mutant-2 的故障检测上下文.其中,mutant-1 的检测上下文不包括语句 $min = Integer.Maximal$ 和 return min;而 mutant-2 的上下文移除了无关语句 $length = list.Length$.

```
/* detection context of mutant-1 */
...
int length=list.Length;
for (int k=0;k<length;k++)
    /* k≤length */
    if (list[k]<min){
        ...
    }
...
```

```
/* detection context of mutant-2 */
int min=Integer.Maximal;
...
for (int k=0;k<length;k++)
    if (list[k]<min){
    /* list[k]≤min */
        min=list[k];
    }
return min.
```

Fig. 2 Fault detection context for mutants

图 2 故障检测上下文案例

通过分析 mutant-1 的 FDC 中 for 语句 $k=0$, $k++$可以证明,变异体 $k \leqslant length$ 必然在 $k == length$ 时引发运行时错误,执行循环语句体;同时,通过分析循环体的 $list[k] < min$ 和 $length = list.Length$ 可知,该变异体必然引发 $list[k]$下标越界,造成内存泄漏以及系统崩溃.因此 mutant-1 不是等价变异体.

与此同时,通过分析 mutant-2 的 FDC 可以发现,当 $list[k] == min$ 时,虽然故障条件 $list[k] \leqslant min$ 会错误地执行赋值语句 $min = list[k]$,但是该赋值对最终结果没有影响,故 mutant-2 是等价变异体.

在上面的分析中,故障检测上下文为推断变异体的等价性提供了证据信息.对任一等价变异体识别技术而言,一个重要的技术瓶颈是如何自动地抽取故障检测上下文,以提高识别准确率.本节将提供 FDC 的形式定义,以支持故障检测上下文的自动抽取.

2.2 故障检测上下文与程序依赖图

故障检测上下文是故障注入点的依赖子图.下面回顾程序依赖图[34-36]的定义.

定义 1. 控制流程图.令 p 为被测程序,p 的控制流图(control flow graph, CFG)是一个多元组,记作 $CFG=(p,N,E,entry,exit)$.其中,N 为程序 p 的语句集,表示控制流图的节点;E 为控制流图的有向边,满足 $(x,y)\in E$ 当且仅当语句 y 可能在 x 后被执行;$entry$ 和 $exit$ 分别是程序入口和出口节点.

定义 2. 控制支配关系.给定程序 p 的控制流图 CFG,语句 x 被语句 y 后向支配,当且仅当所有从 x 到出口 $exit$ 的路径都经过 y,意味着 y 是 x 到程序出口的必经点.

定义 3. 控制依赖关系.令 x 和 y 是程序 p 的 2 条语句,定义 y 控制依赖于 x,记作 $y \rightarrow_{C} x$,当且仅当:1)存在 1 条从 x 到 y 的路径,其中 y 后向支配路径上除 x 外的所有语句;2)x 不被 y 后向支配.其中,控制依赖关系满足传递性,有

$$(x \rightarrow_{C} y) \wedge (y \rightarrow_{C} z) \Rightarrow (x \rightarrow_{C} z). \tag{4}$$

定义 4. 直接控制依赖关系.给定语句 x 和 y,称 y 直接控制依赖于 x,记作 $y \rightarrow_{DC} x$,当且仅当 $y \rightarrow_{C} x$,且不存在其他语句 z 使 $y \rightarrow_{C} z$ 且 $z \rightarrow_{C} x$.在结构化程序中,直接控制依赖关系 $y \rightarrow_{DC} x$ 意味着 x 往往是 1 个分支条件,而 y 是该条件的真分支或假分支上必经的某条语句.

定义 5. 数据依赖关系.令 $U(s)$表示语句 s 中使用的变量集,$D(t)$表示语句 t 中被定义(或赋值)的变量集;定义直接数据依赖 $s \rightarrow_{DD[x]} t$,使得存在 $x \in U(s)$且 $x \in D(t)$,且存在 1 条从 t 到 s 的路径,使得该路径上没有其他语句对变量 x 进行再赋值.其中,直接数据依赖关系表示变量 x 在语句 s 中的使用值是在语句 t 中被定义的.

定义 6. 程序依赖图(program dependence graph, PDG).PDG 是一个有向图 $G=(N,E)$,其中 N 为程序语句集,关系集 E 包括:

1) 控制依赖边(s,c,b),c 为分支条件而 b 为布尔值,表示从 c 到 s 是真分支还是假分支;

2) 数据依赖边(s,t,x),其中 s 通过使用变量 x 直接数据依赖于 t.

图 3 展示了图 1 中的案例程序的程序依赖图.其中,分支条件 $k < length$ 数据依赖于 $k=0$ 和

k++语句,因为前者使用了后者中被赋值的变量 k.此外,赋值语句 $min=list[k]$ 控制依赖于分支 $list[k]<min$,且该语句在分支 $list[k]<min$ 为真值时被执行.

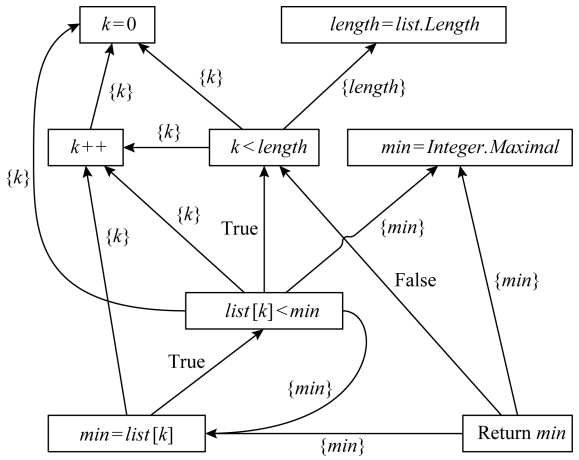

Fig. 3　Program dependence graph

图 3　程序依赖图示例

基于程序依赖图模型的概念,下面引入 k 阶依赖子图.

定义 7. 控制依赖子图.给定程序依赖图 G 以及语句 s,定义语句 s 的 k 阶依赖子图($k\geqslant 0$)是 G 的一个子图,记作 $G_{s,k}=(s,N_{s,k},E_{s,k})$,其中 $N_{s,k}$ 和 $E_{s,k}$ 表示子图的节点集和有向边集,它们满足

$$N_{s,k}=\{t\mid \exists e\in E_{s,k},t\in e\},\tag{5}$$

$$E_{s,k}=\{e\mid \exists p\in P_{s,k},e\in p\},\tag{6}$$

其中 $P_{s,k}$ 是依赖图 G 的一个简单路径子集,满足

$$P_{s,k}=\{p_{s,t}\mid \|p_{s,t}\|\leqslant k\}\cup\{p_{t,s}\mid \|p_{t,s}\|\leqslant k\}.\tag{7}$$

在定义 7 中,$p_{s,t}$ 是程序依赖图中一条从 s 到 t 的简单路径;该路径不包含重复的有向边.从定义 7 不难发现,路径集 $P_{s,k}$ 包含了所有从 s 出发或者到达 s 的长度不大于 k 的简单路集.而 k 阶依赖子图包含依赖图中以 s 为中心、距离为 k 以内的节点和边.作为例子,考虑图 4 中 $list[k]<min$ 的一阶依赖子图.

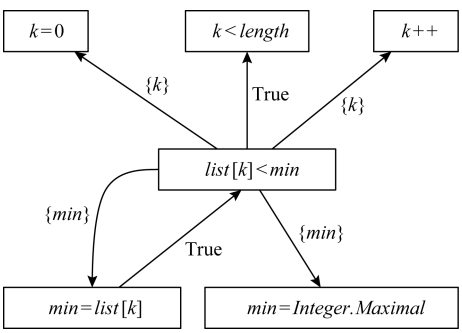

Fig. 4　First-order dependence subgraph example

图 4　一阶程序依赖子图示例

在图 4 中故障语句 $list[k]<min$ 的依赖子图包含了 3 组依赖路径,分别是:$list[k]<min$ 到 $k<length$ 的控制依赖关系,其中前者是否被覆盖取决于后者的真值,该依赖关系与故障的覆盖约束相关;其次,从 $list[k]<min$ 到 $k=0$,k++,$min=list[k]$ 和 $min=Integer.Maximal$ 的数据依赖关系,这些关系决定了条件语句中变量 $list[k]$ 和 min 的取值,与故障的触发约束相关;最后,从 $min=list[k]$ 到 $list[k]<min$ 的控制依赖关系,揭示了受其影响的语句,与故障的传播约束相关.

从上述分析不难看出,k 阶依赖子图能自动识别与检测条件相关的语句及依赖关系,因此本文将故障语句的 k 阶依赖子图定义为故障检测上下文.

定义 8. 故障检测上下文.令 $oprt$ 为变异算子,s 为变异体 m 的注入语句,m 的故障检测上下文定义为二元组 $FDC_m=(oprt,G_{s,k})$,其中 $G_{s,k}$ 是以 s 为中心的 k 阶依赖子图.

2.3　故障检测上下文与依赖链长度

在定义 8 中,非负整数 k 是一个由用户指定的超参数,关于 k 的选择将会影响 FDC_m 的特征有效性.当 k 太小时,FDC_m 表示力不足;而 k 太大时,FDC_m 中将包含大量的依赖路径链,其中只有极少部分被用于等价变异体判定,导致 FDC_m 包含了太多噪音信息,进而降低算法有效性.

图 5 通过图 1 中 mutant-1 的二阶依赖子图说明了这一问题.其中灰色节点为论证变异体的等价性提供了证据信息,其他无色节点为没有使用到的信息.

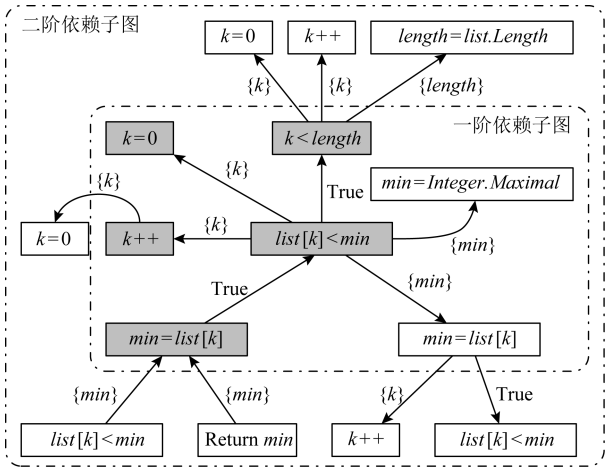

Fig. 5　Second-order dependence graph example

图 5　二阶依赖子图示例

不难发现,大部分对等价变异体识别过程有帮助的语句和依赖关系与故障注入点的距离都比较近,都在一阶依赖子图范围内.为了降低特征表示的复杂度以及分析的有效性,在实验评估中本文限制 $k=2$.

3 等价变异体识别算法

故障检测上下文特征为推断变异体等价性提供了充分的分析信息.因此理论上可以通过逻辑推理和形式验证技术,在 FDC 的基础上实现等价变异体检测.然而,逻辑推理技术强烈地依赖于人工设计的推导规则;这些规则通常只能通过简单的上下文模式识别等价变异体,对复杂模式,其可扩展性将受到限制.

为了能在故障检测上下文的基础上有效地识别等价变异,本文提出采用统计学习技术实现基于 FDC 的等价变异体识别算法.其理由为:首先,等价变异体存在数量大、特征复杂等问题,而统计学习技术为分析复杂特征的大数据信息提供了有效的分析手段;其次,统计学习的分类准确度会随着训练样本的增加而不断改进,增强了识别方法的可用性和扩展性;最后,现有的机器学习研究提供了大量成熟的机器学习模型库,为本文快速和有效地实现等价变异体分类提供了技术上的支持.

3.1 算法流程

图 6 展现了该算法的流程图,包括训练样本标签化、故障特征提取、故障特性文档化、训练分类器以及变异体分类 5 个子过程.

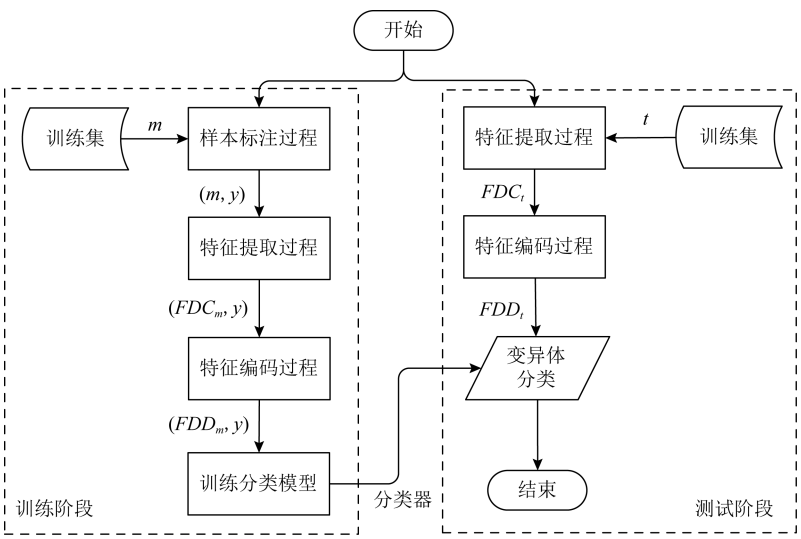

Fig. 6 FDC-based equivalent mutant detection flowchat

图 6 基于检测上下文的等价变异体识别算法流程

样本标签化过程通过人工辅助的软件测试手段,识别训练样本中的每一个变异体 m 是否为等价变异体,输出是一个带标签的二元组 (m,y),其中 y 为布尔值,表示 m 是否为等价变异体.

故障特征提取过程通过程序依赖分析提取变异体依赖子图,作为变异体 m 故障检测上下文 FDC_m.接着,故障特征文档化过程将结构化的故障检测上下文转换为故障检测文档(fault detection document, FDD),该文档是一个以自然文本描述故障检测上下文的语句集,记作 FDD_m.该文档将结构化的程序分析问题转换为文本分类问题,并使用现成的统计学习模型进行分析,将变异体分类为等价或非等价.

接下来将解释故障检测文档的模型定义、故障检测上下文到故障检测文档的转换规则,以及统计学习技术如何使用故障文档模型识别等价变异体.

3.2 故障检测文档

故障检测文档是由句子(sentence)构成的集合

$$FDD_m=\{S_1,S_2,\cdots,S_D\}, \tag{8}$$

其中,每个句子 S_i 是一个由单词(word)构成的不定长序列,用于描述 k 阶依赖子图中一条以故障注入点 s 为起点或终点的依赖关系链,表示为

$$S_i=(w_{i,1},w_{i,2},\cdots,w_{i,n-1},w_{i,n}). \tag{9}$$

在该定义中序列 S_i 的一个单词 $w_{i,j}$ 表示依赖关系链上的一个节点或节点之间的关系.其中,节点单词描述节点的抽象语法树类型,而关系单词描述语句与语句、语句与变量的关系.表 2 列出了一部分节点和关系的对应描述单词.

Table 2　Objects in FDD and Words for Describing
表 2　故障检测文档语句部分描述对象及其单词

描述对象	单词
ROR-operator	(<,≤)或(<,≠)
x,y,z(int)	id_expr
x && y	logic_and
if E then S_1 else S_2	if_statement
x in$\{y=x\}$	[x] used_in [$y=x$]
y in$\{y=x\}$	[y] defined [$y=x$]
C in{if C then S}	[C] condition_of [if C then S]
$s\rightarrow_{DC} t$	[s] in_branch [t]
$s\rightarrow_{DC} t$	[t] lead_to [s]
$s\rightarrow_{DD} t$	[s] use_from [t]
$s\rightarrow_{DD} t$	[t] define_for [s]

作为例子,考虑图 5 中的 3 条依赖关系路径,分别是 $list[k]<min$ 到 $k++$ 的依赖链以及 Return min 到 $list[k]<min$ 的依赖链.图 7 将上述依赖链的节点及对应词组罗列出来,并描述了每条路径的文本表示.其中,各个节点和依赖边都有对应的单词.

以第 1 条依赖路径为例,路径的起始点对应变异体,其单词(<,≤)表示变异算子;接着,变异节点连接到故障注入点 $list[k]<min$ 的边对应单词 seeded_in,表示右侧节点是变异算子植入的位置.故障注入点是一个关系表达式,其对应的单词为 relation_expr_<;相应地,该节点到下一个节点的控制依赖关系将通过单词 in_branch_true 描述,意味着只有右侧节点分支取真时,左侧节点才会被执行.类似地,从 $k<length$ 到 $k++$ 的数据依赖之间用 1 个节点 k 表示依赖变量;数据依赖关系用 used_in 来描述;由于 k 是 $k<length$ 的左操作数,用 lop(left_operand)来描述两者间的关系.将上面单词连接起来得到词序列:(<,≤)seeded_in relation_expr_< in_branch_true relation_expr_< left_operand id_expr used_in increment.

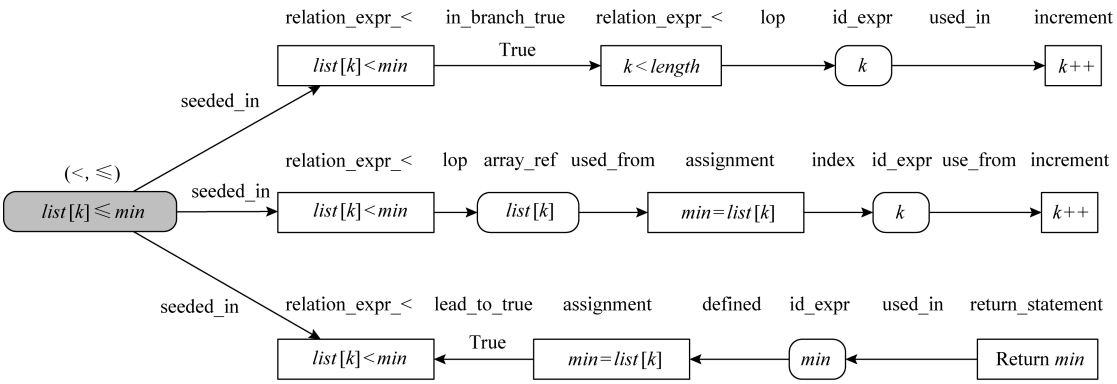

Fig. 7　Fault detection document example
图 7　变异体故障检测文档示例

使用故障检测文档作为故障特征输入的原因有 2 方面:

首先,从机器学习模型的角度出发,序列化的文档模型适用于大部分现有的且成熟的机器学习模型.这类模型对数据量和样本输入的鲁棒性更强;而以结构化特征为输入的模型如神经网络,对输入数据量的要求则更高.从实用性的角度,直接以结构化的故障检测上下文作为特征未必能取得比故障检测文档更好的分类效果.

其次,从等价变异体识别角度考虑,结构化的故障上下文特征要求使用抽象的推理规则和形式验证技术,使得等价变异分类器的可扩展性受到限制.而通过将故障检测上下文转换为故障检测文档,等价变异体识别问题被转换为了相对简单的文本分类问题,而后者提供了相对简单的推理规则,确保模型的简易和泛用性.

3.3　故障特征编码

大部分现有的机器学习模型,如贝叶斯网络、支持向量机、随机森林或多层感知器,都是以一个定长的实值向量作为特征输入;而在 3.2 节中引入的故障文档模型却包含了可变数量的语句和单词.为此,在训练分类器之前,需要将故障文档 FDD_m 转换为一个定长的实数向量 $\boldsymbol{X}_m$.故障检测文档的特征编码分为 3 个阶段,分别是预处理、句编码和文档编码.

3.3.1　预处理

在预处理阶段,故障检测文档中表示节点和边的单词会被重组,以构成表示具有完整语义信息的

关系组合词.具体而言,在一个依赖路径中,设 n_x 和 n_y 为相邻节点对应单词,$e_{x,y}$ 连接从 n_x 到 n_y 的依赖关系的对应单词,则 n_x 和 n_y 的关系组合词为

$$w_{x,y}=n_x \cdot e_{x,y} \cdot n_y, \tag{10}$$

其中'·'表示字符串连接操作.

对于一个给定的语句,其词序列为(n_1,$e_{1,2}$,n_2,$e_{2,3}$,n_3,…,$e_{r-1,r}$,n_r).经过预处理后,该序列的单词组合为关系词组,有

$$S_i=(w_{1,2},w_{2,3},\cdots,w_{k,k+1},\cdots,w_{r-1,r}), \tag{11}$$

其中 $w_{k,k+1}=n_k \cdot e_{k,k+1} \cdot n_{k+1}$ 表示关系组合词.

将邻接节点组合起来的思想是基于这一观察:在依赖路径中,单独表示节点和边的单词无法描述"依赖关系"的语义.而邻接点的关系组合词能表示依赖链中一个最基本的连接关系语义,这将有效地提升特征编码的表达能力.

3.3.2 句编码

在故障检测文档中,一个语句描述了从故障注入点出发(或终止)的一条完整依赖链;其中每个单词表示依赖链中的一条依赖边.对于由一组特定顺序构成的依赖链,它往往描述了特殊的故障检测上下文结构.如图 7 中第 3 行的依赖链描述了变异体对程序的影响方式,首先通过对变量 *min* 赋值,然后通过返回语句影响外部输出.

显然,每个语句都表示某种特定的"主题",如故障通过某种方式影响外部变量,或者通过某条特殊路径被覆盖.语句编码的目的是将这种主题语义从词序列中抽离出来,以向量的形式进行表示.这种向量被称为句向量(sentence vector)[37].

在自然语言处理领域 doc2vec[38]是一种自动抽取句向量的机器学习算法.该算法的基本思想是,在给定算法的部分单词的条件下,根据算法的主题语义向量,可以恢复出剩余空缺单词.

图 8 展示了 doc2vec 基于分布式存储(distributed memory)的实现.以图 7 中第 3 行语句为例,如果去除最后一个 *min* 到 return *min* 的依赖关系,使用 doc2vec 生成的句向量可以从其他依赖关系后恢复出该依赖关系,这意味着"通过返回语句影响程序"的语义信息已经被抽取到了该语句的句向量中.给定依赖路径的语句 S_i,doc2vec 算法将输入语句 S_i 转换为一个固定长度的句向量,记作 $\boldsymbol{D}_i$.

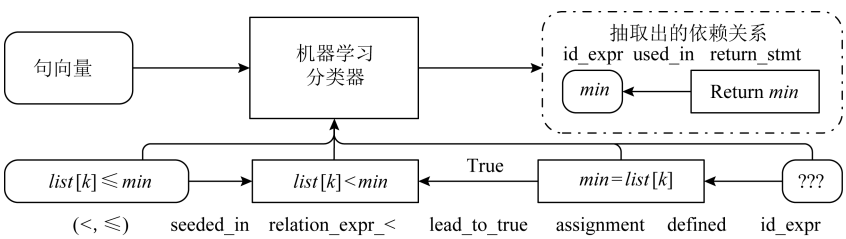

Fig. 8 Distributed memory model in doc2vec

图 8 doc2vec 算法基于公布式存储的实现

3.3.3 文档编码

在句向量编码完成后,故障检测文档被转换为由一组句向量 $\boldsymbol{D}_i$ 构成的集合,记作

$$FDD_m=\{\boldsymbol{D}_1,\boldsymbol{D}_2,\cdots,\boldsymbol{D}_n\}. \tag{12}$$

在等价变异体分析中,并非所有的语句(依赖链)都被用于论证变异体的等价性.正如 2.3 节图 5 的案例所展示的那样,故障检测上下文中,一部分依赖关系起着关键,而另一些依赖关系则几乎毫无帮助.基于这一观察,可以假设变异体的特征向量是其检测文档中句向量的加权和,满足

$$\boldsymbol{X}_m=\sum_{i=1}^{n}\alpha_i\boldsymbol{D}_i, \tag{13}$$

其中,参数 α_i 表示每个语句的权重,权重参数是非负实数,其总和为 1.在等价变异体识别过程中,权重越大的语句表示故障检测上下文中对论证变异体等价性越关键的依赖路径;反之,权重越小的语句表示对论证等价性帮助越小的路径;权重等于 0 表示该依赖路径对在论证等价变异体中没有帮助.

本文采用了基于注意力机制的全连接网络,实现对文档向量的表示学习.图 9 展示了这种基于注意力机制的表示学习网络的整体框架[39-40].

首先,故障检测文档的所有语句通过 doc2vec 算法映射到一组句向量,这组句向量大小是可变的.由于注意力权重向量通常是定长的(设为 n),这里采用截断策略:令故障文档包含 d 个句子,如果 $d<n$,则补充 $n-d$ 个全 0 向量作为伪特征;如果 $d>n$,只取前 n 个句向量,而忽略多余的 $d-n$ 个句子信息.

接着,通过一个注意力权重网络注意力权重计算每个句向量的权重 α_i.其中,注意力网络定义了一个

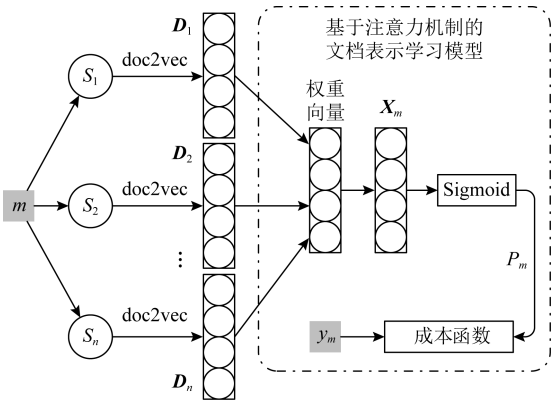

Fig. 9　Attention-based document embedding networks

图 9　基于注意力机制的文档表示学习网络

长度为 n 的注意力向量,记作 $\boldsymbol{\beta}$.对每个句向量 $\boldsymbol{D}_i$ 其权重计算为

$$\alpha_i=\frac{\exp(\boldsymbol{\beta}\cdot\boldsymbol{D}_i)}{\sum_{j=1}^{n}\exp(\boldsymbol{\beta}\cdot\boldsymbol{D}_j)}. \tag{14}$$

从式(14)不难看出,各个语句的权重总和为 1.

在完成了权重计算后,根据式(13)求出故障检测文档的特征向量 $\boldsymbol{X}_m$.最后 $\boldsymbol{X}_m$ 通过一个 Sigmoid 函数映射到概率 P_m.这一概率表示变异体 m 为等价变异体的概率.概率 P_m 被进一步与变异体的类别 label 进行比较,并计算成本函数:

$$-\frac{1}{N}\sum_{m}[y_m\times\mathrm{lb}(P_m)+(1-y_m)\times\mathrm{lb}(1-P_m)], \tag{15}$$

其中,N 表示训练样本总数,y_m 表示 m 的类别标签,$y_m=1$ 表示 m 为等价变异体.式(15)计算 N 个样例中被正确分类的训练样本比例,对应于分类精确率.在获得成本函数值之后,根据神经网络的反向传递算法,进一步优化注意力权重和 Sigmoid 网络参数,进而得到最优的文档表示学习网络.

3.4　机器学习分类

在获得了故障特征向量 $\boldsymbol{X}_m$ 后,这些故障特征被进一步用作一系列机器学习分类器的输入.在本文中,将会考虑贝叶斯、逻辑回归、决策树、随机森林、梯度上升、多层感知器等分类器以实现基于故障检测上下文的等价变异体识别技术.

4　与现有技术的比较

基于特征的变异体分类算法为评估故障特征以及等价变异体识别算法的有效性提供了分析框架.令被测程序 p 的变异体集合为 M.基于特征的等价变异体识别方法会为每个变异体 m 定义一个故障特征 x,所有这些特征的集合称为特征空间 F.定义 F 上的特征提取函数 $\Phi:M\to F$,使得函数 $\Phi(m)$ 返回 m 在 F 中的特征点 x.

给定变异体在 F 中的特征点 x,引入特征分类函数 $\Psi:F\to\{0,1\}$,该函数负责根据故障特征 x 判断其对应变异体是否为等价变异.$\Psi(x)=1$ 表示具有特征 x 的变异体为等价变异体.

通过组合特征提取函数 Φ 和等价分类函数 Ψ,一个等价变异体识别算法可以通过下面的复合函数抽象表示出来:

$$y=\Psi(\Phi(m)), \tag{16}$$

其中 y 为算法预测的变异体 m 的类别.

现有的等价变异体识别技术以及本文提出的基于故障检测上下文的识别算法都可以看作是基于特征的变异体分类算法的某种具体实现,如图 10 所示.

其中,基于选择变异的实现方案使用最单一的特征(变异算子)和最宽松的判定条件(相关性分析)判定等价变异体,极大地降低了分类的准确性.而基于约束求解的实现方法使用过度抽象和复杂的符号

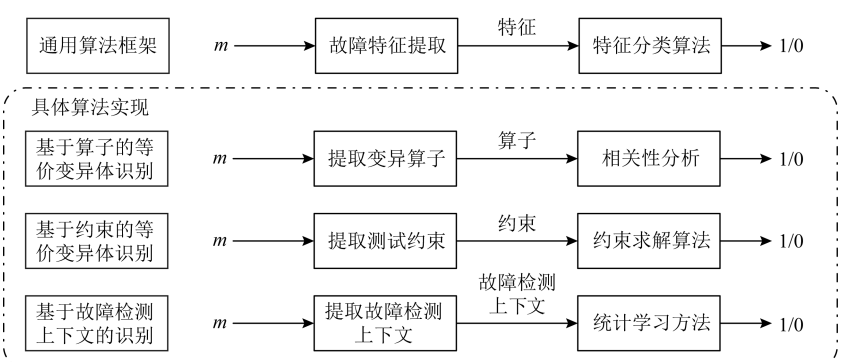

Fig. 10　Feature-based equivalent mutation detection

图 10　基于故障特征的等价变异体检测算法框架

表示作为特征,增加了分类算法实现的难度,极大地限制了算法的适用范围,降低了召回率.

相比之下,本文提出的等价变异体识别算法以故障检测上下文为输入,确保了故障特征的复杂性和信息量.同时,通过统计学习技术,自动化构建等价变异分类器,简化了算法实现,提高了可扩展性.

在现实中,研究者们往往希望近似地推断变异体的等价性.这意味着对于给定的被测程序及其变异体样本集 M,能最大化其在样本集上的识别准确度($accuracy$).$accuracy$ 的定义为

$$accuracy=\frac{|\{m|\psi(\Phi(m))=equiv(m)\}|}{|M|},\quad(17)$$

其中,谓词 $equiv(m)$ 表示 m 为等价变异体.给定特征空间 F 和特征提取函数 Φ,定义一个最优等价判定函数 Ψ^*,使得复合函数 $\Psi^*(\Phi(m))$ 在任意变异体构成的集合 M 上实现分类精确率的最大化,有

$$\Psi^*:\max accuracy.\quad(18)$$

对一个大小有限的给定的变异体集 M,可以证明,总存在一个可以实现最优等价判定函数 Ψ^* 的算法.算法 1 给出其基本原理:令 M_x 为集合 M 中符合特征点 x 的变异体子集;当 M_x 中等价变异体个数大于或等于非等价变异体个数时,可以将特征点 x 映射到 1;反之则将特征点 x 映射到 0.算法 1 的基本思想是通过最小化集合 M_x 中的误分类样本数来最大化预测精确率.其中,E_x 和 N_x 分别表示 M_x 中等价和非等价变异体构成的集合.

算法 1. 有限样本集上的最优等价分类算法.

① function $determine\_euivalence(x)$:
② $M_x=\{M$ 中符合特征 x 的变异体$\}$;
③ $E_x=\{M_x$ 中的等价变异体$\}$;
④ $N_x=\{M_x$ 中的非等价变异体$\}$;
⑤ if $|E_x|\geqslant|N_x|$:
⑥ return true;
⑦ else
⑧ return false;
⑨ end if
⑩ end function

需要注意的是,算法 1 不是一个实用化的算法,它的目的是衡量故障特征 F 对等价变异体识别的有效性:在一组测试样本集 M 上,算法 1 的最优分类所取得的精确率是任一分类函数在样本集 M 上该特征所能取得的最大精确率.

一个好的特征空间应该保证对于任意的特征点 x,其对应的变异体中大部分要么都是等价的,要么都是非等价的.这要求从特征点 x 到变异等价性的映射关系的随机性应该尽可能地小.在后面的分析中,还将用特征点的熵(混乱度)来评估 x 的好坏,有

$$H(x)=-p_e\times\mathrm{lb}(p_e)-(1-p_e)\times\mathrm{lb}(p_e),\quad(19)$$

其中 p_e 表示 M_x 中等价变异体所占比例.当 $p_e=1$ 或者 $p_e=0$ 时,有 $H(x)=0$,此时特征点对应的所有变异样本都对应到等价或者非等价,也因此从 x 到等价性的映射关系是完全确定的.当 $p_e=0.5$ 时,$H(x)=1$ 且最大化;此时无法判断具有特征 x 的变异体是属于等价变异体还是非等价变异体.

5 实验评估

为了验证本文提出的基于故障检测上下文的等价变异识别方法的有效性以及可用性,我们进行了一个实验研究.本文提出了 3 个研究问题来估计提出方法的有效性.

Q1. 在等价变异体的识别问题中,故障检测上下文是否是好的故障特征表示.

Q2. 哪一类机器学习模型更适用于基于故障检测上下文的等价变异程序的识别任务.

Q3. 相比于现有技术,基于故障检测上下文的等价变异识别是否具有更好的精确率和泛用度.

5.1 实验程序

本文使用了来自 22 个 C 程序、共计 118 000 个变异体样例作为训练样本,来构造适用于等价变异体识别任务的机器学习分类器.表 3 列出了实验中使用到的程序、代码复杂度以及变异体个数.

实验中我们使用 C 程序变异测试工具 Proteum[41] 实现对程序源代码的自动故障注入功能.为了避免生成过多的样本数、减少手工构造等价变异体的成本,本文去除了常量替换和变量替换类型的变异算子.根据以往的研究,这筛选策略能确保测试的有效性.为识别训练样本中每一个变异体的等价性,本文使用项目组开发的开源变异测试工具 JCMuta① 辅助等价变异体的人工识别工作.步骤如下:

1) 通过自动化工具生成分支覆盖测试用例集 T;

2) 在变异体集 M 上执行测试集 T,并将所有被 T 杀死的变异体标记为“非等价的”;

3) 对未被杀死的 18 526 个变异体,人工分析其是否为等价变异体.

① https://github.com/dzt2/jcsa/JCMuta

Table 3　Subject Programs, Test Cases and Mutants Number
表 3　实验程序、测试用例与变异体数量

程序	代码行	分支数	函数量	变异体
bubble_sort	45	14	5	127
quick_sort	52	16	6	197
prime	96	14	4	211
profit	30	12	4	414
triangle	33	18	4	499
days	38	32	6	587
calendar	261	30	11	595
prime_factor	49	12	6	993
max_tri_path	39	14	10	1 283
md4	791	48	16	3 464
md5	577	78	28	3 722
print_tokens	836	88	18	5 472
print_tokens2	689	102	20	5 370
replace	964	134	16	8 018
schedule	41	50	18	3 847
schedule2	38	52	18	4 035
tcas	14	15	12	2 995
tot_info	33	43	67	7 086
flex	7 395	928	94	11 892
space	6 231	798	136	14 583
sed	8 357	1 460	64	13 964
gzip	6 753	856	80	28 584
合计	33 362	4 814	643	117 938

为了简化人工分析的难度,本文通过 JCMuta 工具收集了每个变异体在测试过程中被测试用例覆盖和检测的程度.具体而言,JCMuta 可以统计某条故障语句是否多少测试用例覆盖;如果没有测试用例覆盖故障语句,则提示用户该变异体等价的原因是因为语句不可达;类似地,通过分析程序中间变量的值是否发生变化,判断一个变异体等价的原因是无法触发状态错误,还是错误无法传播.基于上述分析,通过 2 个月的分析,识别了 11 126 个等价变异样例.

最后,通过使用原型工具 JCMuta,我们为每一个变异体提取相应的 2 阶依赖子图和故障检测上下文,并根据等价变异体识别算法将其转换为故障检测文档和特征向量.在实验中,我们使用 scikit-learn 的编程库搭建机器学习分类器.

5.2　Q1. 故障检测上下文特征评估

为了评估故障检测上下文特征的有效性,实验使用了故障检测上下文特征向量的信息熵,采用式(19)来估计特征的有效性.特征点的信息熵越大,说明基于该特征点对等价性判断的确定性越低.

图 11 展示了故障特征向量空间中每个特征点 $\boldsymbol{X}_m$ 的信息熵的密度分布函数.从图 11 可以发现,多达 98%的特征向量对应的熵 $H(x)=0$,意味着对于实验中使用的大部分故障上下文特征 x 到等价性的映射关系是完全确定的.除此之外,只有约 0.7%特征点的信息熵较高,其 $H(x)>0.50$.

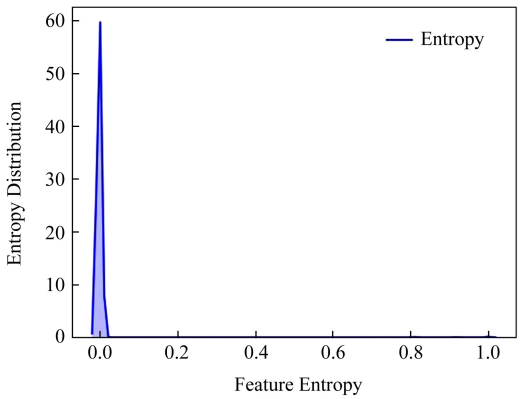

Fig. 11　Distribution of entropy of FCD feature points
图 11　故障检测上下文特征熵的统计分布

表 4 进一步展示了,在故障检测上下文特征空间以及给定训练样本中,最优分类器(算法 1)所能取得的精确率、等价类精确率和召回率等信息.其中,最优分类器能够取得 97.5%的分类精确率,取得多于 90%的精确率以及 91%的召回率.此外,基于故障检测上下文的等价变异识别方法能取得远高于

Table 4　Accuracy, Precision and Recall in Cross-Validation
表 4　等价分类准确率、精确率和召回率(交叉验证)

评估方法	分类器	准确率/%	精确率/%	召回率/%
Train-set=Test-set	最优分类器(算法 1)	97	91	91
交叉验证(2-折)	朴素贝叶斯	69	63	74
	罗吉斯特回归	88	81	57
	决策树	92	83	82
	随机森林	90	87	81
	XGBoost	90	89	64
	Bagging	91	90	70
	多层感知机	93	90	82
交叉验证(5-折)	朴素贝叶斯	70	68	75
	罗吉斯特回归	88	82	60
	决策树	92	85	82
	随机森林	91	81	75
	XGBoost	90	89	65
	Bagging	92	90	76
	多层感知机	95	91	85

传统的等价变异识别算法的精确率和召回率(关于传统方法在等价变异识别问题中的精确率和召回率见表 1).

5.3 Q2.统计学习模型比较

表 4 展示了在训练样本交叉验证中,各种机器学习分类器在验证集(validation set)所取得的准确率(accuracy)、精确率(precision)、召回率(recall)信息.对最优分类器,我们采用 Train-set=Test-set(在训练集上训练和验证)来验证特征空间有效性.而对其他机器学习模型,我们采用了 2-折和 5-折交叉验证技术去评估机器学习分类器的分类准确率、精确率、召回率.

从表 4 可以发现,基于故障检测上下文的最优分类器的精确率是 97%,这是所有其他机器学习分类器在交叉验证中能够获得的最高分数.在实验中使用的学习模型中,决策树、随机森林、梯度上升、Bagging 和多层感知器在交叉验证中取得了最佳分类效果.其中决策树能够获得 92%的精确率以及 83%的等价类精确率;分数最高的多层感知机取得了 90%的识别精确率,并检测了超过 82%的等价变异体.

此外,通过比较 2-折和 5-折交叉验证结果可以发现,虽然在 2-折验证中,各个分类器的精准度和召回率都有所下降,但是这一下降并不明显,意味着本文提出的特征构造和编码方式使得学习模型对训练样本数鲁棒性相对较高,有较好的扩展性.

5.4 Q3. 模型泛用性评估

为了验证机器学习模型在训练集以外程序中进行等价变异识别任务的泛用性,本文采用了留一工程间验证(leave-one-project, LOP).非形式地,LOP 验证每次以训练样本中的 1 个工程的变异体为测试集,而剩余其他程序的样本为训练集,进行学习和预测.

图 12 展示了多层感知机分类器在实验中使用的 22 个被测程序上的 LOP 验证的等价类精准度以及召回率信息.可以发现,对于某些被测的程序,如 md5,prime_factor,print_tokens2,replace,space,flex,sed,分类器能取得超过 90%的预测精确率;然而对另一些程序,如 calendar,triangle,schedule2,gzip,多层感知器取得的精确率甚至低于 50%.类似地,多层感知机在 print_tokens,md4,quick_sort 上的召回率在 90%以上,在 replace,triangle 中召回率低于 45%.

造成学习模型在一部分被测程序的变异体样本作为测试集时性能下降的原因,是因为某些被测程序中存在其他程序中未曾出现过的特征模式,这使得学习模型无法根据其他程序的经验,对具有未曾遇到过的故障特征的变异体进行有效分类.

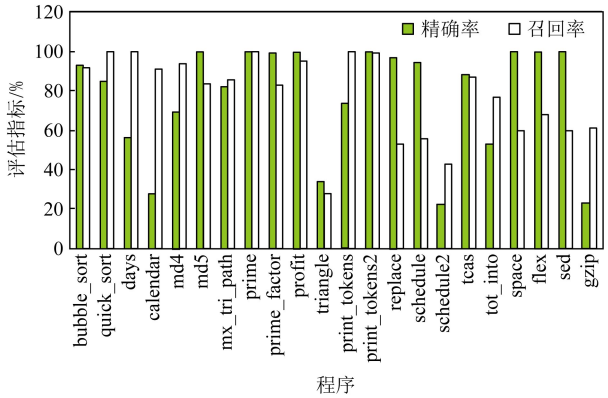

Fig. 12 Leave-one-project validation results

图 12 留一工程间验证统计结果

表 5 给出了实验中使用的 5 个机器学习模型在 LOP 验证中取得的平均精确率(precision)和平均召回率(recall).其中,多层感知机的平均精确率和召回率远高于其他模型.这也意味着通过注意力网络生成的文档表示特征更适合通过多层全连通网络预测等价性.

Table 5 Leave-One-Project Average Precision and Recall

表 5 留一工程间验证的平均精准度和召回率 %

分类器	精确率	召回率	$F1$ 分数
随机森林	49	54	51
决策树	70	46	55
XGBoost	60	38	46
Bagging	73	43	54
多层感知机	77	78	77

5.5 有效性威胁

首先,本文对变异体的故障检测上下文的文档模型并未保证从特征空间到变异等价性的映射是完全确定性的.图 13 以 calendar 程序的被测代码为例,这段代码将日历信息存储到字符串缓存 buffer 中.

```
if (condition_being_mutated)
    sprintf(buffer[line],
        "%s%s%d%s",
        buffer[line],
        "",day,"");
else
    sprintf(buffer[line],
        "%s%d",
        buffer[line],day);
```

Fig. 13 Example of mutant being incorrectly classified

图 13 错分类变异体的样例

在该例子中,条件语句真假分支的语句输出是等价的.而判断该变异体的等价性依赖于对 sprintf 格式化参数的理解,本文提出的故障检测上下文特征并不能有效表示这类与功能相关的代码模式特征.

其次,本文使用机器学习技术来构建从故障上下文特征到变异体等价性之间的映射关系.然而该映射的精确性和泛用性依赖于使用数据的代表性和广泛性.在实验中本文仅收集了 22 组被测程序的变异体作为训练数据集,其泛用性仍有待强化.

最后,通过机器学习技术检测等价变异体要求用户首先提供一组等价变异体的训练样例.这意味着在该方法投入实践之前,手工或者半自动的等价变异体检测仍然是必要的.这一定程度又限制了该方法的迅速普及和使用.

6　总结与未来工作

长期以来,等价变异识别问题一直是阻碍变异测试方法在产业界普及的关键障碍[42].现有的等价变异体识别算法难以同时在精确率和召回率方面取得较好的检测效果.

为了解决这一问题,本文提出了基于故障检测上下文的等价变异识别方法.该方法定义变异体的故障检测上下文作为特征,通过转换为文档模型,使得该问题转变成文档分类问题;最后,通过基于注意力机制的表示学习模型,生成故障检测上下文的文档特征向量,并以该向量为输入,训练机器学习模型,以实现对变异体等价性的自动识别.

在实验分析中,本文提出的方法在 22 个被测程序的 118 000 个变异体上取得了超过 94%的预测精确率.其中在 5-折交叉验证中,多层感知机取得了 91%的精确率(precision)和 85%的召回率(recall),意味着该方法检测出来的变异体超 90%为等价变异体,且样本中约 80%的等价变异体被检测.在工程间验证中,机器学习技术取得 77%的预测精准度和 78%的召回率,验证了本文提出方法的泛用性.本文的其他贡献包括:

1) 提出了用于进行等价变异体检测的故障检测上下文特征模型以及提取该特征的程序分析原型工具 JCMuta.

2) 通过实验分析,验证了基于故障检测上下文的等价变异识别技术的有效性,并指出深度学习框架是相对适合实现本文提出方法的一种学习技术.

在未来的工作中,我们将基于故障检测上下文特征和机器学习模型识别等价变异体可能的故障上下文模式,并通过挖掘这些模式,为测试和开发人员更好地检测等价变异体提供建议.

参　考　文　献

[1] Jia Yue, Harman M. An analysis and survey of the development of mutation testing [J]. IEEE Transactions on Software Engineering, 2010, 37(5): 649-678

[2] Just R, Jalali D, Inozemtseva L, et al. Are mutants a valid substitute for real faults in software testing?[C] //Proc of the 22nd ACM SIGSOFT Int Symp on Foundations of Software Engineering. New York: ACM, 2014: 654-665

[3] Namin S, Andrews J, Murdoch D. Sufficient mutation operators for measuring test effectiveness [C] //Proc of the 30th Int Conf on Software Engineering. Piscataway, NJ: IEEE, 2008: 351-360

[4] Offutt J, Pan J, Tewary K, et al. An experimental evaluation of data flow and mutation testing [J]. Journal of Software: Practice and Experience, 1996, 26(2): 165-176

[5] Wong E, Mathur P. Fault detection effectiveness of mutation and data flow testing [J]. Software Quality Journal, 1995, 4(1): 69-83

[6] Papadakis M, Malevris N. Automatic mutation test case generation via dynamic symbolic execution [C] //Proc of the 21st Int Symp on Software Reliability Engineering. Piscataway, NJ: IEEE, 2010: 121-130

[7] Harman M, Jia Yue, Langdon W B. Strong higher order mutation-based test data generation [C] //Proc of the 19th ACM SIGSOFT Symp and the 13th European Conf on Foundations of Software Engineering. New York: ACM, 2011: 212-222

[8] DeMilli A, Offutt J. Constraint-based automatic test data generation [J]. IEEE Transactions on Software Engineering, 1991, 17(9): 900-910

[9] Wong E, Mathur P, Maldonado C. Mutation versus all-uses: An empirical evaluation of cost, strength and effectiveness [M] //Software Quality and Productivity. Berlin: Springer, 1995: 258-265

[10] Papadakis M, Shin D, Yoo S, et al. Are mutation scores correlated with real fault detection? A large scale empirical study on the relationship between mutants and real faults [C] //Proc of the 40th IEEE Int Conf on Software Engineering. Piscataway, NJ: IEEE, 2018: 537-54

[11] Chilenski J, Miller P. Applicability of modified condition/decision coverage to software testing [J]. Software Engineering Journal, 1994, 9(5): 193-200

[12] Just R, Kapfhammer M, Schweiggert F. Do redundant mutants affect the effectiveness and efficiency of mutation analysis?[C] //Proc of the 5th IEEE Int Conf on Software Testing, Verification and Validation. Piscataway, NJ: IEEE, 2012: 720-725

[13] Mresa S, Bottaci L. Efficiency of mutation operators and selective mutation strategies: An empirical study [J]. Journal of Software Testing, Verification and Reliability, 1999, 9(4): 205-232

[14] Ammann P, Delamaro E, Offutt J. Establishing theoretical minimal sets of mutants [C] //Proc of the 7th IEEE Int Conf on Software Testing, Verification and Validation. Piscataway, NJ: IEEE, 2014: 21-30

[15] Kurtz B, Ammann P, Offutt J, et al. Are we there yet? How redundant and equivalent mutants affect determination of test completeness [C] //Proc of the 9th Int Conf on Software Testing, Verification and Validation. Piscataway, NJ: IEEE, 2016: 142-151

[16] Kurtz B, Ammann P, Delamaro E, et al. Mutant subsumption graphs [C] //Proc of the 7th IEEE Int Conf on Software Testing, Verification and Validation. Piscataway, NJ: IEEE, 2014: 176-185

[17] Kurtz B, Ammann P, Offutt J, et al. Analyzing the validity of selectivemutation with dominator mutants [C] //Proc of the 24th ACM SIGSOFT Int Symp on Foundations of Software Engineering. New York: ACM, 2016: 571-582

[18] Papadakis M, Henard C, Harman M, et al. Threats to the validity of mutation-based test assessment [C] //Proc of the 25th Int Symp on Software Testing and Analysis. New York: ACM, 2016: 354-365

[19] Madeyski L, Orzeszyna W, Torkar R, et al. Overcoming the equivalent mutant problem: A systematic literature review and a comparative experiment of second order mutation [J]. IEEE Transactions on Software Engineering, 2013, 40(1): 23-42

[20] Offutt A, Huffman J. A semantic model of program faults [C] //Proc of the 6th ACM SIGSOFT Int Symp on Software Testing and Analysis. New York: ACM, 1996: 195-200

[21] Schuler D, Zeller A.(Un-)covering equivalent mutants [C] //Proc of the 3rd IEEE Int Conf on Software Testing, Verification and Validation. Piscataway, NJ: IEEE, 2010: 45-54

[22] Grün M, Schuler D, Zeller A. The impact of equivalent mutants [C] //Proc of the 7th IEEE Int Conf on Software Testing, Verification, and Validation Workshops. Piscataway, NJ: IEEE, 2009: 192-199

[23] Schuler D, Zeller A. Covering and uncovering equivalent mutants [J]. Software Testing, Verification and Reliability, 2013, 23(5): 353-374

[24] Offutt J, Pan Jie. Automatically detecting equivalent mutants and infeasible paths [J]. Journal of Software Testing, Verification and Reliability, 1997, 7(3): 165-192

[25] Offutt J, Pan Jie. Detecting equivalent mutants and the feasible path problem [C] //Proc of the 11th Annual Conf on Computer Assurance. Piscataway, NJ: IEEE, 1996: 224-236

[26] Pan Jie. Using constraints to detect equivalent mutants [D]. Fairfax: George Mason University, 1994

[27] Offutt J, Craft M. Using compiler optimization techniques to detect equivalent mutants [J]. Journal of Software Testing, Verification and Reliability, 1994, 4(3): 131-154

[28] Papadakis M, Jia Yue, Harman M, et al. Trivial compiler equivalence: A large scale empirical study of a simple, fast and effective equivalent mutant detection technique [C] //Proc of the 37th IEEE Int Conf on Software Engineering. Piscataway, NJ: IEEE, 2015: 936-946

[29] Zhang Jie, Zhu Muyao, Hao Dan, et al. An empirical study on the scalability of selective mutation testing [C] //Proc of the 25th Int Symp on Software Reliability Engineering. Piscataway, NJ: IEEE, 2014: 277-287

[30] Yao Xiangjuan, Harman M, Jia Yue. A study of equivalent and stubborn mutation operators using human analysis of equivalence [C] //Proc of the 36th IEEE Int Conf on Software Engineering. Piscataway, NJ: IEEE, 2014: 919-930

[31] Hierons R, Harman M, Danicic S. Using program slicing to assist in the detection of equivalent mutants [J]. Journal of Software Testing, Verification and Reliability, 1999, 9(4): 233-262

[32] Harman M, Hierons R, Danicic S. The relationship between program dependence and mutation analysis [M] //Mutation Testing for the New Century. Berlin: Springer, 2001: 5-13

[33] Just R, Kurtz B, Ammann P. Inferring mutant utility from program context [C] //Proc of the 26th ACM SIGSOFT Int Symp on Software Testing and Analysis. New York: ACM, 2017: 284-294

[34] Ferrante J, Ottenstein J, Warren D. The program dependence graph and its use in optimization [J]. ACM Transactions on Programming Languages and Systems, 1987, 9(3): 319-349

[35] Korel B. The program dependence graph in static program testing [J]. Information Processing Letters, 1987, 24(2): 103-108

[36] Weiser M. Program slicing [J]. IEEE Transactions on Software Engineering, 1984(4): 352-357

[37] Mikolov T, Sutskever I, Chen K, et al. Distributed representations of words and phrases and their compositionality [C] //Proc of the 27th Annual Int Conf on Neural Information Processing Systems. New York: ACM, 2013: 3111-3119

[38] Trieu L, Tran H, Tran M. News classification from social media using Twitter-based doc2vec model and automatic query expansion [C] //Proc of the 8th Int Symp on Information and Communication Technology. New York: ACM, 2017: 460-467

[39] Alon U, Zilberstein M, Levy O, et al. Code2vec: Learning the distributed representations of code [J]. ACM Transactions on Program Language, 2019, 3(1): 40-69

[40] Kartchner D, Christensen T, Humpherys J, et al. Code2vec: Embedding and clustering medical diagnosis data [C] //Proc of the 16th IEEE Int Conf on Healthcare Informatics. Piscataway, NJ: IEEE, 2017: 386-390

[41] Delamaro E, Maldonado C, VincenziR. Proteum/IM 2.0: An integrated mutation testing environment [M] //Mutation Testing for the New Century. Berlin: Springer, 2001: 91-101

[42] Sun Changai, Wang Zhen, Pan Lin. Optimized mutation testing techniques for WS-BPEL programs [J]. Journal of Computer Research and Development, 2019, 56(4): 895-905 (in Chinese)
(孙昌爱, 王真, 潘琳. 面向 WS-BPEL 程序的变异测试优化技术[J]. 计算机研究与发展, 2019, 56(4): 895-905)

Yu Chang, born in 1995. Received her BSc and MSc degrees in Beijing University of Posts and Telecommunications. Her main research interests include machine learning and mutation testing.

于　畅,1995 年生.毕业于北京邮电大学,获学士和硕士学位.主要研究方向为机器学习和变异测试.

Wang Yawen, born in 1983. Received her PhD degree in Beijing University of Posts and Telecommunications. Associate professor. Member of CCF. Her main research interests include static program analysis and automatic software testing.

王雅文,1983 年生.毕业于北京邮电大学,获博士学位.现任北京邮电大学副教授,CCF 会员.主要研究方向为静态程序分析和自动化软件测试.

Lin Huan, born in 1989. Received his MSc degree from Beijing University of Aeronautics and Astronautics. PhD candicate in Beijing University of Posts and Telecommunications. His main research interests include program analysis and mutation testing.

林　欢,1989 年生.毕业于北京航空航天大学,获硕士学位.现为北京邮电大学博士研究生.主要研究方向为程序分析和变异测试.

Gong Yunzhan, born in 1962. Received his PhD degree from the Institute of Computing Technology, Chinese Academy of Sciences. Professor of Beijing University of Posts and Telecommunications. His main research interests include software testing and reliability.

宫云战,1962 年生.毕业于中国科学院计算技术研究所,获博士学位.现任北京邮电大学教授.主要研究方向为软件测试与可靠性.