计算机科学
COMPUTER SCIENCE

基于深度学习的回归测试用例优先级排序方法

张李政, 杨秋辉, 李兴佳, 代声馨

引用本文

张李政, 杨秋辉, 李兴佳, 代声馨. 基于深度学习的回归测试用例优先级排序方法[J]. 计算机科学, 2024, 51(12): 46-52.

ZHANG Lizheng, YANG Qiuhui, LI Xingjia, DAI Shengxin. Regression Test Case Prioritization Approach Based on Deep Learning [J]. Computer Science, 2024, 51(12): 46-52.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.231000147

基于深度学习的回归测试用例优先级排序方法

张李政　杨秋辉　李兴佳　代声馨
四川大学计算机学院　成都 610065
(zleo0824@gmail.com)

摘　要　在回归测试中对测试用例排序可以更快地发现代码缺陷,节约测试时间和资源,提高测试效率。现有的测试用例排序方法没有同时考虑代码的变更信息以及测试用例的历史执行信息,也没有考虑不同测试用例执行历史长短的区别,因此排序效果不佳。针对这些问题,提出基于深度学习的回归测试用例优先级排序方法。首先分别构建基于代码变更信息和历史执行信息的分类模型;然后基于类间关系图识别受代码变更影响的类,对这些类的测试用例以及近期执行发现缺陷的测试用例进行分类,使用分类模型和启发式排序方法对测试用例分类进行排序;最后通过交替排序融合排序结果。在 RTPTorrent 数据集上选取 6 个项目进行实验,结果表明:1)在无时间约束时,所提方法在所有项目上都取得了不错的排序效果,在 cloudify 项目上的 APFD 指标达到 0.972;2)在有时间约束时,所提方法的 NAPFD 指标超过了目前主流的排序方案。

关键词:测试用例排序;深度学习;类间关系图;分类模型;分类排序
中图分类号　TP311.5

Regression Test Case Prioritization Approach Based on Deep Learning

ZHANG Lizheng, YANG Qiuhui, LI Xingjia and DAI Shengxin
College of Computer Science, Sichuan University, Chengdu 610065, China

Abstract　Prioritizing test cases in regression testing can expedite the detection of code defects, save testing time and resources, and enhance testing efficiency. However, existing test case prioritization methods often fail to consider both code change information and test case execution history simultaneously, and they do not adequately account for differences in the length of test case execution history, resulting in poor prioritization outcomes. To address these issues, this paper introduces a deep learning-based approach for prioritizing regression test cases. Initially, it constructs classification models based on code change information and historical execution data separately. Subsequently, it identifies classes affected by code changes using inter-class relationship graphs and classifies test cases belonging to these classes, as well as those that have recently exposed defects. Finally, it employs classification models and heuristic sorting method to prioritize the test cases, followed by merging the sorted results through an iterative process. Experimental results on 6 projects selected from the preprocessed RTPTorrent dataset demonstrate that: 1) in scenarios without time constraints, the proposed approach achieves impressive prioritization results across all projects, with an APFD of 0.972 on the cloudify project; 2) under time-constrained conditions, the proposed approach outperforms popular existing prioritization methods in terms of NAPFD metrics.

Keywords　Test case prioritization, Deep learning, Interclass relation graph, Classification model, Categorization sorting

1　引言

目前大多数企业在软件开发过程中都采用持续集成(Continuous Integration,CI)环境,在 CI 环境下,每当开发者提交代码变更后,都需要执行一次代码构建并进行回归测试,以此保证软件代码的更改不会引入新的缺陷。这个过程便是一个集成周期[1]。然而,回归测试非常消耗时间和资源。有文献指出,回归测试需要消耗 80%的测试资源[2],可能花费超过 33%的软件费用[3]。在频繁提交代码变更的持续集成环境中,这个问题更为严峻,测试用例的重新运行需要几个小时甚至数天才能完成[4]。

为了解决这一问题,研究人员提出了一系列回归测试优化方法。其中包括测试用例排序,指按照一定的规则或标准对测试用例进行排列,以便测试团队在有限的资源下更有效地执行测试,减少测试时间和资源的浪费[5]。代码变更会引入缺陷,在排序时需要考虑受变更影响的类的测试用例,

到稿日期:2023-10-20　返修日期:2024-03-16
基金项目:国家自然科学基金(62302323);四川省科技计划资助项目(2023NSFSC1413,2023YFG0117)
This work was supported by the National Natural Science Foundation of China (62302323) and Sichuan Science and Technology Program (2023NSFSC1413,2023YFG0117).
通信作者:杨秋辉(yangqiuhui@scu.edu.cn)

同时，历史执行出错的测试用例在以后的执行中也有较大概率出错[6]，因此同时考虑代码的变更信息和测试用例的历史执行信息可以得到更好的排序效果。然而现有排序方案都是单一排序方法，排序过程中没有将这两类信息结合起来。

针对目前排序方案所存在的问题，本文提出基于深度学习的回归测试用例优先级排序方法。该方法同时考虑了代码的变更信息和历史执行信息，并通过分类排序策略解决了单一排序方法效果不好的问题。实验表明该方法能有效提高排序方案的 APFD 和 NAPFD 指标。

2 相关工作

目前主要的测试用例排序研究有基于贪心算法[7]、基于搜索技术[8]、基于信息检索[9-10]、基于机器学习[11-12]等方法。随着人工智能技术的发展，近年来对测试用例优先级排序的研究中大量应用深度学习、强化学习等方法。

基于深度学习的排序方法使用深度神经网络训练模型预测测试用例的优先级。Sharif 等[13]提出的 DeepOrder 通过测试用例的历史执行信息训练预测模型。Xiao 等[14]基于 LSTM 构建时间序列测试模型，根据前一个 CI 周期的测试历史信息，预测下一个周期中每个测试用例发现故障的概率。但这些方法只使用了测试用例的历史执行信息，没有考虑持续集成环境中的代码变更相关信息。Abdelkarim 等[15]提出的 TCP-Net 是以与测试用例的源代码相关的特征和测试用例覆盖等信息进行训练的深度神经网络，用于测试用例优先级排序。但该方法没有考虑测试用例间的区别。

基于强化学习的排序方法通过设计智能体和奖励函数，使智能体在持续和环境交互后获得预测优先级的能力。Spieker 等[16]提出了一种名为 RETECS 的轻量级回归测试用例优先级排序方法，首次将强化学习技术应用于回归测试优化领域。Wu 等[17]分析了不同长度的历史信息对基于强化学习的测试用例排序技术的影响，并提出了 APHFW 排序方案，使用带时间窗口的历史故障平均百分比（APHFW）的奖励函数，利用测试用例的部分历史执行信息实现更优的排序效果。Yang 等[18]提出的 APHFDSW 方案同样采用强化学习方法，提出了针对单个测试用例的基于动态滑动窗口的奖励函数 APHFDSW。Zhao 等[19]提出了一种基于强化学习的指针排序方法，在每个集成周期中，输入测试用例的历史信息等特征，智能体利用指针注意力机制获取对所有备选测试用例的关注程度，由此得到排序结果，之后根据执行结果的反馈得到策略更新的方向。

3 问题描述与基本思路

测试用例排序问题指对于测试用例集 T 以及其全排列 PT，找出一个 $T'(T'\in PT)$，使得对于任意 $T''(T''\in PT$，且 $T''\neq T')$，都有 $f(T')\geqslant f(T'')$。其中，f 是根据排序目标为排列计算得分的函数[20]。常见的排序目标有更快的错误检测速率、更高的覆盖率和更高的系统无缺陷置信度。

本文方案的目标是提高测试用例排序方法的错误检测速率。为了达到这一目的，在回归测试中，选择受到代码变更影响的类的测试用例，以及不受代码变更影响但近期执行发现缺陷的测试用例，这些测试用例更有可能发现缺陷。然后根据执行次数和缺陷发现情况，将这些测试用例分为 $Type_1$，$Type_2$ 和 $Type_3$ 3 类并分别进行排序。

$Type_1$ 是执行次数较少的测试用例。根据文献研究结果[13]，执行次数少的测试用例使用基于历史信息的优先级排序方法得到的效果相对较差，因此可以考虑根据基于代码变更信息的分类模型的预测结果对测试用例排序。

$Type_2$ 是执行次数较多且近期执行发现缺陷的测试用例。这类测试用例有充足的历史执行信息，由于开发者没有发现或没有完全修复代码缺陷，因此在未来也有较高概率发现缺陷。对于这类测试用例，根据基于历史执行信息的分类模型的预测结果进行排序。

$Type_3$ 是执行次数较多且近期执行未发现缺陷的测试用例。这类测试用例因为无法揭示代码缺陷，或者曾经揭示的缺陷已经被修复，所以在近期通过了较多次的测试，未来揭示缺陷的概率比其他两类都低。研究[21]发现，历史执行次数越多，这类测试用例能揭示缺陷的概率越高。因此对于这类测试用例，基于其在集成周期中的历史失败率排序。

基于测试用例的分类策略，本文提出一种回归测试用例优先级排序方法，主要流程如图 1 所示。首先，分别基于代码变更信息和历史执行信息构建两个分类模型；然后，根据提交的变更信息更新类间关系图，通过分析类间关系图得到受变更影响的类代码，选择这些代码的测试用例，同时通过历史执行信息选择最近发现缺陷的测试用例；接着，将测试用例分为 3 类，分别用不同的方法排序；最后，将分类排序后的结果融合为一个排序序列。

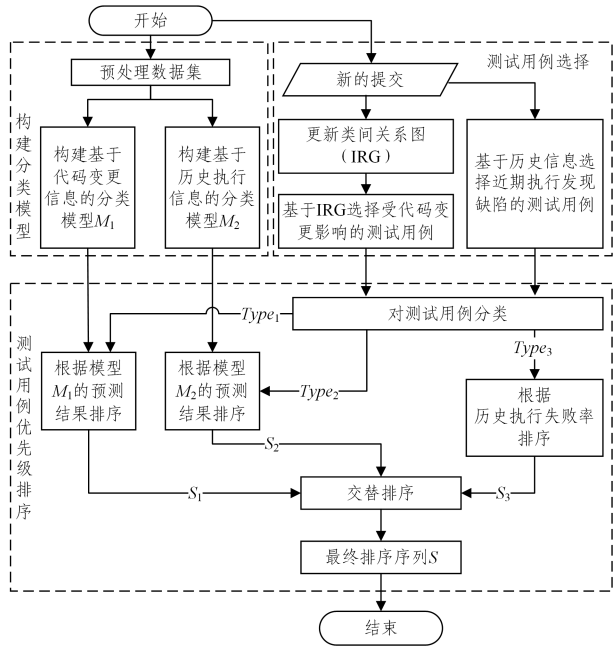


图 1 测试用例优先级排序方法流程图

Fig. 1 Flowchart of test case prioritization approach

4 基于深度学习的分类模型

本章针对 $Type_1$ 测试用例训练基于代码变更信息的分类模型，通过代码变更信息预测测试用例能否发现缺陷；针对 $Type_2$ 测试用例训练基于历史执行信息的分类模型，通过

历史执行信息预测测试用例能否发现缺陷。在后续的分类排序阶段中，两个模型的预测结果会分别作为$Type_1$和$Type_2$测试用例的优先级。

4.1 构建数据集

RTPTotrrent[22]数据集收集了来自 TravisCI 的 10 万多个真实构建历史，其中包含了项目的版本控制元数据(Metadata)、源代码、测试结果等信息。本文在 RTPTorrent 数据集基础上进行扩展，为每个项目生成以代码构建为单位的数据项，每个数据项包括源代码、源代码类间关系图、测试用例套件以及对应的测试结果等信息。对于每个代码构建，将其所有测试用例根据自身的执行次数和缺陷发现情况划分到相应的数据集中。

针对基于历史执行信息的分类模型，本文借鉴文献[23]选取代码构建中执行次数不少于 16 次，且最近 16 次执行中发现缺陷的测试用例，并选取测试用例最近 16 的次执行结果、方法数量、执行时长作为其特征。方法数量和执行时长体现了测试的复杂程度，包含更多的方法数量或者执行时间更长的测试用例揭示缺陷的概率更大。

针对基于代码变更信息的分类模型，选取代码构建中受到代码变更影响且执行次数少于 16 次的测试用例。选取的特征包括测试用例名字、最近变更类名字、全部变更文件名、最近 3 次执行结果和测试用例年龄。测试用例名字、最近变更类名字和全部变更文件名主要被用于提取代码变更和测试用例的语义信息，使模型学习到代码变更信息与测试用例执行结果之间的关联；测试用例的历史执行结果是测试用例排序任务中常用的信息，由于测试用例执行次数较少，因此仅使用最近 3 次的执行结果；测试用例的年龄表示测试用例初次加入到现在经过的构建次数，较老的测试用例发现缺陷的概率小于较新的测试用例。同时，由于一次构建中修改的文件数量越多，越可能引入缺陷[24]，因此需要提取出修改的文件数量特征。类间关系图中，一个类结点到最近的变更类结点需要经过的边数称为距离，这个类的测试用例是否能揭示缺陷与距离也有相关性[25]，因此还要选取距离特征。

两个模型的数据集标签设置方法相同，如果测试用例能够揭示缺陷或触发错误，则标签为 1，否则为 0。

数据集构建完成后，基于历史执行信息的分类模型的数据集中共有 75509 条数据；基于代码变更信息的分类模型的数据集中共有 56646 条数据。

4.2 网络结构

基于代码变更信息的分类模型结构如图 2 所示。使用 CodeBERT[26]将测试用例名字、最近变更类名字和全部变更文件名字作为模型的嵌入层，将修改的文件数量、距离、最近 3 次执行结果和测试用例年龄特征输入全连接网络 Dense4 中，做初步转换。最后，将以上两部分输出拼接后输入全连接网络 Dense5 中，Dense5 后接一个线性层和 softmax[27]层，由 softmax 层输出预测概率分布向量 $\boldsymbol{p}$。

基于历史执行信息的分类模型根据历史执行结果、方法数和平均执行时长来预测下一次执行能否发现缺陷。模型结构如图 3 所示，主要由若干个全连接层和一个 softmax 层组成，最终由 softmax 层输出预测概率分布向量 $\boldsymbol{p}$。

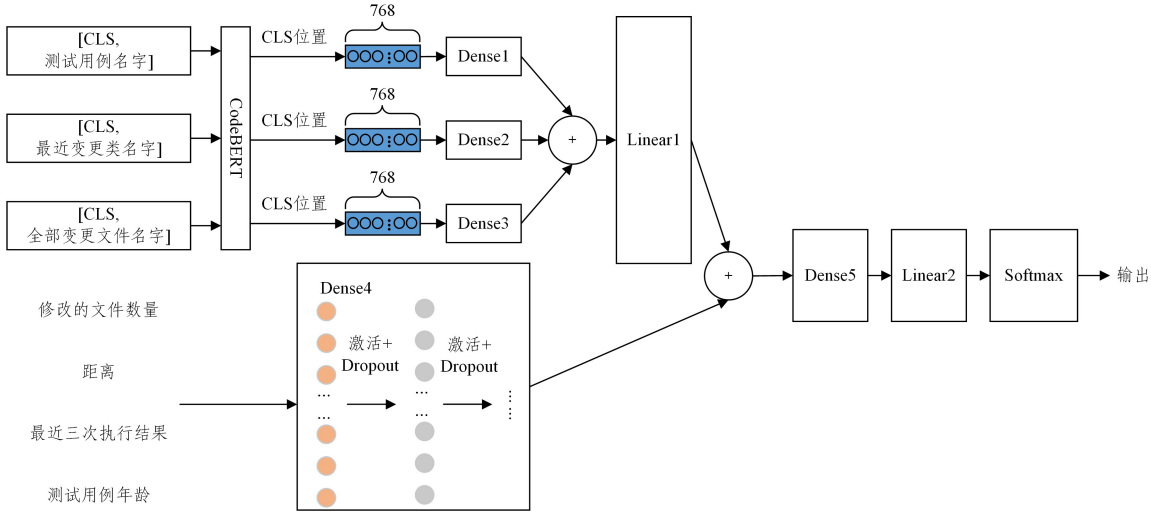


图 2 基于代码变更信息的分类模型结构

Fig. 2 Structure of classification model based on code change information

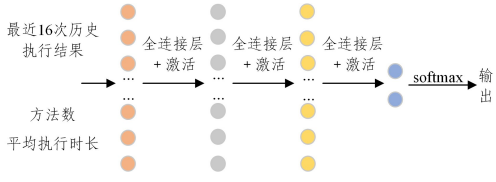


图 3 基于历史执行信息的分类模型结构

Fig. 3 Structure of classification model based on historical execution information

4.3 模型训练和评估

将数据集中的测试用例按照时间顺序，以 6∶2∶2 的比例分为训练集、验证集和测试集。两个模型的训练都使用 Focal Loss 损失函数，该损失函数可以有效解决分类问题中数据不平衡的问题[28]。同时，使用 Adam 优化器和 ReLU 激活函数。在验证集上通过网格搜索对超参数进行调优后，两个模型的超参数设置如表 1 和表 2 所列，评估结果如表 3 所列。

表 1 基于代码变更信息的分类模型的超参数

Table 1 Hyperparameters of classification model based on code change information

超参数	参数值
学习率	0.001
训练轮次	250
Dense1，Dense2，Dense3 Dropout 概率	0.2
Dense4，Dense5 Dropout 概率	0.1

表 2 基于历史执行信息的分类模型的超参数设置

Table 2 Hyperparameters of classification model based on historical execution information

超参数	参数值
学习率	0.001
训练轮次	140
Dropout 概率	0.2

表 3 模型评估结果

Table 3 Results of model evaluation

项目	基于代码变更信息的分类模型			基于历史执行信息的分类模型		
	Precision	*Recall*	*F*1	*Precision*	*Recall*	*F*1
cloudify	0.72	0.73	0.72	0.84	0.95	0.89
deeplearning4j	0.76	0.97	0.85	0.98	0.99	0.98
graylog2-server	0.74	0.76	0.75	0.41	1.00	0.58
jetty. project	0.70	0.83	0.76	0.95	0.98	0.96
okhttp	0.73	0.77	0.75	0.80	0.83	0.81
wicket-bootstrap	0.81	0.96	0.88	0.97	1.00	0.98

5 基于分类策略的优先级排序方案

5.1 测试用例选择

开发者提交变更后，需要选择出重新运行的测试用例，本文通过类间关系图[29]选择测试用例。针对 Java 语言，使用 Python 的 javalang 模块将 Java 源文件解析成 AST，然后从中提取出类和接口、继承或实现关系、调用关系，生成类间关系图。当图中某一结点发生变化时，能通过图中的类间关系分析出所有受影响的结点。

当提交变更后，基于类间关系图提取出所有受影响的代码并更新类间关系图，选择出所有受影响代码的测试用例。为了保证测试用例选择的安全性，避免遗漏可能发现缺陷的测试用例，还需要从历史执行信息中选择近期揭示过缺陷的测试用例，这些测试用例在未来仍然有较大概率揭示缺陷。本文参考文献[23]选择历史 16 次运行结果中揭示过缺陷的测试用例。

5.2 测试用例排序

根据执行次数和缺陷发现情况，将选择的测试用例分为 3 类，分别使用基于代码变更信息的分类模型、失败率和基于历史执行信息的分类模型来排序。

针对 $Type_1$ 和 $Type_2$ 测试用例，分别使用基于代码变更信息的分类模型和基于历史执行信息的分类模型来预测测试用例揭示缺陷的概率。将概率值作为优先级，按优先级降序排列。优先级相同时，按照历史平均执行时间升序排列，若执行时间也相同，则随机排列。最终得到序列 S_1 和 S_2。

$Type_3$ 测试用例的历史执行结果特征不明显，使用模型预测的效果较差。已有研究发现历史失败率高的测试用例有更大的可能性揭示缺陷，因此本文基于历史失败率对 $Type_3$ 测试用例排序。首先计算测试用例的历史失败率，然后按历史失败率降序排序，得到序列 S_3。

测试用例 t 的历史失败率 $HFR_i(t)$ 计算式如式(1)所示，其中 $|TH_i^{FR}(t)|$ 和 $|TH_i(t)|$ 分别表示在第 i 次集成周期时测试用例 t 历史执行中检测到缺陷的次数和历史执行总次数。

$$HFR_i(t)=\frac{|TH_i^{FR}(t)|}{|TH_i(t)|}\times 100\% \tag{1}$$

5.3 合并排序结果

在得到分类排序的结果后，需要将 S_1，S_2 和 S_3 融合为一个序列 S。为保证 S 有较高的揭错效率，要将有较大概率揭示缺陷的测试用例放在靠前位置，本文使用交替排序方法合并排序结果。

针对序列 S_1 和 S_2，首先将序列中模型预测概率值大于 0.5 的测试用例标注为正例，代表可能揭示代码缺陷；将其余测试用例标注为负例，这类测试用例揭示代码缺陷的概率很小。大部分情况下，基于历史执行信息的分类模型准确率更高，因此认为 S_2 中的结果更可信，将 S_2 中的正例放在序列 S 的首部位置，接着将序列 S_1 中的正例添加到序列 S 中，这些测试用例相比 S_1 和 S_2 序列中的负例更有可能发现缺陷；然后依次添加 S_1 和 S_2 中的负例。

$Type_3$ 类型的测试用例由于近期多次通过测试，下一次执行测试时也有很大的概率通过测试，其揭示缺陷的概率最小，因此最后将序列 S_3 添加到序列 S 的末尾。

6 实验验证

6.1 实验对象

为了保证实验结果的有效性，选择 RTPTorrent 数据集中代码构建次数和测试用例执行次数都较多的部分项目作为实验对象。实验对象信息如表 4 所列。针对每个项目，选择预处理过程中被划分为测试集的测试用例来对排序方法进行评估。

表 4 实验对象信息

Table 4 Information of experiment subjects

项目名称	代码构建次数	测试用例执行次数	失败率/%
cloudify	496	18797	3.20
deeplearning4j	566	8372	10.85
graylog2-server	247	7656	5.26
jetty. project	327	45303	0.92
okhttp	772	22288	4.21
wicket-bootstrap	342	13734	65.58

6.2 评估指标

平均故障检测率 APFD[30]是 Rothermel 等最早提出的

用于评估测试用例优先级排序方法有效性的指标。APFD 衡量了优先级排序后的测试套件检测缺陷的速度，其值越大，缺陷检测速度越快。其计算公式为：

$$APFD=1-\frac{\sum_{j=1}^{m}TF_j}{nm}+\frac{1}{2n} \tag{2}$$

其中，TF_j 表示在优先级排序后的测试套件中首次检测到第 j 个缺陷的测试用例，n 表示测试用例数量，m 表示测试套件检测到的缺陷数量。

在测试资源有限的情况下，Qu 等提出的归一化平均故障检测率[31](NAPFD)可以评估只能执行部分测试用例时排序方法的效果。其计算式为：

$$NAPFD=p-\frac{\sum_{j=1}^{m}TF_j}{nm}+\frac{p}{2n} \tag{3}$$

其中，n，m，TF_j 含义与式(2)中一致，p 表示优先级排序的测试套件检测出的故障数量和完整的测试套件检测出的故障数量的比值。

6.3 实验设计

为了验证方案的有效性，在所选的实验对象上，将目前流行的 RETECS[8]，APHFW[9]，APHFDSW[10]，DeepOrder[6]，TCP-Net[15] 排序方案和本文方案的排序结果进行对比，并回答如下两个研究问题。

问题 1：在没有时间约束的条件下，本方案与流行的排序方案相比，是否有更优的排序结果？

问题 2：在有时间约束的条件下，无法运行所有测试用例时，本方案是否有更优的排序结果？

6.4 实验结果与分析

6.4.1 问题 1

通过本文方案对测试套件中的 3 类测试用例进行排序后得到序列 S，将剩余的测试用例按照历史失败率降序排列得到序列 $Rest$，将序列 $Rest$ 添加到序列 S 后面得到执行序列 $Final$，最后通过 $Final$ 序列的执行结果计算 APFD 指标。因为 RETECS，APHFW，APHFDSW 方案会受到随机性影响[32]，因此对这些方案运行 30 次并取 APFD 结果的平均值。

图 4 所示的实验结果表明，本文方案在 6 个项目上都取得了不错的结果。在 graylog2-server 项目上，本方案效果(0.546)略差于 APHFW 方案(0.550)和 APHFDSW 方案(0.568)；在 wicket-bootstrap 项目上，本方案效果(0.641)略差于 DeepOrder 方案(0.644)和 TCP-Net 方案(0.652)；在另外4 个项目上，本方案都优于其他方案，在 cloudify 项目上本文方案表现最好，APFD 指标达到了 0.972。

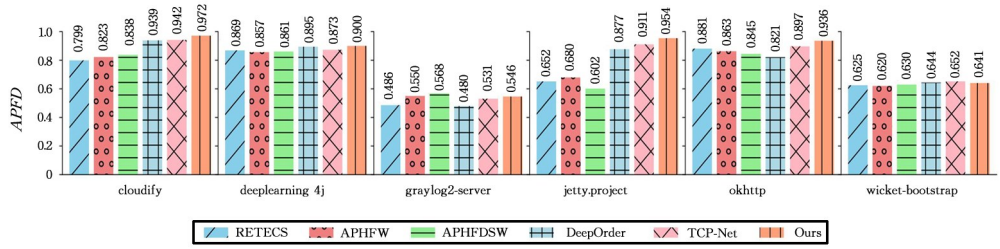


图 4 不同方案在各项目上的 APFD 结果

Fig. 4 APFD results of different approaches on various projects

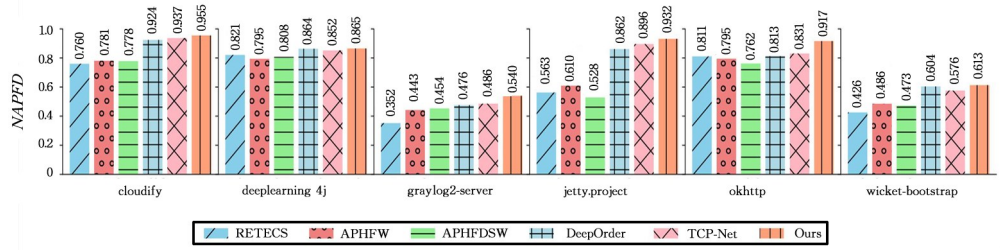


图 5 不同方案在各项目上的 NAPFD 结果

Fig. 5 NAPFD results of different approaches on various projects

通过分析发现，在 graylog2-server 项目中，大量测试用例使用相同的名字“TestSuite”，导致无法区分这些测试用例的运行结果，使得模型做出了错误的预测。在 wicket-bootstrap 项目中，其最后两个集成周期仅有少量测试用例执行失败，不符合之前的集成周期的执行结果所表现出的数据模式，导致模型无法学习到这些异常的数据。

总的来说，实验结果证明了方案在无时间约束条件下的有效性。

6.4.2 问题 2

在问题 1 的研究基础上，参考文献[31]，选择序列 $Final$ 中占总执行时长前 50% 的测试用例作为执行序列，模拟在受时间约束条件下只能执行部分测试用例的情景。

图 5 所示的实验结果表明，在受时间约束的条件下，本文方案仍然达到了更优的排序效果，在 6 个实验项目上的 NAPFD 指标都超过了其他方案。

结束语 本文针对目前测试用例排序方法存在的不足，提出了基于深度学习的测试用例排序方法，解决了测试用例排序方案没有充分利用代码和测试用例的信息，以及使用单一排序方法效果不好的问题。首先通过静态分析代码变更后的类间关系图，选择出受变更影响的代码；然后选择这些代码的测试用例和近期发现缺陷的测试用例，对这些测试用例按照执行次数和缺陷发现情况分类后，分别使用基于深度学习

的分类模型和启发式方法进行排序；最后使用交替排序方法将分类排序结果融合成一个排序序列。实验表明，本文方案能取得不错的排序效果，具有有效性。

然而，所提方法仍然存在一些不足之处。1）基于类粒度的测试用例选择方法并没有充分考虑方法的准确性。具体来说，首先通过类间关系图选择受代码变更影响的类，然后再选择类代码的测试用例，当代码变更的粒度是方法级别时，会导致受影响的类的所有测试用例被选择出来，然而可能存在部分测试用例并没有对变更的方法进行测试，因此执行结果可能并不会受影响。2）在对排序方案进行评估时，只选取了RTPTorrent数据集上的部分项目进行了实验，而方案在其他项目上的泛化能力尚未得到验证，因此未来可以在更多的数据集上进行实验以进行更全面的评估。

参 考 文 献

[1] MARIJAN D. Comparative study of machine learning test case prioritization for continuous integrationtesting [J]. Software Quality Journal, 2023, 28(4): 1-24.

[2] MARCHETTO A, SCANNIELLO G, SUSI A. Combining code and requirements coverage with execution cost for test suite reduction[J]. IEEE Transactions on Software Engineering, 2017, 45(4): 363-390.

[3] KHATIBSYARBINI M, ISA M A, JAWAWI D N A, et al. Test case prioritization approaches in regression testing: A systematic literature review [J]. Information and Software Technology, 2018, 93(C): 74-93.

[4] LIMA J A, VERGILIO S R. A multi-armed bandit approach for test case prioritization in continuous integration environments [J]. IEEE Transactions on Software Engineering, 2020, 48(2): 453-65.

[5] LOU Y, CHEN J, ZHANG L, et al. A survey on regression test-case prioritization[M]//Advances in Computers. Elsevier, 2019: 1-46.

[6] SIQUEIRA V, MIRANDA B. Investigating the Adoption of History-based Prioritization in the Context of Manual Testing in a Real Industrial Setting[C]// 48th Euromicro Conference on Software Engineering and Advanced Applications (SEAA). IEEE, 2022: 141-148.

[7] LI F, ZHOU J, LI Y, et al. AGA: An Accelerated Greedy Additional Algorithm for Test Case Prioritization[J]. IEEE Transactions on Software Engineering, 2022, 48(12): 5102-5119.

[8] DI N D, PANICHELLA A, ZAIDMAN A, et al. A Test Case Prioritization Genetic Algorithm Guided by the Hypervolume Indicator [J]. IEEE Transactions on Software Engineering, 2020, 46(6): 674-696.

[9] YANG L, CHEN J, YOU H, et al. Can Code Representation Boost IR-Based Test Case Prioritization? [C]// IEEE 34th International Symposium on Software Reliability Engineering (ISSRE). IEEE, 2023: 240-251.

[10] AZIZI M. QRTest: Automatic Query Reformulation for InformationRetrieval Based Regression Test Case Prioritization[C]// IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW). IEEE, 2021: 254-262.

[11] LIN C T, YUAN S H, INTASARA J. A Learning-to-Rank Based Approach for Improving Regression Test Case Prioritization[C]// 28th Asia-Pacific Software Engineering Conference (APSEC). IEEE, 2021: 576-577.

[12] SHARMA S, CHANDE S V. Optimizing test case prioritization using machine learning algorithms[J]. Journal of Autonomous Intelligence, 2023, 6(2): 661-676.

[13] SHARIF A, MARIJAN D, Liaaen M. DeepOrder: Deep learning for test case prioritization in continuous integration testing [C]// IEEE International Conference on Software Maintenance and Evolution (ICSME). IEEE, 2021: 525-534.

[14] XIAO L, MIAO H, SHI T, et al. LSTM-based deep learning for spatial-temporal software testing [J]. Distributed and Parallel Databases, 2020, 38(3): 687-712.

[15] ABDELKARIM M, ELADAWI R. TCP-Net: Test Case Prioritization using End-to-End Deep Neural Networks[C]// IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW). IEEE, 2022: 122-129.

[16] SPIEKER H, GOTLIEB A, MARIJAN D, et al. Reinforcement learning for automatic test case prioritization and selection in continuous integration[C]// Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis. 2017: 12-22.

[17] WU Z, YANG Y, LI Z, et al. A time window based reinforcement learning reward for test case prioritization in continuous integration[C]// Proceedings of the 11th Asia-Pacific Symposium on Internetware. 2019: 1-6.

[18] YANG Y, PAN C, LI Z, et al. Adaptive Reward Computation in Reinforcement Learning-Based Continuous Integration Testing [J]. IEEE Access, 2021, 9: 36674-36688.

[19] ZHAO Y F, HAO D. Test Case Prioritization Technique in Continuous Integration Based on Reinforcement Learning[J]. Journal of Software, 2023, 34(6): 2708-2726.

[20] ROTHERMEL G, UNTCH R H, CHU C, et al. Prioritizing test cases for regression testing[J]. IEEE Transactions on software engineering, 2001, 27(10): 929-948.

[21] NOOR T B, HEMMATI H. A similarity-based approach for test case prioritization using historical failure data[C]// IEEE 26th International Symposium on Software Reliability Engineering (ISSRE). IEEE, 2015: 58-68.

[22] MATTIS T, REIN P, DÜRSCH F, et al. RTPTorrent: An open-source dataset for evaluating regression test pri-oritization [C]// Proceedings of the 17th International Conference on Mining Software Repositories. 2020: 385-396.

[23] XIAO L. Research on Test Case Prioritization Technique Based on History Execution Information in Continuous Integration [D]. Shanghai: Shanghai University, 2020.

[24] MACHALICA M, SAMYLKIN A, PORTH M, et al. Predictive test selection[C]// IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP). IEEE, 2019: 91-100.

[25] MEMON A, GAO Z, NGUYEN B, et al. Taming Google-scale continuous testing[C]// IEEE/ACM 39th International Confe-

rence on Software Engineering: Software Engineering in Practice Track(ICSE-SEIP). IEEE, 2017: 233-242.

[26] FENG Z, GUO D, TANG D, et al. CodeBERT: A Pre-Trained Model for Programming and Natural Languages[C]// Findings of the Association for Computational Linguistics. EMNLP, 2020: 1536-1547.

[27] ADEM K, KILICARSLAN S, COMERT O. Classification and diagnosis of cervical cancer with softmax classification with stacked autoencoder[J]. Expert Systems with Applications, 2019, 115: 557-564.

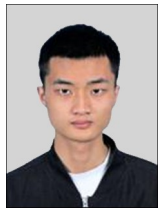
[28] LIN T Y, GOYAL P, GIRSHICK R, et al. Focal loss for dense object detection[C]// Proceedings of the IEEEInternational Conference on Computer Vision. IEEE, 2017: 2980-2988.

[29] ORSO A, SHI N, HARROLD M J. Scaling regression testing to large software systems[J]. ACM SIGSOFT Software Engineering Notes, 2004, 29(6): 241-251.

[30] ROTHERMEL G, UNTCH R H, CHU C, et al. Test case prioritization: An empirical study[C]// Proceedings IEEE International Conference on Software Maintenance. IEEE, 1999: 179-188.

[31] QU X, COHEN M B, WOOLF K M. Combinatorial interaction regression testing: A study of test case generation and prioritization[C]// IEEE International Conference on Software Maintenance. IEEE, 2007: 255-264.

[32] BAGHERZADEH M, KAHANI N, BRIAND L. Reinforcement learning for test case prioritization[J]. IEEE Transactions on Software Engineering, 2021, 48(8): 2836-2856.



ZHANG Lizheng, born in 2000, postgraduate, is a member of CCF (No. Q6918G). His main research interests include software quality assurance and testing and automated program repair.

YANG Qiuhui, born in 1970, Ph.D, associate professor. Her main research interests include software automation testing and software project management.

(责任编辑:柯颖)