软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2019,30(5):1438−1449 [doi: 10.13328/j.cnki.jos.005714]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

面向持续集成测试优化的强化学习奖励机制*

何柳柳, 杨 羊, 李 征, 赵瑞莲

(北京化工大学 信息科学与技术学院,北京 100029)

通讯作者: 李征, E-mail: lizheng@mail.buct.edu.cn

摘 要: 持续集成环境下的测试存在测试用例集变化大、测试时间有限和快速反馈等需求,传统的测试优化方法难以适用.强化学习是机器学习的一个重要分支,其本质是解决序贯决策问题,可以用于持续集成测试优化.但现有的基于强化学习的方法中,奖励函数计算只包括测试用例在当前集成周期的执行信息.从奖励函数设计和奖励策略两个方面开展研究.在奖励函数设计方面,采用测试用例的完整历史执行信息替代当前执行信息,综合考虑测试用例历史失效总次数和历史失效分布信息,提出了两种奖励函数.在奖励策略方面,提出对当前执行序列的测试用例整体奖励和仅对失效测试用例的部分奖励两种策略.在 3 个工业级被测程序进行实验研究,结果表明:(1) 与现有方法相比,所提出的基于完整历史执行信息奖励函数的强化学习方法可以大幅度提高持续集成测试序列的检错能力;(2) 测试用例历史失效分布有助于发现潜在失效的测试用例,对强化学习奖励函数的设计更加重要;(3) 整体奖励与部分奖励两种奖励策略受到被测程序的多种因素影响,需要根据实际情况具体选择;(4) 包含历史信息的奖励函数会增加时间消耗,但并不影响测试效率.

关键词: 持续集成测试;测试用例优先排序;测试用例历史执行信息;强化学习;奖励函数

中图法分类号: TP311

中文引用格式: 何柳柳,杨羊,李征,赵瑞莲.面向持续集成测试优化的强化学习奖励机制.软件学报,2019,30(5):1438−1449. http://www.jos.org.cn/1000-9825/5714.htm

英文引用格式: He LL, Yang Y, Li Z, Zhao RL. Reward of reinforcement learning of test optimization for continuous integration. Ruan Jian Xue Bao/Journal of Software, 2019,30(5):1438−1449 (in Chinese). http://www.jos.org.cn/1000-9825/5714.htm

Reward of Reinforcement Learning of Test Optimization for Continuous Integration

HE Liu-Liu, YANG Yang, LI Zheng, ZHAO Rui-Lian

(College of Information Science and Technology, Beijing University of Chemical Technology, Beijing 100029, China)

Abstract: Testing in continuous integration environment is characterized by constantly changing test sets, limited test time, fast feedback, and so on. Traditional test optimization methods are not suitable for this. Reinforcement learning is an important branch of machine learning, and its essence is to solve sequential decision problems, thus it can be used in test optimization in continuous integration. However, in the existing reinforcement learning based methods, the reward function calculation only includes the execution information of the test case in the current integration cycle. The research is carried out from two aspects: reward function design and reward strategy. In the design of reward function, complete historical execution information of the test case is used to replace the current execution information and the total number of historical failures and historical failure distribution information of the test case is also considered. In terms of the reward strategy, two reward strategies are proposed, which are overall reward for test cases in current execution sequence and partial reward only for failed test cases. In this study, experimental research is conducted on three industrial-level programs. The results show that: (1) Compared with the existing methods, reinforcement learning methods based on reward function with

* 基金项目: 国家自然科学基金(61872026, 61472025, 61672085)
Foundation item: National Natural Science Foundation of China (61872026, 61472025, 61672085)
本文由智能化软件新技术专刊特约编辑申富饶教授和李戈副教授推荐.
收稿时间: 2018-08-29; 修改时间: 2018-10-31; 采用时间: 2018-12-13

complete historical information proposed in this study can greatly improve the error detection ability of test sequences in continuous integration; (2) Test case historical failure distribution can help to identify potential failure test cases, which is more important for the design of the reward function in reinforcement learning; (3) The two reward strategies, i.e. overall reward and partial reward, are influenced by various factors of the system under test, therefore the reward strategy need to be selected according to the actual situation; and (4) History-based reward functions have longer time consumption, though the test efficiency is not affected.

Key words: continuous integration testing; test case prioriitization; historical execution information of test case; reinforcement learning; reward function

持续集成(continuous integration,简称 CI)是一项旨在实时部署的软件开发实践.持续集成包含版本控制、软件配置管理、自动构建以及回归测试等步骤.持续集成要求开发人员必须频繁地整合他们的工作,通常是至少每天一次有时甚至每天多次,同时要求将发布软件保持在潜在可发布状态.每次集成会通过自动构建测试过程进行验证,以尽快检测集成错误.与早期相隔较长时间再集成的开发模式相比,持续集成的集成频率高,因此需要快速发现错误并快速反馈,使开发人员了解软件集成的情况,并对不成功的集成进行快速的修改,进而提高软件开发的效率和质量.但随着软件规模的增大,测试用例集包含的信息越来越多,现有的回归测试优化技术难以满足持续集成对测试结果快速反馈的需求.

为提高持续集成回归测试的效率,需要将潜在发现错误的测试用例优先运行.主要方法延续了传统的测试优化,包括测试用例约减、测试用例选择和测试用例优先排序.测试用例约减识别并消除测试用例集中过时或冗余的测试用例.测试用例选择基于特定准则选取测试用例集的子集,通常选取用于测试软件更改部分的测试用例.而测试用例优先排序旨在找出一个好的测试用例执行序列,使潜在发现错误的测试用例能尽早地被执行.在持续集成测试优化中,测试用例优先排序技术和测试用例选择是主要研究并应用的两种技术.

近年来,随着机器学习方法的兴起,测试用例优化技术逐步和基于学习的方法相结合.Busjaeger 等人[1]提出使用机器学习方法来集成多个现有测试用例优化技术,进行测试用例集的优先排序.强化学习是机器学习的一个重要分支,其重点在于学习者不依赖监督机制或完整的环境模型,直接与其环境相互作用而学习.强化学习强调基于环境反馈进行决策以获取最大利益.在软件持续集成测试中,由于每次集成都需要针对集成的代码对测试用例集合优化,以生成适用于本次集成的测试用例子集,所以持续集成测试优化可以定义成一个基于软件代码集成的测试用例执行序列决策(序贯决策,sequential decision)问题,即根据集成代码的特点,选择要执行的测试用例子集并确定其执行次序.持续集成测试优化是一个测试用例选择和优先排序相混合的优化问题,可以采用强化学习解决这个问题.Spieker 等人[2]在 2017 年提出一种基于强化学习的方法 RETECS(reinforced test case selection),首次将强化学习应用于软件持续集成测试优化中.

强化学习通过奖励函数(reward function)来计算智能体选取某个动作后的即时奖励,是智能体与环境交互的直接反馈.好的奖励函数能更加准确地反映两者的交互情况,使智能体更加准确地感知其环境状态,从而加快强化学习的收敛过程.Spieker 等人提出的 RETECS 方法中,根据测试用例集的持续时间、邻近一次执行和失效信息来自动学习测试用例的检错能力,提高易于发现错误的测试用例的执行优先级.

强化学习的核心是在与环境的交互过程中进行学习,即基于历史交互信息来决定当前的执行.RETECS 方法的奖励函数考虑了测试用例当前一次集成中的执行信息,包括失效总数、执行结果与失效的测试用例在序列中的位置对测试序列的影响,但缺少测试用例在集成过程中的整体历史执行信息.本文从奖励函数设计和奖励策略两个方面开展研究.在奖励函数设计方面,综合考虑测试用例历史执行失效总次数和失效在全部集成过程中的分布,提出两种基于测试用例历史执行信息的强化学习奖励函数.在奖励策略方面,提出整体奖励和部分奖励两种策略,形成面向持续集成测试优化强化学习方法的奖励机制.本文的主要贡献包括:

(1) 针对测试用例在全部持续集成过程中的历史执行信息,定义了测试用例历史平均失效率,在表征了测试用例整体历史失效次数的同时,度量了失效在集成过程中的分布.

(2) 在持续集成测试优化的强化学习奖励函数设计上,综合考虑测试用例全部执行历史信息,提出了两种基于历史信息的强化学习奖励函数;在奖励策略方面,提出了整体奖励和部分奖励两种策略.

(3) 在实际工业程序上进行了实证研究,结果显示:在软件持续集成测试优化中,融合历史信息的强化学习奖励函数优于现有方法.

本文第 1 节总结相关研究,引出本文研究动机.第 2 节提出本文基于测试用例历史执行信息的奖励函数.第 3 节通过实验对比分析验证本文所提出奖励函数的有效性.第 4 节介绍相关工作.最后,第 5 节进行总结与展望.

1 相关工作

测试用例优先排序(test case prioritization,简称 TCP)问题是一个时序问题.在测试过程中,测试用例先后逐个执行,测试人员希望检测到错误可能性大的测试用例优先执行,以更早地发现错误和修复错误,减少损失[3].测试用例优先排序是对测试用例个体进行评价并依据其重要性进行排序,最早由 Wong 等人在 1997 年提出[4],即在软件回归测试过程中,对原有的测试用例集,寻找满足测试准则的最优测试用例执行序列.Elbaum 和 Rothermel 等人[5]在 2000 年给出形式化描述,并开展了一系列的实证研究.随后提出了多种测试用例优先排序技术,包括基于测试用例覆盖信息、基于程序修改信息、基于故障、基于需求和基于历史等的 TCP 技术[6].

基于搜索的思想可以有效解决解决复杂优化问题.Li 等人率先提出基于搜索的软件回归测试优先排序[7],进一步研究多目标 TCP 问题的搜索方法,并使用 GPU 并行技术提高算法效率[8,9].Souza 等人[10]和 Yu[11]等人采用基于搜索的方法,根据测试用例相似度解决 TCP 问题.

基于学习的方法是一种数据驱动的方法.Busjaeger 等人[1]将机器学习用于测试用例优先排序,结合启发式思想,定义了可以面向不同维度的测试用例信息表示方法,提出一种基于学习的测试优先排序通用化框架.Chen 等人[12]将半监督聚类算法应用于测试用例选择,但存在计算复杂度高等问题.强化学习是机器学习的一个重要分支,Groce 等人将强化学习与 ABP 结合,用来对软件 API 进行测试[13];Reichstaller 等人将强化学习用于基于风险的互操作性测试,通过强化学习进行风险评估,进而实现测试用例生成[14].

软件持续集成的关键是代码集成到主干之前,必须通过自动化回归测试,即每次代码集成都需要检测其是否引入新的缺陷,一旦有测试用例失败,就不能集成.所以持续集成测试要求在有限的时间内完成回归测试,并尽早发现错误.在面向软件持续集成测试的研究中,Marijan 等人[15]提出了面向持续集成的测试用例优先排序技术,随后,多种优化目标和方法被提出[16,17].面对多个优化目标时,Ammar 等人[18]采用改变不同优化目标的权重来进行测试用例优先排序.Strandberg 等人[19]对测试用例执行时间及其相关影响因素,如故障检测情况、上次执行的时间间隔和被测代码的修改信息等进行分析组合,通过测试用例集优先权的分配、合并来选择测试用例.

Spieker 等人[2]在 2017 年首次结合强化学习解决持续集成测试优化问题,提出了一种基于强化学习的测试用例优先排序和选择方法 RETECS(reinforced test case selection).图 1 显示了 RETECS 方法的流程,主要包括对测试用例集进行优先级排序,然后从排序后的测试用例集中挑选子集执行(实线框是 RETECS 方法步骤,虚线框是与持续集成环境交互的接口).

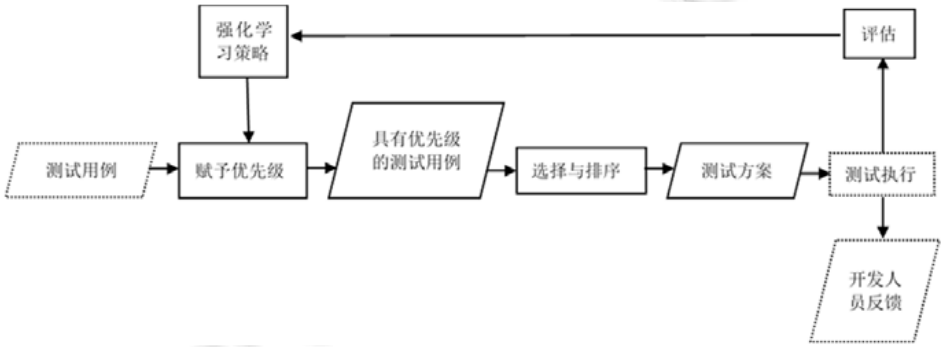


Fig.1 Testing in CI

图 1 面向持续集成的测试

RETECS 方法通过强化学习在每个集成周期后对测试执行结果进行评估,根据结果对测试用例执行的优

先级进行奖励.该方法采用了 3 种奖励函数:Failure Count Reward(FC 奖励)将测试序列在当前集成周期中失效的测试用例总数作为整个测试用例集的奖励值;Test Case Failure Reward(TF 奖励)对失效的测试用例奖励值为 1,通过的测试用例奖励值为 0;Time- ranked Reward(TR 奖励)考虑到测试序列中失效的测试用例的位置.Spieker 的实验结果表明,TF 奖励所得测试序列的检错能力优于其他两种奖励函数.

2 基于测试用例历史信息的 CI 强化学习奖励

针对持续集成测试优化的强化学习方法中奖励函数仅仅考虑当前测试用例执行信息的问题,本文首先引入测试用例在持续集成过程中的全部历史执行信息,提出基于测试用例历史失效次数的奖励函数.在此基础上,本文进一步考虑测试用例失效次数的分布信息,定义了测试用例平均历史失效率,用来度量一个测试用例的在整个持续集成过程中发生失效的分布情况,并在此基础上提出了基于测试用例历史失效率的奖励函数.

在基于强化学习的测试优化中,有两种奖励策略:整体奖励与部分奖励.整体奖励策略对所有的测试用例进行奖励,且奖励值相同;而部分奖励只对失效的测试用例进行奖励,且奖励值不同.整体奖励将排序后的测试序列视为整体,但无法体现单个测试用例对检测故障的贡献.部分奖励可以体现测试用例间的差异,却容易忽略基于测试用例优先级的不同排序策略的不同作用.本文在提出两种奖励函数时,基于整体奖励和部分奖励两种策略分别定义相应的奖励函数.

2.1 基于测试用例历史失效次数的奖励

在持续集成测试的历史过程中,失效次数越多的测试用例错误检测能力越强,应该给予奖励来提高下次执行时的优先级.现有的奖励函数中,只考虑测试序列在当前一次集成周期中的执行失效的情况,没有考虑到测试用例的历史执行情况.由于持续集成中测试用例并非在每次集成都被执行,而测试用例未执行时信息难获取,因此本文关注集成过程中测试用例被执行的信息.本文使用一个元组来表示测试用例的历史执行结果(historical result,简称 HR),用 i 表示集成周期数,对测试用例 t,$HR_i(t)=[r_n,\ldots,r_2,r_1]$表示 t 在第 i 次集成为止,n 次执行的结果,r_n 等于 0 表示 t 在第 n 次执行的结果为通过,r_n 等于 1 表示 t 在第 n 次执行的结果为失效.需要注意的是,一个测试用例不一定会在每次集成中都被执行,所以 n 和 i 并不一定相等.

本文提出了基于历史执行失效次数的奖励函数,定义如下.

定义 1(基于测试用例历史执行失效次数的奖励(historical failure count reward,简称 HFC 奖励)).

- HFC 整体奖励函数:

$$Reward_i^{(HFC\_overall)}(t)=|HR_i^{fail}(t)| \tag{1}$$

- HFC 部分奖励函数:

$$Reward_i^{(HFC\_partial)}(t)=\begin{cases}|HR_i^{fail}(t)|, & t\text{在第}i\text{次集成中失效}\\ 0, & \text{其他}\end{cases} \tag{2}$$

其中,*fail* 表示测试用例失效;$|HR_i^{fail}(t)|$ 表示$[r_n,\ldots,r_2,r_1]$元组中 1 的个数,即测试用例 t 在第 i 次集成时,历史执行过程中失效的总次数.历史执行失效次数奖励不仅包含当前集成周期的执行结果,还包含历史执行结果.整个持续集成测试过程中,失效总次数越多的测试用例将得到更大的奖励值.基于不同的奖励策略,我们分别定义了奖励函数.公式(1)中采用整体奖励,对当前序列中所有测试用例都进行奖励;公式(2)中采用部分奖励,仅对失效的测试用例进行奖励.

2.2 基于测试用例平均历史失效率的奖励

在集成测试中,除了测试用例历史执行中的失效总次数可以评估其错误检测能力,失效发生次序也非常重要.例如在 30 次集成的测试中,两个测试用例同样在 10 次集成中失效了,但一个测试用例是在最初的 10 次集成中失效,另一个是在临近的 10 次集成中失效.比较两者,临近 10 次失效的测试用例在下一次集成中发现错误的可能性更高,应该具有更高的优先级.

考虑到测试用例在历史执行中失效的分布问题,本文提出一个新的衡量指标:测试用例平均历史执行失效

率(average percentage of historical failure,简称 APHF),其计算公式定义如下.

定义 2(测试用例平均历史执行失效率(average percentage of historical failure,简称 APHF)).

$$APHF_i(t)=1-\frac{\sum_{j=1}^{m}R_j}{n\times m}+\frac{1}{2n} \tag{3}$$

其中,i 表示第 i 次集成,n 表示测试用例 t 的历史执行次数,m 表示测试用例 t 在 n 次执行的失效总次数,R_j 表示测试用例最近第 j 次失效在历史执行的倒数次序,$APHF\in(0,1)$.$APHF$ 值越高,表示测试用例在临近的历史执行中失效概率越大,则越可能在下一次执行中检测出错误.

假设测试用例 t_1 与 t_2 历史执行次数均为 n,其历史执行信息分别表示为[1,0,…,0],[0,…,0,1].可以看出,t_1 与 t_2 的历史总失效次数均为 1,t_1 在最近一次执行中失效,t_2 在第一次执行中失效.很可能是由于 t_2 第 1 次执行检测出的 bug 已经被修复,且在最近几次集成中 t_2 没有检测出 bug,而 t_1 在最近一次执行中检测出 bug,时效性更强,那么 t_1 在下一次集成中检测出错误的概率更大,如下所示.

$$APHF(t_1)-APHF(t_2)=\left(1-\frac{1}{n}+\frac{1}{2n}\right)-\left(1-\frac{n}{n}+\frac{1}{2n}\right)=\frac{n-1}{n},\lim_{n\to+\infty}(APHF(t_1)-APHF(t_2))=1.$$

因此,集成周期数趋近正无穷时,测试分布如 t_1 与 t_2 的 APHF 差值趋近于 1,即 APHF 最大值.由此可见,集成周期数越多,测试用例执行次数越多,其历史失效分布对检测错误能力的影响越大.基于此,我们提出测试用例平均历史执行失效率奖励,定义如下.

定义 3(平均历史执行失效率奖励(average percentage of historical failure reward,简称 APHF 奖励)).

- APHF 整体奖励函数:

$$Reward_i^{(APHF\_overall)}(t)=APHF_i(t) \tag{4}$$

- APHF 部分奖励函数:

$$Reward_i^{(APHF\_partial)}(t)=\begin{cases}APHF_i(t), & t在第i次集成中失效\\ 0, & 其他\end{cases} \tag{5}$$

其中,$APHF_i(t)$指测试用例 t 到第 i 次集成为止的平均历史执行失效率.同样地,针对不同的奖励策略,提出 APHF 整体奖励与部分奖励两种奖励函数:公式(4)中采用整体奖励,对当前序列中所有测试用例都进行奖励;公式(5)中采用部分奖励,仅对当前序列中失效的测试用例进行奖励.平均历史执行失效率奖励不仅包含测试用例历史执行的结果信息,还体现了历史执行过程中失效的分布,使得最近失效的测试用例能获得更大的奖励值.

本文所提出的基于测试用例历史执行信息的奖励函数主要有两个优点.

(1) HFC 包含测试用例的历史执行结果信息,而不仅仅是当前一次集成的执行结果.在基于历史的测试用例优先排序技术中,我们认为,如果测试用例在过去检测出错误,则其更易在下次测试中执行到有缺陷的代码.而缺陷预测相关研究表明:在过去有错误的代码很可能再次出错,尤其是这部分代码被修改的情况[20,21].因此,考虑测试用例的历史执行结果而非仅仅是当前一次执行的结果是非常必要的.

(2) APHF 度量了测试用例历史失效的分布,反映了测试用例的时效性,即临近的失效会获得更大的度量值,更符合强化学习序贯决策的特点.

TCP 问题是一个序列决策问题,除了考虑测试用例的执行结果,还应关注其失效的分布,最近出现失效的测试用例更有助于下一次的错误检测.

3 实验分析

为了验证本文所提出基于测试用例历史执行信息的奖励函数的有效性,我们复现了文献[2]中的 RETECS (implementation available at https://bitbucket.org/helges/retecs)实验进行对比,采用 3 个大型工业数据集 Paint Control、IOF/ROl、GSDTSR 为测试对象.

由于持续集成测试具有较强的时限性,测试过程中不能将所有测试用例都执行,因此实验中设置时间阈值为测试集中所有测试用例执行时间的 1/2,即测试用例按照优先级降序排序,直到达到 1/2 总体执行时间停止执

行.为了消除实验中随机因素对实验结果的影响,我们将实验重复 30 次取平均值.下面首先介绍实验所用评估指标(第 3.1 节)与实验对象(第 3.2 节),第 3.3 节介绍实验设计,第 3.4 节对实验结果进行分析.

3.1 评估指标

本实验中采用 NAPFD(normalized average percentage of faults detected)作为评估指标,其计算公式如下.

$$NAPFD(TS_i)=p-\frac{\sum_{t\in TS_i^{fail}} rank(t)}{|TS_i^{fail}|\times|TS_i|}+\frac{p}{2\times|TS_i|} \tag{6}$$

其中, $p=\frac{|TS_i^{fail}|}{|TS_i^{otal,\ fail}|}$,$rank(t)$表示测试序列 TS 中第 t 个失效的测试用例的位置,$|TS_i^{fail}|$表示测试序列 TS 中失效的测试用例总数,$|TS_i|$表示测试序列中测试用例总数,$|TS_i^{total,\ fail}|$表示测试用例集中失效的测试用例总数.

1999 年,Rothermel 等人[17]首次提出平均错误检测率(average percentage of faults detected,简称 APFD)来衡量测试用例优先排序技术的有效性.APFD 根据测试序列中失效的测试用例的序号来衡量测试序列的有效性.当 TCP 技术中存在测试用例选择时,不是所有的测试用例都被执行,不是所有的错误都能被检测到,而 APFD 假设所有的错误都被检测到,因此它只适用于没有测试用例选择的情况.NAPFD[15]是 APFD 的扩展,它能反映测试套件检测到的错误占所有错误的比例,适用于存在测试用例选择的情况.若所有错误都被检测到,NAPFD 则与 APFD 相同(p=1).本实验中计算 NAPFD 值时,我们假设每次失效对应的错误都是不同的.

3.2 实验对象

为了确定本文方法在实际业界的适用性,本文使用工业数据集进行实验:Paint Control 和 IOF/ROL 均来自 ABB Robotics Norway2,用于测试复杂工业机器人;Google Shared Dataset(GSDTSR)是谷歌开源的数据集.这些数据集中包含超过 300 个集成周期中的测试用例执行结果等历史信息.表 1 列出了 3 个数据集的详细信息,包括测试用例集大小、集成周期数等.表中执行结果数指所有测试用例在整个集成过程中被执行的总次数,而失效率表示测试用例被执行的总数中失效的比重.可以看出,Paint Control 数据集规模较小,同时,测试用例失效率近 1/5;IOF/ROL 数据集测试用例集规模中等,但失效率更大,是 3 个数据集中最高的;谷歌开源数据集 GSDTSR 测试用例集规模最大,但失效率仅 0.25%,即在大量的测试用例中仅有少量的失效测试用例,在持续集成测试优化中,把大量测试用例中的少量失效测试用例优先执行,优化难度最大.

Table 1 Statistics of Industrial datasets

表 1 工业数据集信息

数据集	测试用例数	集成周期	执行结果数	失效率(%)
PaintControl	114	312	25 594	19.36
IOF/ROL	2 086	320	30 319	28.43
GSDTSR	5 555	336	126 0617	0.25

3.3 实验设计

为了验证本文提出的 HFC 奖励函数和 APHF 奖励函数在基于强化学习的测试用例优先排序技术中的有效性,本文针对以下 4 个研究问题进行实验与分析.

- 研究问题 1:包含测试用例历史执行信息的奖励函数是否能有效提升测试序列的检错能力?
- 研究问题 2:测试用例历史失效分布信息是否有助于检测到潜在容易失效的测试用例?
- 研究问题 3:整体奖励策略是否优于部分奖励策略?
- 研究问题 4:包含历史信息的奖励函数在时间开销方面是否影响持续集成测试整体的效率?

研究问题 1 的目的是验证本文提出的包含测试用例历史完整失效信息的奖励函数的有效性.本文提出两种包含测试用例历史执行信息的奖励函数,即 HFC 奖励和 APHF 奖励.HFC 奖励是基于测试用例历史执行过程中总的失效次数,APHC 在此基础上进一步考虑失效的分布.所以,为了回答研究问题 1,需要对比基于这两种奖励函数的强化学习方法生成的测试序列和现有方法奖励函数生成测试序列的检错能力.研究问题 2 是为了验

证测试用例历史失效信息分布是否更有助于发现潜在易失效的测试用例,即对比 HFC 和 APHF 两者的优劣.研究问题 3 是为了研究整体奖励(即对当前序列中所有的测试用例进行奖励,且奖励值相同)和部分奖励(即只对当前序列中失效的测试用例进行奖励,且奖励值不同)两种机制对测试用例优先排序技术结果的影响.研究问题 4 对比包含历史信息与不包含历史信息奖励函数的时间消耗.目前,将强化学习用于 TCP 的研究较少, Spieker 等人[2]在 2017 年提出了 RETECS 方法,首次将强化学习与持续集成环境下的 TCP 相结合:强化学习过程中的状态表示单个测试用例的元数据,包括测试用例的执行持续时间、上次执行的时间和历史执行结果;动作表示测试用例当前集成周期中的优先级.测试用例集中所有测试用例都被赋予优先级后,依据其优先级由高到低进行排序.通过对排序后得到的测试序列进行评估,计算奖励值并反馈给智能体,通过奖励值来对先前的行为进行奖励或惩罚,使智能体得以学习,从而正确地进行优先级排序.RETECS 方法中包含 3 种奖励函数(详见第 1 节):TF 奖励、FC 奖励及 TR 奖励.Spieker 的实验结果表明,TF 奖励所得测试序列的检错能力优于其他两种奖励函数.

为了进行实验对比,本文重现了文献[2]中最优的 Test Case Failure Reward(TF 奖励,见公式(7))的实验.

$$\mathrm{Re}\,ward_i^{(TF)}(t)=\begin{cases}1, & t\text{在第}i\text{次集成中失效}\\ 0, & \text{其他}\end{cases} \tag{7}$$

同时,实现了本文提出的基于 HFC 奖励与 APHF 奖励的强化学习算法.实验中采用了基于神经网络(network-based)强化学习智能体,即使用神经网络连续地表示状态与动作.

针对本文提出的 HFC 奖励与 APHF 奖励,我们分别采用两种奖励策略(第 2.1 节)对 HFC 整体奖励函数(公式(1))、HFC 部分奖励函数(公式(2))、APHF 整体奖励(公式(4))、APHF 部分奖励(公式(5))进行了实验.

3.4 实验结果及分析

图 2 是 TF 奖励函数在 Paint Control、IOF/ROL、GSDTSR 这 3 个数据集上的重现实验结果,图中横轴为集成周期数,纵轴为每个集成周期测试序列的 NAPFD 值.NAPFD 值越大,表明测试序列检测错误能力越强.图中曲线为每次集成时,通过强化学习方法生成测试序列的 NAPFD 均值,直线为使用多项式线性拟合的结果.

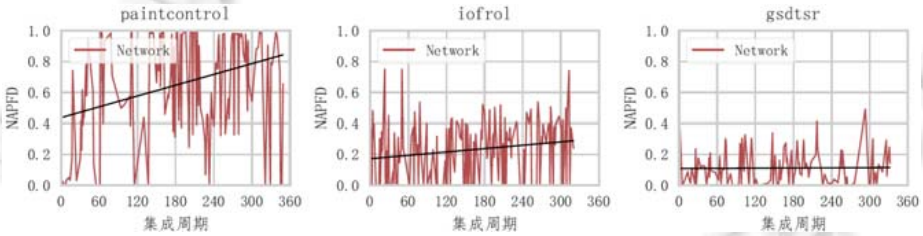
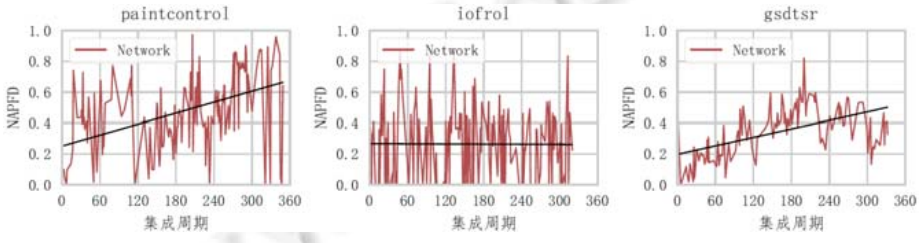


Fig.2 Experimental results of TF reward

图 2 TF 奖励实验结果

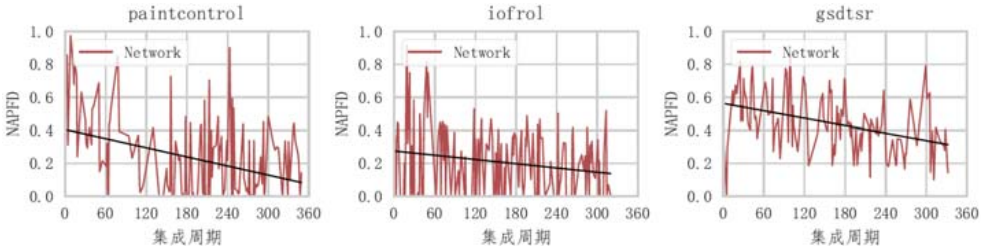
图 3、图 4 分别是本文提出的 HFC 奖励函数和 APHF 奖励函数的实验结果,其中,图(a)、图(b)分别表示部分奖励和整体奖励.相同的,图中横轴与纵轴分别表示集成周期数和相应周期中测试序列的 NAPFD 值.



(a) HFC 部分奖励函数

Fig.3 Experimental results of HFC reward

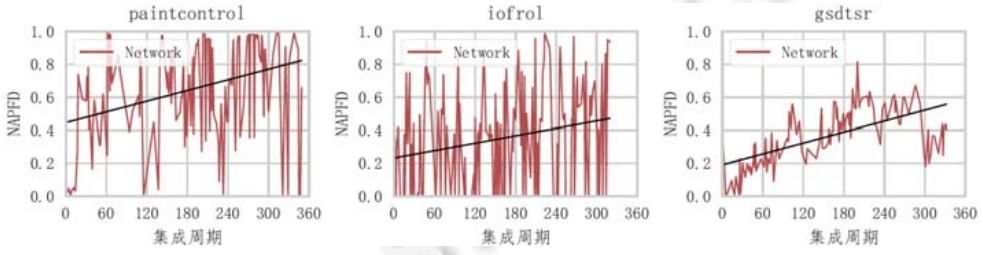
图 3 HFC 奖励实验结果



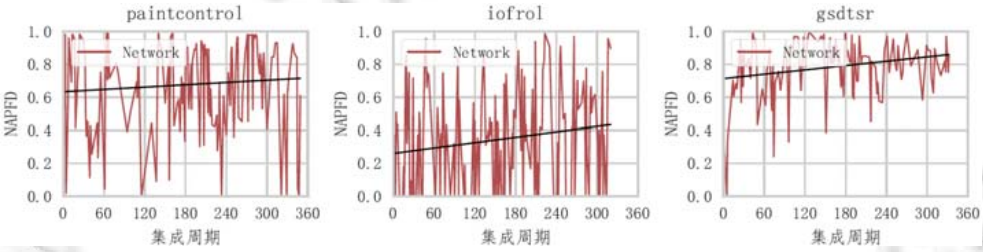
(b) HFC 整体奖励函数

Fig.3 Experimental results of HFC reward (Continued)

图 3 HFC 奖励实验结果(续)



(a) APHF 部分奖励函数



(b) APHF 整体奖励函数

Fig.4 Experimental results of APHF reward

图 4 APHF 奖励实验结果

为进一步说明结果的有效性,我们计算了基于历史的奖励函数(图中只包括 HFC 部分奖励与 APHF 部分奖励)与 TF 奖励的 NAPFD 差值,如图 5 的差值直方图;同时,统计了本文提出的 4 种基于历史信息的奖励函数实验结果,如图 6 箱型图所示.

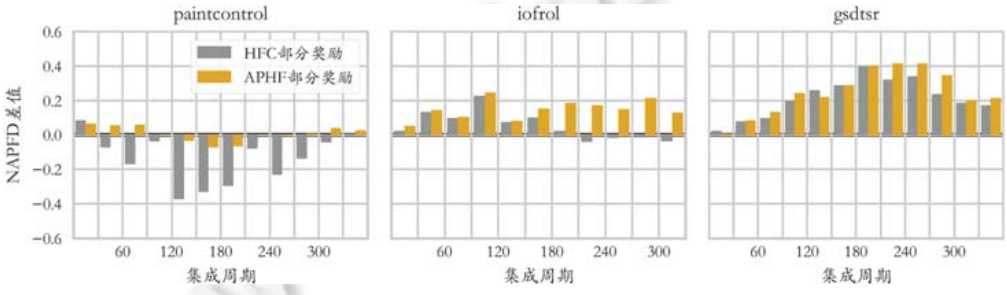


Fig.5 Comparison of experimental results between TF reward and history-based reward

图 5 TF 奖励与基于历史的奖励实验结果比较

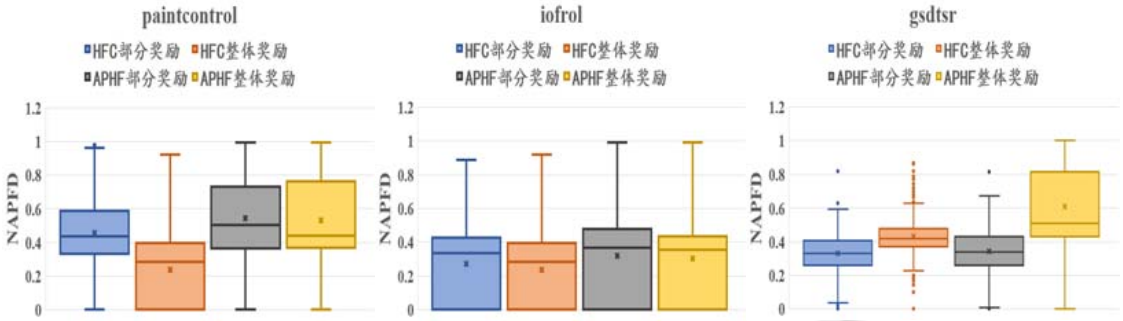


Fig.6 Experimental results of HFC reward and APHF reward

图 6 HFC 奖励与 APHF 奖励实验结果

3.4.1 对研究问题 1 的分析

为了回答研究问题 1,我们使用基于历史失效次数的 HFC 和基于平均历史失效率的 APHF 奖励函数的强化学习方法生成了测试序列,并与使用 RETECS 方法的 TF 奖励函数生成的测试序列进行检错能力的比较.由于 TF 奖励函数是部分奖励,因此对比实验中只对部分奖励策略的实验结果进行对比.

图 3(a)及图 4(a)分别是 HFC 奖励与 APHF 奖励在部分奖励方式下的实验结果,可以看出,随着集成周期的增长,某些情况下,奖励函数在强化学习过程中对测试用例优先级的评定越来越正确,并最终获得检错能力更高的测试序列;但有些时候,学习过程并不有效,测试序列的检错能力保持在初始水平甚至下降.Paint Control 数据集中,使用 APHF 奖励所得测试序列 NAPFD 值与 TF 奖励大致相等,而使用 HFC 奖励所得测试序列 NAPFD 值不到 0.7,略低于 APHF 奖励和 TF 奖励的 0.8,但差距不大.通过检查该数据集,一些测试用例主要在集成前期失效而后期几乎不失效,采用 HFC 奖励提高此类测试用例的优先级,导致排序序列的 NAPFD 值下降.而 APHF 奖励考虑到失效次序,因此不会受到这种情况的影响;IOF/ROL 数据集中,使用 APHF 奖励所得测试序列 NAPFD 值约 0.5,HFC 奖励与 TF 奖励均在 0.3 左右,APHF 奖励显著优于其他两者;而 GSDTSR 数据集中,APHF 奖励与 HFC 奖励对应 NAPFD 值分别为 0.6 与 0.5,TF 奖励则为 0.1.

图 5 显示了 3 种奖励函数在 3 个数据集中的比较结果.我们对 30 次循环中每个集成周期的平均 NAPFD 值的差值进行比较,图中差值为正表明相应的奖励函数表现优于 TF 奖励函数,差值为为负说明 TF 奖励函数表现优于被比较的奖励函数.从图中可以看出,在 PaintControl 数据集上,HFC 奖励表现不如 TF 奖励,但 APHF 奖励与 TF 奖励大致相同;而在 IOF/ROL 与 GSDTSR 数据集上,HFC 奖励与 APHF 奖励明显优于 TF 奖励,与前面分析相符.

综合以上实验结果,APHF 奖励在 3 个数据集上表现均优于 TF 奖励;针对 HFC 奖励,除了 Paint Control 数据集中 HFC 奖励表现差于 TF 奖励,其他两个数据集中 HFC 奖励也均优于 TF 奖励.总的来说,包含历史信息的奖励函数明显优于不包含历史信息的奖励函数,尤其是对于规模最大但失效率仅为 0.25%的谷歌 GSDTSR 数据集,测试用例的 NAPFD 值提高近 6 倍,说明即使在失效信息少的情况下,基于历史信息也能学习出检错能力较高的测试序列,进一步验证了历史失效信息对测试用例优先排序技术的有效性.

3.4.2 对研究问题 2 的分析

研究问题 2 的目的是验证测试用例历史失效分布信息有助于发现潜在易失效的测试用例,即对比 HFC 和 APHF 两者的优劣.图 3(a)与图 4(a)分别是 HFC 奖励与 APHF 奖励在部分奖励方式下的实验结果,从图中可以看出,采用部分奖励机制时,在 3 个数据集中,APHF 奖励所得测试序列的 NAPFD 值均高于 HFC 奖励,NAPFD 值最多提高了 0.2(IOF/ROL 数据集).在 GSDTSR 数据集上提高效果不显著,可能是由其测试用例失效率低,所含失效分布信息少导致.

图 3(b)和图 4(b)分别是 HFC 奖励与 APHF 奖励在整体奖励方式下的实验结果,由图可知,采用整体奖励机制时,APHF 奖励所得测试序列的 NAPFD 值明显高于 HFC 奖励:PaintControl 数据集上,NAPFD 值由 0.4 提高至

0.7,GSDTSR 数据集上由不到 0.5 提升至 0.85.

从图 6,即 HFC 奖励与 APHF 奖励的箱型图中也可以看出,在 3 个数据集上,采用同种奖励机制时,APHF 奖励均优于 HFC 奖励;而在 GSDTSR 数据集上,APHF 整体奖励较 HFC 整体奖励提高效果尤为明显.

因此可以得出:包含测试用例历史失效分布信息的奖励函数所得测试序列检错能力强于不包含失效分布信息的奖励函数,证明了测试用例历史执行失效分布有助于发现潜在易失效的测试用例.

3.4.3 对研究问题 3 的分析

研究问题 3 分析两种奖励策略的优劣.

图 3(a)、图 3(b)分别是 HFC 部分奖励函数与整体奖励函数在 3 个数据集上的实验结果.对于 HFC 奖励,总体来说,部分奖励的结果在 3 个数据集上均明显优于整体奖励.而图 6 中,GSDTSR 数据集上 HFC 部分奖励的 NAPFD 虽然略高于整体奖励,但离群点较多,且图 3(b)中斜率为负,说明 HFC 部分奖励表现并不好.通过分析,HFC 是基于测试用例历史失效总次数的奖励,部分奖励可以加大当前集成周期中失效测试用例的奖励力度,NAPFD 值有明显提升,而整体奖励针对当前序列中所有的测试用例都进行奖励,缩小了当前失效测试用例与通过测试用例的差距,反而使结果更差.因此针对 HFC 奖励函数,部分奖励策略更有效.

图 4(a)、图 4(b)分别是 APHF 部分奖励函数与整体奖励函数在 3 个数据集上的实验结果.综合分析图 4 与图 6 可知,对于 APHF 奖励,在前两个数据集上两种奖励策略结果近似,但部分奖励机制略优于整体奖励,说明部分奖励可以增大近期失效的测试用例所得奖励.而在谷歌开源数据集 GSDTSR 上,整体奖励机制明显优于部分奖励.综合分析 GSDTSR 数据集中测试用例失效率很低,关于失效分布的历史信息较少,因此采用部分奖励时进一步损失了历史失效信息,导致每个测试用例都难以得到奖励;相反,整体奖励避免了历史失效信息的损失,反而能增大测试用例得到的奖励,因此,整体奖励适用于失效率较低的测试集.

总的来说,不同的奖励函数适用的奖励策略不同,同时还可能受到测试集中测试用例失效信息的影响.

3.4.4 对研究问题 4 的分析

研究问题 4 主要分析不同奖励的函数的时间消耗是否会影响持续集成测试整体效率.基于强化学习的 TCP 过程可大致包括通过奖励函数计算测试用例执行优先级并基于优先级进行排序,排序过程采用贪心策略,对总时间影响不大,因此,本实验统计一次集成的总时间进行比较,结果见表 2.可以看出,包含历史的奖励函数由于增加了历史信息的计算,时间消耗多于不包含历史的奖励策略.其中,HFC 奖励时间消耗比 TF 奖励增加约 1s,APHF 增加约 1.5s.而整体奖励策略由于计算量多于部分奖励策略,因此时间消耗也略有增加.

Table 2 Time consumption for different reward functions (s)

表 2 不同奖励函数的时间消耗 (s)

数据集	TF 奖励	HFC 整体奖励	HFC 部分奖励	HFC 平均值	APHF 整体奖励	APHF 部分奖励	APHF 平均值
PaintControl	0.012 1	1.559 2	1.135 2	1.347 2	1.669 6	0.977 0	1.323 3
IOF/ROL	0.021 0	0.704 4	0.530 2	0.617 3	1.823 9	2.338 3	2.081 1
GSDTSR	1.535 0	2.353 5	2.337 1	2.345 3	2.522 0	2.493 3	2.507 7
平均值	0.522 7	1.539 0	1.334 2	1.436 6	2.005 2	1.936 2	1.970 7

总的来说,包含历史信息的奖励函数时间开销增大但都在秒级,最长时间不超过 3s.同时,由于错误检测率有明显提高,对于一个工业程序,在秒级内的时间开销增长是可接受的.

4 结论与展望

本文针对持续集成测试优化的强化学习方法,从奖励函数设计和奖励策略两方面研究强化学习的奖励机制,提出了包含测试用例历史执行信息的奖励函数,用于基于强化学习的持续集成测试优化.基于测试用例历史执行信息,考虑到其历史失效分布对排序结果的影响,定义了平均历史执行失效率,并提出了两种奖励函数——基于测试用例历史执行失效次数奖励与基于测试用例平均历史执行失效率奖励.HFC 奖励与 APHF 奖励不仅包含测试用例当前执行结果,还包括其历史执行结果,APHF 考虑了测试用例历史失效的分布因素.在 3 个大型工业数据集上进行对比实验,结果表明:(1) 相对于不含历史信息的奖励函数,使用包含测试用例历史失效信息

的奖励函数,所得测试序列的检错能力得到明显提高;(2) 测试用例历史失效分布信息有助于发现潜在易失效的测试用例;(3) 整体奖励与部分奖励两种奖励策略受到被测程序的多种因素影响,需要根据实际情况具体选择;(4) 包含历史信息的奖励函数时间消耗有所增加,但对于都在秒级,不会对持续集成测试效率造成很大影响.

在未来工作中,可以考虑:(1) 基于强化学习的多目标的测试用例优先排序技术;(2) 使用测试用例的其他信息,如覆盖信息、修改信息、相似度信息等;(3) 结合大规模神经网络与深度学习方法,对强化学习的智能体进行优化.

References:

[1] Busjaeger B, Xie T. Learning for test prioritization: An industrial case study. In: Foundations of Software Engineering. 2016. 975−980.

[2] Spieker H, Gotlieb A, Marijan D, *et al*. Reinforcement learning for automatic test case prioritization and selection in continuous integration. In: Proc. of the Int'l Symp. on Software Testing and Analysis. 2017. 12−22.

[3] Bian Y, Yuan F, Guo JX, Li Z, Zhao RL. CPU+GPU heterogeneous computing orientated multi-objective test case prioritization. Ruan Jian Xue Bao/Journal of Software, 2016,27(4):943−954 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4968.htm [doi: 10.13328/j.cnki.jos.004968]

[4] Wong WE, Horgan JR, London S, Agrawal H. A study of effective regression testing in practice. In: Proc. of the Int'l Symp. on Software Reliability Engineering. 1997. 264−274.

[5] Rothermel G, Untch RH, Chu C, Harrold MJ. Prioritizing test cases for regression testing. IEEE Trans. on Software Engineering, 2001,27(10):929−948.

[6] Yoo S, Harman M. Regression testing minimization, selection and prioritization: A survey. Software Testing, Verification & Reliability, 2012,22(2):67−120.

[7] Li Z, Harman M, Hierons RM. Search algorithms for regression test case prioritization. IEEE Trans. on Software Engineering, 2007, 33(4):225−237.

[8] Li Z, Bian Y, Zhao RL, *et al*. A fine-grained parallel multi-objective test case prioritization on GPU. In: Proc. of the Symp. on Search Based Software Engineering. 2013. 111−125.

[9] Bian Y, Li Z, Zhao RL, *et al*. Epistasis based ACO for regression test case prioritization. IEEE Trans. on Emerging Topics in Computational Intelligence, 2017,1(3):213−223.

[10] De Souza LS, De Miranda PB, Prudencio RB, Barros FA. A multi-objective particle swarm optimization for test case selection based on functional requirements coverage and execution effort. In: Proc. of the Int'l Conf. on Tools with Artificial Intelligence. 2011. 245−252.

[11] Yu L, Xu L, Tsai WT. Time-constrained test selection for regression testing. In: Proc. of the Advanced Data Mining and Applications. Springer-Verlag, 2010. 221−232.

[12] Chen S, Chen Z, Zhao Z, *et al*. Using semi-supervised clustering to improve regression test selection techniques. In: Proc. of the Int'l Conf. on Software Testing Verification and Validation. 2011. 1−10.

[13] Groce A, Fern A, Pinto J, Bauer T, Alipour A, Erwig M, Lopez C. Lightweight automated testing with adaptation-based programming. In: Proc. of the Int'l Symp. on Software Reliability Engineering. 2012. 161−170.

[14] Reichstaller A, Eberhardinger B, Knapp A, Reif W, Gehlen M. Risk-based interoperability testing using reinforcement learning. In: Proc. of the Int'l Conf. on Testing Software and Systems. 2016. 52−69.

[15] Marijan D, Gotlieb A, Sen S, *et al*. Test case prioritization for continuous regression testing: An industrial case study. In: Proc. of the Int'l Conf. on Software Maintenance. 2013. 540−543.

[16] Noor TB, Hemmati H. A similarity-based approach for test case prioritization using historical failure data. In: Proc. of the 2015 IEEE 26th Int'l Symp. on Software Reliability Engineering (ISSRE). IEEE, 2015. 58−68.

[17] Elbaum SG, Rothermel G, Penix J, *et al*. Techniques for improving regression testing in continuous integration development environments. In: Foundations of Software Engineering. 2014. 235−245.

[18] Ammar A, Baharom S, Ghani AA, Din J. Enhanced weighted method for test case prioritization in regression testing using unique priority value. In: Proc. of the Int'l Conf. on Information Systems Security. 2016. 1−6.

[19] Strandberg PE, Sundmark D, Afzal W, Ostrand TJ, Weyuker EJ. Experience report: Automated system level regression test prioritization using multiple factors. In: Proc. of the Int'l Symp. on Software Reliability Engineering. 2016. 12–23.

[20] Zimmermann T, Premraj R, Zeller A. Predicting defects for eclipse. In: Proc. of the 3rd Int'l Workshop on Predictor Models in Software Engineering (PROMISE 2007). IEEE, 2007.

[21] Noor T, Hemmati H. Test case analytics: Mining test case traces to improve risk-driven testing. In: Proc. of the 2015 IEEE 1st Int'l Workshop on Software Analytics (SWAN). IEEE, 2015. 13–16.

附中文参考文献:

[3] 边毅,袁方,郭俊霞,李征,赵瑞莲.面向 CPU+GPU 异构计算的多目标测试用例优先排序.软件学报,2016,27(4):943−954. http://www.jos.org.cn/1000-9825/4968.htm [doi: 10.13328/j.cnki.jos.004968]

何柳柳(1995−),女,四川达州人,硕士生,CCF 学生会员,主要研究领域为软件测试,持续集成测试.

李征(1974−),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为基于搜索的软件工程,软件测试,源代码分析.

杨羊(1991−),女,博士生,CCF 学生会员,主要研究领域为软件工程,软件测试,持续集成测试.

赵瑞莲(1964−),女,博士,教授,博士生导师,CCF 高级会员,主要研究领域为软件测试.