计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2022,31(2):325−334 [doi: 10.15888/j.cnki.csa.008300]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于相似性的 CITCP 强化学习奖励策略①

杨　羊, 潘超月, 曹天歌, 李　征

(北京化工大学 信息科学与技术学院, 北京 100029)
通信作者: 李　征, E-mail: lizheng@mail.buct.edu.cn

摘　要: 在面向持续集成测试用例优先排序 (continuous integration test case prioritization, CITCP) 的强化学习方法中, 智能体通过对测试用例实施奖励从而调整测试用例优先排序策略以适应后续集成测试, 可以满足持续集成测试频繁迭代和快速反馈的需求. 智能体通常只奖励执行失效测试用例, 但实际工业程序持续集成测试具有集成高频繁但测试低失效的特点, 对 CITCP 的实际应用提出新的挑战. 测试低失效, 即稀少的执行失效测试用例数量, 会导致强化学习中奖励对象稀少, 引发强化学习的稀疏奖励问题. 本文研究一种强化学习奖励对象选择策略, 在奖励执行失效测试用例的基础上, 通过选择与执行失效测试用例相似的执行通过测试用例实施奖励, 从而增加奖励对象, 以解决奖励稀疏问题. 研究具体包括, 设计了一种测试用例历史执行信息序列和执行时间特征向量表示的相似性度量方法, 并基于相似性度量选择与执行失效测试用例集相似的执行通过测试用例集实施奖励. 在 6 个工业数据集上开展了实验研究, 结果表明基于相似性的奖励对象选择策略通过增加有效奖励对象解决了稀疏奖励问题, 并进一步提高了基于强化学习的持续集成测试用例优先排序质量.

关键词: 持续集成测试; 强化学习; 测试用例优先排序; 相似性; 奖励对象选择策略; 稀疏奖励

引用格式: 杨羊,潘超月,曹天歌,李征.基于相似性的 CITCP 强化学习奖励策略.计算机系统应用,2022,31(2):325–334. http://www.c-s-a.org.cn/1003-3254/8300.html

Similarity-based Reward Strategy of Reinforcement Learning in CITCP

YANG Yang, PAN Chao-Yue, CAO Tian-Ge, LI Zheng

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

Abstract: In the reinforcement learning method for the continuous integration test case prioritization (CITCP), the agent rewards the test cases to realize the adjustment of test case prioritization strategy, and thus they can meet the needs of frequent iteration and rapid feedback in continuous integration testing. The agent usually only rewards the failure test cases. However, in the actual industrial processes, the continuous integration testing features high-frequency integration and low-failure-rate tests, which poses a new challenge to the actual application of CITCP. Low-failure-rate tests can be understood as a sparse number of failure test cases, which can lead to the sparsity of reward objects in reinforcement learning and bring about the sparse reward problem. In this study, a reward object selection strategy is proposed to solve the sparse reward problem. With the failure test cases rewarded, passing test cases similar to failure test cases are selected to be rewarded, and thus the number of reward objects increases. Specifically, the similarity measure method for test cases is designed with the feature vector representation of historical execution information sequences and duration time. Then, the passing test cases similar to the failure test cases are selected to be rewarded through the similarity measure. The experiments are conducted in six industrial data sets, and the results show that the similarity-based reward object selection strategy can effectively solve the sparse reward problem by increasing the reward objects and further improve the quality of reinforcement learning-based CITCP.

① 基金项目: 国家自然科学基金 (61872026)
收稿时间: 2021-04-10; 修改时间: 2021-05-11; 采用时间: 2021-05-19; csa 在线出版时间: 2022-01-17

Key words: continuous integration testing; reinforcement learning; test case prioritization; similarity; reward object selection strategy; sparse reward

1 引言

持续集成环境允许代码频繁地集成到主干上以进行软件的更改[1]. 持续集成测试用例优先排序针对每次集成连续地进行测试用例执行序列的调整, 以保障持续集成的每次修改没有引入新的错误[2]. 基于强化学习的持续集成测试用例优先排序技术[3], 通过历史经验的学习自适应地进行测试用例优先排序策略的调整, 以适应持续集成环境的变化, 其框架被定义为 reinforced test case selection (RETECS).

强化学习与持续集成测试用例优先排序的交互如图 1 所示. 强化学习主要包括环境、智能体、动作和奖励 4 个元素[4]. 针对持续集成测试用例优先排序, 环境是持续集成周期, 环境中的状态是在每一集成周期提交的测试用例的元数据, 包括测试用例预计运行时间和测试用例历史执行结果等; 智能体是依据历史经验进行测试用例优先排序决策的中枢; 动作是智能体根据历史学习的策略进行当前持续集成周期测试用例优先级排序结果的返回; 奖励是对当前测试用例优先排序策略的反馈, 通过评价当前集成周期测试用例执行结果以实现奖励.

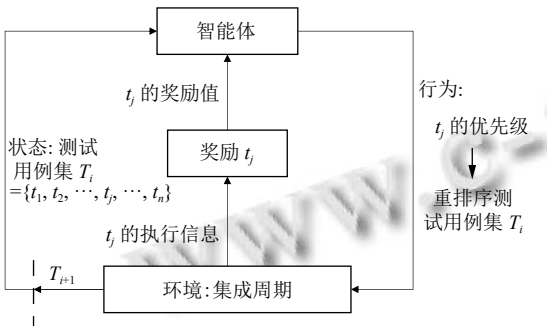

图 1 强化学习与持续集成测试用例优先排序交互图

强化学习是一种基于奖励机制的机器学习方法, 核心是通过奖励函数对当前行为进行有效的评估, 并反馈给智能体选择合适的后续行为, 以实现期望的最大化. 基于强化学习的持续集成测试用例优先排序技术, 通过对测试用例实施奖励以实现测试用例优先排序策略的调整, 从而最终提高持续集成测试用例优先排序的质量. 测试用例的奖励包括奖励函数设计方法和奖励对象选择策略. Yang 等人[5] 系统化研究了奖励函数设计方法, 提出了基于历史信息的奖励函数, 通过提取测试用例历史信息序列的特征以实现测试用例检错能力的度量, 可以实现测试用例优先排序效果的优化.

奖励对象选择策略通常选择执行失效测试用例作为奖励对象实施奖励. 然而, 实际工业软件持续集成测试具有集成高频繁和测试低失效的特点. 例如, 在 Google 开源数据集 GSDTSR 中, 共集成 336 个周期, 其中 284 个周期存在执行失效, 涉及 5 555 个测试用例的 1 260 618 次测试执行, 其中失效执行只占 0.25%. 在基于执行失效的奖励对象选择策略下, 持续集成测试的低失效会引发奖励对象稀少, 从而导致强化学习奖励稀疏问题, 减缓了强化学习速度, 甚至可能导致智能体生成高随机的策略[6], 难以真正应用于实际工业生产环境中. 如何解决基于强化学习的持续集成测试用例优先排序技术在实际工业环境中产生的奖励稀疏问题, 是本文的研究重点.

奖励稀疏问题的核心是奖励对象稀少, 可以通过增加奖励对象来解决奖励稀疏问题. 测试用例优先排序旨在优先执行潜在失效测试用例. 当前执行失效的测试用例, 基于失效测试与测试失效的相关性[7], 在后续测试执行中具有较大的失效可能性, 该理论不适用于执行通过的测试用例. 在测试用例优先排序中考虑测试用例之间的依赖关系, 理论上会使排序结果更优, 但依赖性分析特别是大规模程序的依赖性分析是一个非常耗时的过程, 难以满足持续集成测试的快速反馈需求. 在测试用例优先排序中考虑测试用例之间的相似关系, 即基于相似性的测试用例优先排序[8], 通过测试用例间的相似性度量, 优先执行有相同特征的测试用例, 可以有效提高测试序列的检错能力. 因此, 本文基于相似测试用例的假设, 提出一种测试用例相似性的度量方法, 并选择与失效测试用例集相似的通过测试用例实施奖励, 以解决奖励对象稀少导致的强化学习奖励稀疏问题.

根据集成测试的快速反馈需求, Bagherzadeh 等人[9] 认为持续集成测试用例优先排序的最优序列为检错能力高并且执行时间短的序列, 即优先执行检错能

力高的测试用例, 在相同的检错能力下优先执行运行时间短的测试用例. 因此, 本文针对测试用例相似性度量考虑以下两个因素. 首先, 测试用例的历史执行信息序列相似的测试用例具有相似的检错能力, 即两个测试用例在历史执行过程中执行结果相似, 在后续测试中执行结果可能相同. 所以选择与失效测试用例历史执行信息序列相似的执行通过测试用例实施奖励, 有助于在后续测试中提升高检错能力测试用例的执行顺序. 其次, 在第一个条件的基础上, 具有相似执行时间的测试用例有利于优化测试执行时间, 从而提升反馈速度.

本文通过历史信息序列特征和执行时间设计了一个多维度相似性度量方法, 通过增加奖励与执行失效测试用例相似的执行通过测试用例, 解决强化学习奖励稀疏问题, 从而实现基于强化学习的持续集成测试用例优先排序技术在实际工业界的应用. 最终, 在大规模工业数据集上验证了基于强化学习的持续集成测试用例优先排序在基于相似性的奖励对象选择策略下的有效性.

2 基于相似性的奖励对象选择策略

基于强化学习的持续集成测试用例优先排序通过历史经验的学习, 尽可能多地奖励潜在失效测试用例, 通过测试用例优先排序策略的调整以实现在后续测试中优先执行失效测试用例. 基于失效测试与测试失效的相关性[7], 当前执行失效测试用例在后续测试中具有更大的潜在失效可能性, 智能体对其实施奖励有助于提高测试用例优先排序效果. 然而实际工业程序中低失效的持续集成测试导致基于强化学习的持续集成测试用例优先排序奖励对象稀少, 即奖励稀疏, 减缓了强化学习的速度, 甚至形成难以收敛的高随机策略.

为了解决强化学习的奖励稀疏问题, 在奖励失效测试用例的基础上, 基于测试用例相似性假设[8], 本文针对执行通过测试用例与执行失效测试用例进行相似性分析, 从而实现对执行通过测试用例实施奖励, 并提出基于相似性的奖励对象选择策略.

2.1 持续集成测试用例特征分析

基于相似性的测试用例优先排序技术在回归测试中证明了存在相似性的测试用例拥有相似的错误检测率[8], 并且具有潜在失效可能的测试用例倾向于聚集在连续的区域[10].

相似性度量的概念是许多研究领域的研究主题. 两个物体共有特征的程度被称为相似性, 它们不同的程度被称为距离. 许多软件测试文献量化了测试用例之间的相似性, 例如在基于覆盖的测试优化中, 使用覆盖信息进行测试用例之间相似性的度量[11]. 测试用例的相似性度量因素还涉及测试路径[12,13]、测试用例相关源代码[14]、测试输入和输出[15]、从测试脚本中提取的主题模型[16]、测试用例的历史失效信息[17] 等.

在基于强化学习的持续集成测试用例优先排序中, 智能体根据历史学习的策略针对测试用例元数据进行识别以产生测试用例优先排序结果. 测试用例元数据通过持续集成测试日志的提取, 包括 Cycle、ID、Duration、LastRun、LastResults 和 Verdict, 这些属性及其解释如表 1 所示.

表 1 测试用例元数据

属性名	内容
Cycle	所在持续集成周期
ID	测试执行的唯一数字标识符
Duration	测试用例的大概执行时间
LastRun	测试用例上一次执行的时间点
LastResults	测试用例的历史执行结果
Verdict	当前测试结果

针对在同一周期的测试用例, 根据当前执行结果的不同, 即 Verdict 的不同, 可以有效地识别失效测试用例对被测系统的失效影响. 对于执行通过测试用例, 基于 Verdict 无法度量其失效影响, 然而其历史执行结果 LastResults 可以描述其在历史执行过程中的失效情况. 针对历史执行过程相似的测试用例, 其在未来执行中可能有相同的测试结果. 因此, 可以基于测试用例的历史执行过程进行测试用例的相似性度量.

持续集成测试是一种黑盒测试方法, 在快速反馈需求下, 基于黑盒测试的测试用例执行时间, 在相近测试内容下其执行时间非常接近. 测试用例优先排序旨在优先执行潜在失效测试用例. 在实际工业程序中, 潜在失效的测试用例可能与执行失效测试用例不完全相同, 但是非常相似, 例如, 当新测试用例是旧失效测试用例的修改版本时, 新测试用例可以捕捉未检测到的系统故障, 但是当前执行通过. 新测试用例与旧测试用例的执行时间 Duration 非常接近或者相同. 因此新测试用例由于与旧测试用例相似, 并且具有类似的执行时间, 具有相似的失效影响, 在后续测试中应该被优先

执行.

基于测试失效的假设, 失效测试用例对被测系统具有失效影响, 智能体对其实施奖励. 本文通过提取测试用例的历史执行信息序列 LastResults 与测试执行时间 Duration 进行执行通过测试用例与执行失效测试用例集的相似性度量, 以识别执行通过测试用例集中具有失效影响的测试用例, 从而实现奖励的实施.

2.2 测试用例相似性度量方法

测试用例的相似性度量, 通过提取测试用例的特征以实现测试用例间的相似性计算. 本文针对持续集成测试用例使用历史执行信息序列 LastResults 和执行时间 Duration 信息, 进行测试用例的特征向量定义, 如定义 1 所示.

定义 1. 测试用例的特征向量 (feature vector of test case).

$$Feature_i(t)=\left(\frac{t.duration_i}{totalTime_i},t.lastresults_i\right) \quad (1)$$

其中, $totalTime_i$ 表示持续集成 i 周期的计划执行时间, $t.duration_i$ 表示测试用例 t 在 i 周期的执行时间, $\frac{t.duration_i}{totalTime_i}$ 是测试用例 t 在 i 周期执行时间的归一化, $t.lastresults_i$ 表示测试用例 t 在 i 周期测试执行后的历史执行信息序列.

测试用例的特征向量通过测试用例的特征提取以实现测试用例间的相似性度量. 本文通过测试用例的历史执行信息序列和执行时间的属性, 进行测试用例的特征提取, 定义了测试用例的特征向量, 并进一步根据特征向量进行两两测试用例间的特征距离计算以实现相似性计算.

相似性函数通过两两测试用例间的特征距离计算以度量测试用例相似程度. 相似性距离计算的常用方法包括杰卡德距离[11]、Gower-Legendre[18]、Sokal-Sneath[19]、欧氏距离[20–22]、余弦相似性[23] 和比例二进制距离[21,24,25] 等. Wang 等人[26] 通过深入研究了这 6 种常用相似性计算公式对基于相似性的测试用例优先排序效果的影响, 研究表明欧氏距离比其他相似性距离计算公式更有助于进行测试用例优先排序, 从而更有效地发现程序错误. 因此, 本文采用欧氏距离进行持续集成测试的测试用例相似性函数计算, 通过特征向量的逐一比较实现测试用例的相似性度量, 其相似性计算函数如定义 2 所示.

定义 2. 测试用例相似性函数 (test case similarity, TCS).

$$TCS_i(t_1,t_2)=\sqrt{\sum_{j=1}^{n}\left(Feature_i\left(t_{1j}\right)-Feature_i\left(t_{2j}\right)\right)^2} \quad (2)$$

其中, t_1 表示执行通过测试用例, t_2 表示执行失效测试用例, $Feature_i(t_{1j})$ 表示测试用例 t_1 在第 i 集成周期的特征向量的第 j 个特征, n 是测试用例 t_1 和 t_2 的最短有效历史执行序列长度. 由于在持续集成测试中测试用例出现的频率不同, 导致测试用例的历史执行信息序列长度不同. 为了有效地进行特征向量的比较, 基于失效测试与测试失效的相关性[7], 本文依据测试用例中历史信息序列较短的序列长度, 选择临近历史信息序列进行相似性的计算, 从而实现全面测试用例相似性比较.

测试用例相似性函数通过两两测试用例的特征向量距离计算, 以实现测试用例的相似性度量. 距离越小, 即相似性值越低, 表明这两个测试用例越相似.

2.3 基于相似性的奖励对象选择方法

基于失效测试与测试失效的相关性[7], 失效测试用例具有较高的检错能力, 在后续测试中需要进行优先级的提升, 因此智能体优先对失效测试用例实施奖励以实现测试用例优先排序策略的调整, 使其适应后续持续测试. 然而, 持续集成测试的低失效特征, 即失效测试执行稀少, 导致基于强化学习的持续集成测试用例优先排序奖励对象稀少, 导致奖励稀疏问题, 影响了强化学习的学习速度和学习质量.

在持续集成测试中, 对于当前执行通过测试用例, 一方面存在频繁失效、偶然通过的测试用例, 另一方面存在测试用例是旧测试用例的修复版本. 这些测试用例在后续测试中存在失效可能性, 因此具有一定的失效影响, 然而无法基于当前执行结果进行有效的度量. 为了有效度量执行通过测试用例的失效影响, 本文通过与执行失效测试用例的相似性分析, 以选择执行通过测试用例集中与执行失效测试用例相似的测试用例实施奖励, 进一步提出基于相似性的奖励对象选择策略. 智能体针对执行通过测试用例, 使用与执行失效测试用例的相似性分析方法, 实现执行通过测试用例的失效影响分析, 从而增加奖励与执行失效测试用例相似的执行通过测试用例.

针对在某一集成周期的测试用例集, 在测试执行后, 智能体首先对失效测试用例进行奖励, 进一步将执

行通过测试用例逐一与失效测试用例集进行相似性分析以寻找具有失效影响的执行通过测试用例. 若执行通过测试用例与执行失效测试用例存在相似性, 则该执行通过测试用例具有失效影响, 智能体应该奖励该测试用例, 否则执行通过测试用例不被奖励. 其执行流程如图 2 所示. 根据其执行过程, 本文进一步定义了基于相似性的奖励对象选择策略如定义 3 所示.

定义 3. 基于相似性的奖励对象选择策略 (similarity-based reward object selection strategy).

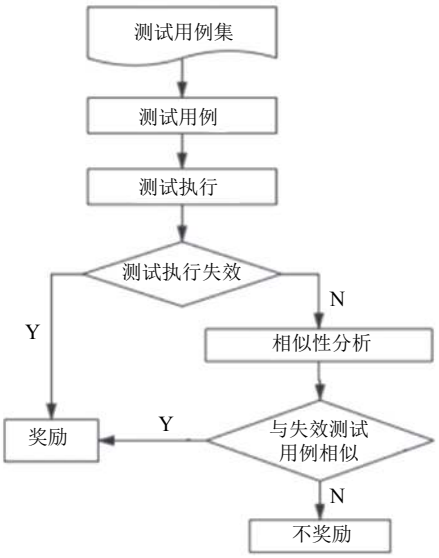

图 2 基于相似性的奖励对象选择策略执行过程

对于第 i 个周期的测试套件 $T_i=\{t_1,t_2,\cdots,t_j,\cdots,t_n\}$, 如果测试用例 t_j 执行失效, 则奖励测试用例 t_j; 若测试用例 t_j 执行通过, 但测试用例 t_j 与 T_i 中的失效测试用例t'存在相似性, 即$TCS_i\left(t_j,t'\right)<\varepsilon$, 则奖励测试用例 t_j; 其他情况智能体不奖励测试用例.

基于相似性的奖励对象选择策略的奖励方式, 如式 (3) 所示.

$$Reward\_Sim(t)=\begin{cases}rewardvalue,\ t.verdict=1\\ rewardvalue,\ t.verdict=0\&TCS(t,t')<\varepsilon\\ \qquad\qquad \&t'.verdict=1\\ 0,\ \text{else}\end{cases} \tag{3}$$

其中, *rewardvalue* 是基于任意奖励函数计算的测试用例的奖励值, *t.verdict* 表示测试用例 t 的执行结果, 1 表示执行失效, 0 表示执行通过, t'是该周期中任意执行失效测试用例.

本文通过持续集成测试用例的特征分析, 提出考虑测试用例历史执行信息序列和执行时间的多维度测试用例特征向量表示方法, 并进一步基于特征向量进行测试用例间的相似性度量. 基于测试用例的相似性特征, 本文提出基于相似性的奖励对象选择策略, 在奖励失效测试用例的基础上, 通过相似性的判断, 自适应地选择执行通过测试用例进行奖励以解决在实际工业程序中的奖励稀疏问题.

3 实验设计及结果分析

基于强化学习的持续集成测试用例优先排序过程如图 3 所示, 其中虚线部分存在于持续集成环境中, 实线部分存在于强化学习框架中. 对于每个集成周期中提交的测试用例集, 强化学习的智能体根据历史学习的测试用例优先排序策略定义测试用例的优先级, 进一步根据快速反馈的时间约束选择测试用例以进行测试排序从而实现测试执行, 最后根据执行结果进行评价并反馈给开发人员和强化学习的智能体. 智能体根据反馈的结果选择测试用例实施奖励, 即奖励对象选择策略, 并进一步根据奖励的结果为后续集成测试调整测试用例优先排序策略, 使其适应于持续集成系统的连续变化.

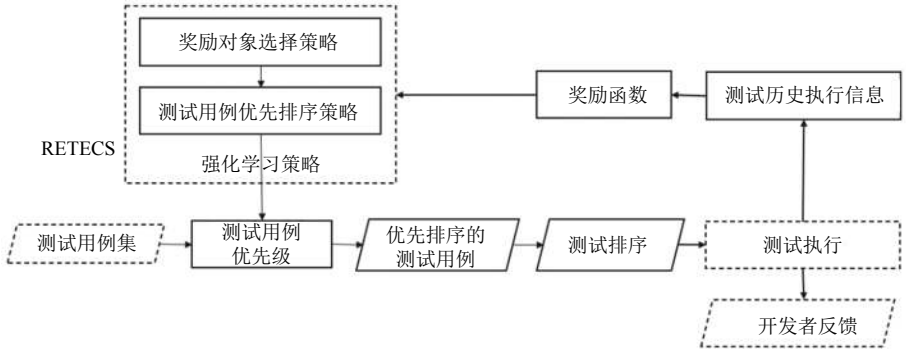

图 3 基于强化学习的持续集成测试过程

3.1 研究问题

本文针对在实际工业程序存在集成高频繁但是测试低失效的特征, 在基于强化学习的持续集成测试用例优先排序中研究奖励对象稀少导致强化学习的奖励稀疏问题, 提出基于相似性的奖励对象选择策略. 本文从以下 3 个研究问题进行实验的分析和验证.

问题 1. 在基于相似性的奖励对象选择策略中, 如何进行相似性阈值的选取, 从而实现持续集成测试用例优先排序?

问题 2. 基于相似性的奖励对象选择策略对基于强化学习的持续集成测试用例优先排序质量的影响如何?

问题 3. 基于相似性的奖励对象选择策略对基于强化学习的持续集成测试用例优先排序框架的时间开销影响如何?

问题 1 通过不同阈值下基于相似性奖励对象选择策略对测试用例优先排序效果的影响研究, 选择适合目标程序的最佳阈值. 问题 2 通过基于相似性的奖励对象选择策略与基于执行失效的奖励对象选择策略进行测试用例优先排序质量的对比, 验证基于相似性的奖励对象选择策略的有效性. 问题 3 是对不同奖励对象选择策略下的框架时间开销进行对比.

3.2 评价指标

本文为了比较不同奖励对象选择策略, 在基于强化学习的持续集成测试用例优先排序下对持续集成测试用例优先排序效果通过 NAPFD (normalized average percentage of faults detected), TTF (test to fail), Recall 和时间开销 4 个评价指标进行度量.

(1) NAPFD

平均历史检测率 (APFD, average percentage of faults detected)[27] 作为测试用例优先排序效果评价的有效方法, 在检测到序列中所有错误的假设下, 根据测试序列中失效位置的索引进行测试序列的评价. 然而, 基于强化学习的持续集成测试用例优先排序, 使用测试用例选择的快速反馈机制模拟, 使得只有占一半执行时间的测试用例被执行, 即存在失效的测试用例不被执行. 由于 APFD 只适用于可以检测到所有错误的测试用例序列, 即不存在测试用例选择. 因此, Qu 等人[28] 于 2007 年提出 APFD 的扩展 NAPFD, 根据实际检测错误率来进行失效分布的计算, 适用于存在测试用例选择的测试用例优先排序序列.

NAPFD 作为测试用例优先排序的评价标准[28], 其定义为:

$$NAPFD(TS_i)=p-\frac{\sum_{t\in TS_i} rank(t)}{\left|TS_i^{\text{fail}}\right|\times|TS_i|}+\frac{p}{2\times|TS_i|} \tag{4}$$

其中, $p=\dfrac{\left|TS_i^{\text{fail}}\right|}{\left|TS_i^{\text{total,fail}}\right|}$, $rank(t)$ 表示的是第 t 次失效测试在排序序列 TS_i 中的位置, $\left|TS_i^{\text{fail}}\right|$表示 TS_i 中失效测试用例的总个数, $|TS_i|$表示 TS_i 中测试用例的总个数, $\left|TS_i^{\text{total,fail}}\right|$表示 TS 中失效测试用例的总个数, TS_i 是 TS 中被选择执行的子集.

根据 NAPFD 的定义, 在持续集成测试中, 如果所有的错误均被检测到, 则 p=1, NAPFD 与 APFD 等同; 如果存在错误未被检测到, 则 p<1, 基于 NAPFD 的评价更符合实际情况.

(2) TTF

TTF 根据测试执行中首次执行失效的测试用例在测试序列中的位置进行评价. 测试用例优先排序的目标是优先执行失效测试用例. 失效位置越靠前, 测试用例优先排序效果越好. 因此, TTF 可以有效地评价测试用例优先排序的效果. TTF 值越小, 即执行失效测试用例在执行序列中位置越靠前, 则测试用例优先排序的效果越好.

(3) Recall

为了有效地评价持续集成测试用例优先排序的质量, 引入 Recall 来评价每一周期实际检测到错误的比例. Recall 根据测试执行序列中实际检测错误与集成周期内实际存在错误的比重进行度量. Recall 在测试选择的基础上, 计算占总执行时间一半的测试用例集检测到的错误占该周期实际总错误的比例. Recall 值越高, 则执行序列检测的错误越多, 相应的测试用例优先排序效果越好.

(4) 时间开销

时间开销作为测试用例优先排序技术的重要评价指标, 对算法的执行效率进行评价. 针对基于强化学习的持续集成测试用例优先排序技术, 其时间开销包括奖励计算的时间、智能体的学习时间、测试用例执行信息的更新时间和测试用例优先排序的时间等.

3.3 实验对象及设计

本文针对个 6 工业数据集进行相关实验, 其中 IOF/ROL、Paint Control 和 GSDTSR 为 Spieker 等人[3] 研究中所使用的数据集, Rails 为开源工业数据集 (http://

gihub.com/elbum/CI−Datasets.git), 其他 2 个数据集为研究者共享的持续集成测试日志 (https://travistorrent.testroots.org/buildlogs/20-12-2016/). 测试用例是唯一识别码, 在持续集成的不同周期中测试不同的组件, 并记录了相关的测试执行结果. 表 2 列出了 6 个工业数据集的持续集成测试的详细信息, 包括测试用例数、持续集成周期数、持续集成测试执行结果数、持续集成失效率和频率. 集成周期不仅给出了持续集成的周期数, 并标注了实际测试过程中失效周期数, 执行结果数强调的是在整个持续集成过程中测试执行总数, 失效率表示持续集成测试过程中测试失效占总测试执行的比例, 频率统计了测试用例在每一周期进行测试的概率.

表 2 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 (%)	频率
Paint Control	114	312 (257)	25 594	19.36	0.82
IOF/ROL	2 086	320 (271)	30 319	28.43	0.05
GSDTSR	5 555	336 (284)	1 260 617	0.25	0.68
Rails	2 010	3 263 (2 047)	781 273	0.12	0.12
Mybatis	303	988 (55)	815 598	0.15	2.72
Apache Drill	556	350 (26)	8 887	2.28	0.05

在基于强化学习的持续集成测试用例优先排序中, 奖励函数设计方法采用目前最优的基于历史信息的奖励函数[5], 即基于平均历史失效分布的奖励函数 APHF (average percentage of historical failure). 在实际工业数据集中, 针对庞大历史信息, 本文使用滑动窗进行临近历史信息的特征选取[29], 即通过提取临近历史信息的特征以实现奖励计算效率的提升. 本文基于持续集成测试的连续变化, 采用基于动态滑动窗的临近历史信息选择方法, 即滑动窗尺寸随着持续集成测试进程动态变化, 以实现自适应的奖励计算[30].

3.4 实验结果及分析

本小节分别针对 3 个研究问题进行实验结果的展示及分析.

3.4.1 测试用例相似性阈值选取分析

本小节针对问题 1 进行实验分析. 测试用例相似性通过测试用例之间的特征匹配以实现相似性度量. 在基于相似性的奖励对象选择策略中, 使用执行通过测试用例与执行失效测试用例进行相似性对比, 从测试历史执行信息序列和测试执行时间角度进行多维度特征匹配, 从而确定智能体是否对执行通过测试用例进行奖励. 由于工业数据集存在不同的失效率、测试频率等特征, 在具体数据集中相似性阈值的不同, 会导致不同的奖励对象数量, 从而产生不同的测试用例优先排序效果. 因此需要针对不同的数据集进行相似性阈值的选取分析.

首先对工业数据集进行相似性特征分析, 即基于相似性度量方法进行测试用例相似性指标值的分析, 图 4 统计了在 6 个数据集上进行测试用例相似性值分析的具体分布情况, 针对具体数据集, 纵坐标表示基于相似性计算获得的相似性值. 由于在不同数据集上测试用例的执行过程和结果不同, 导致相似性值的范围不同, 并且相似性值存在较多的异常数据, 因此为了更好地确定适用于不同数据集的相似性阈值, 本文首先依据相似性值的分布针对具体数据集选取相似性值分布的下四分位数、中位数和上四分位数作为基于测试用例相似性的奖励对象选择策略阈值, 其统计结果如表 3 所示.

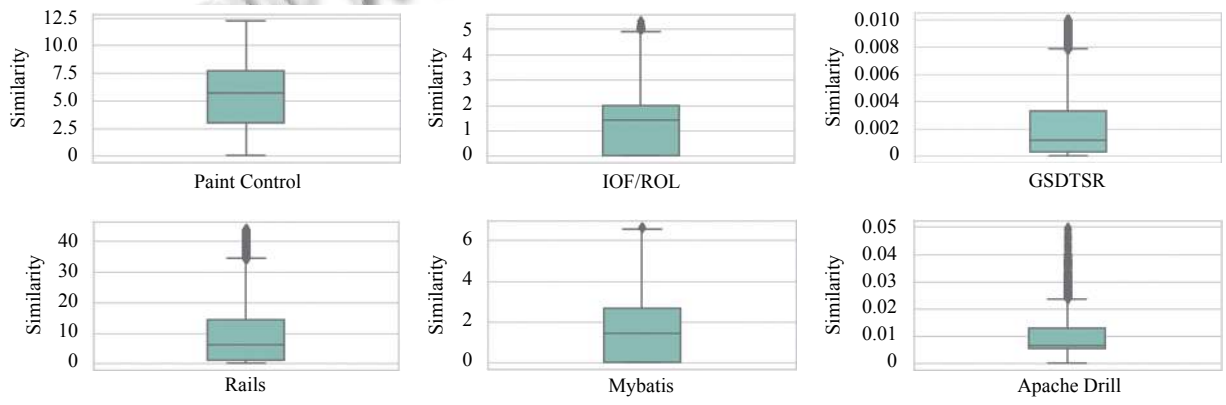

图 4 测试用例相似性数据分布图

进一步在基于强化学习的持续集成测试用例优先排序中依据基于相似性的奖励对象选择策略, 使用不同的相似性阈值进行基于强化学习的持续集成测试用例优先排序, 并使用 NAPFD 指标进行持续集成测试用例优先排序质量的评价, 并实现不同阈值的测试用例优先排序效果的对比, 其结果如表 4 所示, 其中加粗字体标注了在该阈值下可以取得最好的 NAPFD 值.

相似性阈值的选取, 决定了在基于强化学习的持续集成测试用例优先排序中增加奖励的执行通过测试用例的数量. 增加奖励的数量过多, 反而会降低失效影响大的执行通过测试用例的优先级; 而增加奖励的数量过少, 导致部分失效影响大的执行通过测试用例无法获得有效的奖励, 从而降低测试用例优先排序效果.

表 3 针对工业数据集的测试用例相似性阈值 ε 选取

ε	Paint Control	IOF/ROL	GSDTSR	Rails	MyBatis	Apache Drill
下四分位数	3.000 0	0.000 6	0.000 3	1.000 0	0.000 7	0.005 4
中位数	5.656 9	1.414 2	0.001 2	6.164 4	1.414 2	0.006 6
上四分位数	7.681 2	2.000 0	0.003 4	14.317 8	2.645 8	0.012 6

表 4 基于不同相似性阈值选取的持续集成测试用例优先排序 NAPFD 值 (%)

ε	Paint Control	IOF/ROL	GSDTSR	Rails	MyBatis	Apache Drill
下四分位数	68.39	**44.94**	14.08	**88.62**	31.12	37.81
中位数	70.87	44.05	23.31	85.32	31.11	38.24
上四分位数	**72.50**	42.60	**77.28**	88.85	**43.13**	**38.41**

因此针对 6 个数据集, 本文根据 NAPFD 的结果对比分别选取 Paint Control 的 7.681 2、IOF/ROL 的 0.000 6、GSDTSR 的 0.003 4、Rails 的 1.000 0、MyBatis 的 2.645 8 和 Apache Drill 的 0.012 6 作为基于相似性的奖励对象选择策略的相似性阈值, 以实现在有效的奖励数量增加下获得测试用例优先排序效果优化, 为后续实验及分析奠定基础.

3.4.2 基于相似性的奖励对象选择策略有效性分析

本小节针对问题 2 进行实验分析. 基于相似性的奖励对象选择策略, 在奖励失效测试用例的基础上, 通过相似性分析自适应地选择执行通过测试用例实施奖励, 从而实现奖励对象数量的增加以解决奖励稀疏问题. 在基于强化学习的持续集成测试用例优先排序中, 本文采用相同的奖励函数设计方法 APHF, 通过不同奖励对象选择策略, 即基于执行失效的奖励对象选择策略 (APHF_P) 和基于相似性的奖励对象选择策略 (APHF_S), 在 NAPFD、Recall 和 TTF 三个指标下进行测试用例优先排序质量的对比. 实验结果如表 5 所示, 其中加粗数据部分标注了在基于相似性的奖励对象选择策略下, 测试用例优先排序评价指标值优于基于执行失效的奖励对象选择策略的情况.

根据表 5 所示的实验结果, 从整体平均上来看, 基于相似性的奖励对象选择策略在不同的评价指标上均可以有效地进行指标值的提升, NAPFD 可以获得 5.94% 的提升, Recall 可以有效地改进 5.51% 的错误检测率, 并且在 TTF 上可以有效地提升 57.42 个失效测试用例在测试序列中首次执行失效的位置. 针对具体数据集的分析可以看出, 在 6 个数据集上, 基于相似性的奖励对象选择策略可以有效改进 4 个数据集的测试用例优先排序质量, 尤其在 GSDTSR 中可以明显提升 34.51% 的 NAPFD 值、34.13% 的 Recall 值和 293.15 位的 TTF.

表 5 基于不同奖励对象选择策略的持续集成测试用例优先排序质量

数据集	NAPFD (%)		Recall (%)		TTF	
	APHF_P	APHF_S	APHF_P	APHF_S	APHF_P	APHF_S
Paint Control	72.17	**72.50**	82.46	**83.05**	2.49	2.81
IOF/ROL	45.27	44.94	56.01	55.88	1.21	1.25
GSDTSR	42.77	**77.28**	48.01	**82.14**	296.69	**3.54**
Rails	88.37	**88.62**	92.13	**92.30**	4.6	**3.4**
Mybatis	38.85	**43.13**	90.03	**92.71**	356.73	**323.88**
Apache Drill	41.83	38.41	63.37	56.82	11.82	**9.63**
Average	54.88	**60.81**	72.00	**77.15**	112.26	**57.42**

在基于相似性的奖励对象选择策略下, 通过相似性阈值的判断来增加奖励执行通过测试用例. 相比于基于执行失效的奖励对象奖励策略, 基于相似性的奖励对象选择策略增加了强化学习的奖励数量, 缓解了面向持续集成测试用例优先排序的强化学习奖励对象稀少问题. 奖励对象的增加, 使得强化学习的智能体可

以接收到更多环境的反馈, 从而更好地进行测试用例优先排序策略的调整, 最终实现测试用例优先排序质量的提升.

综上所述, 基于相似性的奖励对象选择策略通过增加奖励与执行失效测试用例存在相似的执行通过测试用例, 有效地增加了面向持续集成测试用例优先排序的强化学习奖励对象, 解决了强化学习的奖励稀疏问题, 并进一步有效地提升了测试用例优先排序效果.

3.4.3 基于相似性的奖励对象选择策略的时间开销分析

本小节针对问题 3 进行实验分析. 基于相似性的奖励对象选择策略在基于强化学习的持续集成测试用例优先排序中, 通过相似性分析以实现测试用例优先排序质量的提升, 在一定程度上是以时间开销为代价进行测试用例优先排序质量的提升, 理论上增加了基于强化学习的持续集成测试优先排序框架的时间开销. 本文进一步在基于强化学习的持续集成测试用例优先排序中使用基于执行失效的奖励对象选择策略和基于相似性的奖励对象选择策略进行时间开销的对比分析, 其时间开销展示如表 6 所示.

表 6 基于不同奖励对象选择策略的持续集成测试用例优先排序时间开销 (s)

选择策略	Paint Control	IOF/ROL	GSDTSR	Rails	MyBatis	Apache Drill
APHF_P	0.213	0.070	2.142	0.100	0.386	0.048
APHF_S	0.202	0.259	4.916	0.285	1.567	0.097

基于强化学习的持续集成测试用例优先排序框架的时间开销, 不仅包括了奖励计算的时间、智能体的学习时间和测试用例优先排序的时间, 还考虑了奖励对象选择的时间, 以及相似性分析的时间. 根据表 6 所示的时间开销结果, 明显看出基于相似性的奖励对象选择策略在大部分数据集上均明显增加了时间开销, 这是由于在每一周期进行相似性分析增加了相应的时间开销, 然而基于相似性的奖励对象选择策略通过解决强化学习的奖励稀疏问题, 加速了强化学习的学习速度, 因此对于框架整体来说, 其平均时间开销的增加均是秒级, 是可以接受的时间开销增加.

综上所述, 基于相似性的奖励对象选择策略在有限的时间开销增加下, 不仅通过奖励对象的增加有效地解决了基于强化学习的持续集成测试用例优先排序奖励稀疏问题, 还有效了提升了持续集成测试用例优先排序质量.

4 结论与展望

本文针对工业程序的持续集成测试存在频繁迭代和测试失效低导致奖励稀疏的问题, 在基于强化学习的持续集成测试用例优先排序中, 依据测试用例的历史执行过程和执行时间提出了基于相似性的奖励对象选择策略, 在奖励失效测试用例的基础上, 通过选择与失效测试用例存在相似的通过测试用例作为额外奖励对象, 有效解决了奖励稀疏问题. 提出的方法在有限时间开销增长下, 有效地提升了测试用例优先排序效果.

未来进一步研究构建基于强化学习的持续集成测试用例优先排序策略网络. 通过自适应学习测试用例的新特征以实现测试用例优先排序策略的调整, 从而生成适合后续的测试用例优先排序策略.

参考文献

1 Zhao YY, Serebrenik A, Zhou YM, *et al*. The impact of continuous integration on other software development practices: A large-scale empirical study. 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE). Urbana: IEEE, 2017. 60–71.

2 Pinto G, Castor F, Bonifacio R, *et al*. Work practices and challenges in continuous integration: A survey with Travis CI users. Software: Practice and Experience, 2018, 48(12): 2223–2236. [doi: 10.1002/spe.2637]

3 Spieker H, Gotlieb A, Marijan D, *et al*. Reinforcement learning for automatic test case prioritization and selection in continuous integration. Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis. Santa Barbara: ACM, 2017. 12–22.

4 Sutton RS, Barto AG. Reinforcement Learning: An Introduction. 2nd ed. Cambridge: MIT Press, 2018. 1–25.

5 Yang Y, Li Z, He LL, *et al*. A systematic study of reward for reinforcement learning based continuous integration testing. Journal of Systems and Software, 2020, 170: 110787. [doi: 10.1016/j.jss.2020.110787]

6 杨惟轶, 白辰甲, 蔡超, 等. 深度强化学习中稀疏奖励问题研究综述. 计算机科学, 2020, 47(3): 182–191. [doi: 10.11896/jsjkx.190200352]

7 Cave P. The error of excessive proximity preference —A modest proposal for understanding holism. Nursing Philosophy, 2000, 1(1): 20–25. [doi: 10.1046/j.1466-769x.2000.00003.x]

8 Huang RB, Zhou YN, Zong WW, *et al*. An empirical comparison of similarity measures for abstract test case

prioritization. 2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC). Turin: IEEE, 2017. 3–12.

9 Bagherzadeh M, Kahani N, Briand L. Reinforcement learning for test case prioritization. IEEE Transactions on Software Engineering, 2021.

10 Chen TY, Kuo FC, Merkel RG, *et al*. Adaptive random testing: The art of test case diversity. Journal of Systems and Software, 2010, 83(1): 60–66. [doi: 10.1016/j.jss.2009.02.022]

11 Jiang B, Zhang ZY, Chan WK, *et al*. Adaptive random test case prioritization. 2009 IEEE/ACM International Conference on Automated Software Engineering. Auckland: IEEE, 2009. 233–244.

12 Cartaxo EG, Machado PDL, Neto FGO. On the use of a similarity function for test case selection in the context of model-based testing. Software Testing, Verification and Reliability, 2011, 21(2): 75–100. [doi: 10.1002/stvr.413]

13 Hemmati H, Arcuri A, Briand L. Empirical investigation of the effects of test suite properties on similarity-based test case selection. 2011 Fourth IEEE International Conference on Software Testing, Verification and Validation. Berlin: IEEE, 2011. 327–336.

14 Ledru Y, Petrenko A, Boroday S, *et al*. Prioritizing test cases with string distances. Automated Software Engineering, 2012, 19(1): 65–95. [doi: 10.1007/s10515-011-0093-0]

15 Henard C, Papadakis M, Harman M, *et al*. Comparing white-box and black-box test prioritization. 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE). Austin: IEEE, 2016. 523–534.

16 Thomas SW, Hemmati H, Hassan AE, *et al*. Static test case prioritization using topic models. Empirical Software Engineering, 2014, 19(1): 182–212. [doi: 10.1007/s10664-012-9219-7]

17 Noor TB, Hemmati H. A similarity-based approach for test case prioritization using historical failure data. 2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE). Gaithersbury: IEEE, 2015. 58–68.

18 Deza MM, Deza E. Distances and similarities in data analysis. Encyclopedia of distances. Berlin Heidelberg: Springer, 2013. 291–305.

19 Ricotta C, Pavoine S, Wildi O. Measuring similarity among plots including similarity among species: An extension of traditional approaches. Journal of Vegetation Science, 2015, 26(6): 1061–1067.

20 Chen SY, Chen ZY, Zhao ZH, *et al*. Using semi-supervised clustering to improve regression test selection techniques. 2011 Fourth IEEE International Conference on Software Testing, Verification and Validation. Berlin: IEEE, 2011. 1–10.

21 Dickinson W, Leon D, Fodgurski A. Finding failures by cluster analysis of execution profiles. Proceedings of the 23rd International Conference on Software Engineering. ICSE 2001. Toronto: IEEE, 2001. 339–348.

22 Zhang C, Chen ZY, Zhao ZH, *et al*. An improved regression test selection technique by clustering execution profiles. 2010 10th International Conference on Quality Software. Zhangjiajie: IEEE, 2010. 171–179.

23 Xu R, Wunsch D. Survey of clustering algorithms. IEEE Transactions on Neural Networks, 2005, 16(3): 645–678. [doi: 10.1109/TNN.2005.845141]

24 Fang CR, Chen ZY, Wu K, *et al*. Similarity-based test case prioritization using ordered sequences of program entities. Software Quality Journal, 2014, 22(2): 335–361. [doi: 10.1007/s11219-013-9224-0]

25 Masri W, Podgurski A, Leon D. An empirical study of test case filtering techniques based on exercising information flows. IEEE Transactions on Software Engineering, 2007, 33(7): 454–477. [doi: 10.1109/TSE.2007.1020]

26 Wang RC, Jiang SJ, Chen D. Similarity-based regression test case prioritization. SEKE. 2015: 358–363.

27 Rothermel G, Untch RH, Chu CY, *et al*. Prioritizing test cases for regression testing. IEEE Transactions on Software Engineering, 2001, 27(10): 929–948. [doi: 10.1109/32.962562]

28 Qu X, Cohen MB, Woolf KM. Combinatorial interaction regression testing: A study of test case generation and prioritization. 2007 IEEE International Conference on Software Maintenance. Paris: IEEE, 2007. 255–264.

29 Wu ZL, Yang Y, Li Z, *et al*. A time window based reinforcement learning reward for test case prioritization in continuous integration. Proceedings of the 11th Asia-Pacific Symposium on Internetware. Fukuoka: ACM, 2019. 4.

30 Yang Y, Pan CY, Li Z, *et al*. Adaptive reward computation in reinforcement learning-based continuous integration testing. IEEE Access, 2021, 9: 36674–36688. [doi: 10.1109/ACCESS.2021.3063232]