

基于代码变更与历史信息的测试用例优先级排序

李晋英¹, 刘峰¹, 罗远哲², 刘瑞景²

(1. 北京交通大学计算机科学与技术学院, 北京 100044; 2. 北京中超伟业信息安全技术股份有限公司, 北京 100193)

摘要: 现有测试用例优先级排序方法常依赖单一的数据源或分析技术, 无法全面捕捉影响测试用例重要性的潜在因素, 限制了排序效果。为此, 提出一种基于代码变更与历史信息的测试用例优先级排序框架SSH-TCP。首先, 在粗粒度过滤阶段通过信息检索模型得到候选测试用例集; 其次, 在细粒度排序阶段利用预训练语言模型、基于抽象语法树的中心性分析, 分别计算代码变更和测试用例的语义与语法相似度; 最后, 结合测试用例的平均历史失败率确定最终的排序结果。实验表明, SSH-TCP在考虑成本的平均故障检测率指标上优于所有基线模型, 验证了所提框架在提升测试用例优先级排序效果方面的有效性。

关键词: 测试用例排序; 代码变更; 语义相似度; 语法相似度; 历史失败率

DOI: 10.11907/rjdk.241811

中图分类号: TP18

文献标识码: A

文章编号: 1672-7800(2026)001-0075-08

扫描二维码阅读全文:



Test Case Prioritization Based on Code Changes and Historical Information

LI Jinying¹, LIU Feng¹, LUO Yuanzhe², LIU Ruijing²

(1. School of Computer Information and Technology, Beijing Jiaotong University, Beijing 100044, China;
2. Beijing SinoGuard Security Technology Co., Ltd, Beijing 100193, China)

Abstract: The existing methods for prioritizing test cases often rely on a single data source or analysis technique, which cannot fully capture the potential factors that affect the importance of test cases and limits the effectiveness of the ranking. To this end, a test case prioritization framework SSH-TCP based on code changes and historical information is proposed. Firstly, in the coarse-grained filtering stage, a candidate test case set is obtained through an information retrieval model; Secondly, in the fine-grained sorting stage, pre trained language models and centrality analysis based on abstract syntax trees are used to calculate the semantic and syntactic similarity of code changes and test cases, respectively; Finally, the final ranking result is determined based on the average historical failure rate of the test cases. The experiment shows that SSH-TCP outperforms all baseline models in terms of average fault detection rate considering cost, verifying the effectiveness of the proposed framework in improving the prioritization of test cases.

Key Words: test case prioritization; code changes; semantic similarity; syntactic similarity; historical failure rate

0 引言

回归测试作为确保软件质量的关键环节之一, 其作用是确保修改代码部分无误, 保证代码的更改不会对现有功能产生不良影响。然而, 由于回归测试用例数量庞大且软件版本更新频繁, 回归测试的成本极高。Yang等^[1,2]指出,

超过80%的测试时间被用于回归测试, 测试成本占据软件累计费用33%以上。

目前, 越来越多的开发团队采用持续集成(Continuous Integration, CI)策略, 以应对代码更改后的软件快速发布需求。具体为, 每次发布新版本前迅速完成大量回归测试, 对提升测试效率提出了严峻挑战。在回归测试领域, 基于覆盖的测试用例优先级排序(Test Case Prioritization,

收稿日期: 2024-09-18

基金项目: 铁路轨道在轨安全检测大数据系统开发与验证项目(ZCWY-NB-CG-20231116-220); 云测试软件系统开发与验证项目(ZCWY-NB-CG-20240730-145)

作者简介: 李晋英(2000-), 女, 北京交通大学计算机科学与技术学院硕士研究生, 研究方向为软件测试、测试用例优化; 刘峰(1961-), 男, 博士, 北京交通大学计算机科学与技术学院教授、硕士生导师, 研究方向为软件测试、轨道交通信息技术、高可信软件; 罗远哲(1974-), 男, 博士, 北京中超伟业信息安全技术股份有限公司教授级高级工程师, 研究方向为信息安全技术; 刘瑞景(1970-), 女, 硕士, 北京中超伟业信息安全技术股份有限公司高级工程师, 研究方向为电子信息、信息安全技术。

TCP)方法被广泛研究,但这些方法依赖于先前版本的覆盖率信息,无法有效应对软件迭代中的代码变更。然而,基于信息检索(Information Retrieval, IR)的TCP方法能直接通过代码变更来定位相关测试用例,相较于基于覆盖的TCP方法不依赖详尽的覆盖率信息,直接分析代码变更与测试用例之间的关联性,从而更敏捷地适应快速迭代的开发环境。

目前,大多数基于IR的TCP方法仅限于词法层面的分析,难以捕捉代码的深层语义和语法特征。更为关键的是,这些方法往往忽视了测试用例的执行历史,无法充分利用历史数据来预测未来的测试效果。为此,本文提出SSH-TCP框架(Semantic and Structure-aware History-based Test Case Prioritization),综合考虑代码变更与测试用例之间的语义与语法相似度、测试用例的历史执行信息,以更全面、高效地排序测试用例,提升回归测试的效率和模型性能。

1 相关工作

为了有效应对回归测试中的挑战,研究人员已开发了多种测试优化方法。Yoo等^[3]将这些方法分为测试用例集最小化(Test Suite Minimization, TSM)、测试用例选择(Test Case Selection, TCS)和测试用例优先级排序(Test Case Prioritization, TCP)。其中,TSM方法是一种减少测试用例数量的方法,目的是消除冗余或不必要的测试用例;TCS方法从整个测试用例集中挑选出与最近代码更改最相关的测试用例;TCP方法相较于前面两种方法是一种对测试用例进行排序的方法,目的是让那些最有可能发现错误的测试用例优先执行,以更早地向开发团队提供潜在问题的反馈,加快缺陷修复进程。

综上,前两种方法可能会遗漏关键的测试用例,最后一种方法不仅提升了测试可靠性,还提升了测试过程的整体效率。

Abdelkarim等^[4]提出TCP-Net结合了测试用例的覆盖率特征与基于源文件的特征,利用深度融合架构对测试用例进行优先级排序。在众多TCP方法中基于覆盖的TCP方法被广泛研究,Rothermel等^[5]将基于覆盖的TCP方法分为总体覆盖策略与增量覆盖策略。其中,总体覆盖策略根据测试用例覆盖的代码元素(语句、分支、函数等)总数量对测试用例进行排序,覆盖元素越多的测试用例优先级越高;增量覆盖策略侧重于测试用例所能覆盖的独特代码元素,即那些其他测试用例尚未覆盖的部分,该方式可确保每次测试都能最大程度地揭示新信息。Mondal等^[6]提出一种基于代码覆盖的TCP方法Colosseum,通过计算测试用例执行路径上变更代码元素的位移参数,并优先执行总位移较小的测试用例。

然而,基于覆盖的TCP方法存在一些限制。首先,该

方法的时间开销较大,Yaraghi等^[7]发现与覆盖率相关的特征收集成本最高,但对TCP方法的有效性影响最小;其次,Zhao等^[8]指出这种方法通常基于先前软件版本的覆盖率信息,对最新版本的测试用例进行排序,未充分考虑旧版本信息的过时性,尤其是在版本间的代码变更较大时,这种做法的误差可能会显著影响排序效果。

Saha等^[9]、Azizi等^[10]研究发现,代码变更在软件回归中大概率会引入故障,与代码更改相关的测试用例更有可能检测到故障。基于信息检索IR的TCP方法的研究逐渐增多,将代码变更视为查询、测试用例集视为文档集合,通过计算两者之间的相似度来确定测试用例优先级,从而检测最近代码更改可能引入的错误。相较于传统的基于覆盖的TCP方法,基于IR的TCP方法的优势在于能直接定位与最新代码变更相关的测试用例,无需依赖收集的复杂覆盖信息,提升了测试效率和适应性,使得基于IR的TCP方法更适合快速迭代的软件开发环境,能有效降低成本、提升响应速度。随后,Peng等^[11]提出的OptIR方法,为代码变更与相关测试用例构建token序列,再使用IR模型评估两者之间的相似度。实验证明,OptIR在效果上超越了基于覆盖率的方法,可提供更快的反馈、更高的故障检测效率。

然而,当前基于IR的TCP方法主要依赖传统算法,例如BM25或TF-IDF,主要在词法层面上衡量代码变更与测试用例之间的相似性,在处理大量数据时的效率较高,但只能捕捉词汇的表面匹配,忽略了代码中的深层语义和语法信息。尽管有一些研究通过更复杂的模型来捕获代码的深层语义,但忽视其他维度信息,例如代码语法特征,这些特征对全面理解代码功能至关重要。例如,Li等^[12]提出一种名为SatTCP的语义感知两阶段TCP框架建模代码语义,但未考虑代码的语法特征。同时,基于IR的TCP方法也未充分利用测试用例的历史执行信息,例如历史失效记录、执行频率等。Marijan等^[13]、Siqueira等^[14]指出这些数据对准确评估测试用例的重要性和未来故障检测的能力至关重要,在复杂或频繁变更的项目中忽略多维度信息可能会影响测试用例优先级的评估准确性,从而影响整体测试效率和效果。

为了综合考虑代码变更与测试用例之间相似度、测试用例的历史执行信息,提出基于代码变更与历史信息的测试用例优先级排序框架SSH-TCP。首先,在粗粒度过滤阶段利用信息检索模型筛选与代码变更相关的测试用例;其次,在细粒度排序阶段使用预训练语言模型分析测试用例与代码变更的语义关联;再次,将代码变更与测试用例的抽象语法树转换为树图,并进行中心性分析以深化理解代码结构;最后,结合测试用例的平均历史失败率,通过倒秩融合算法完成最终排序。

综上,SSH-TCP不仅关注代码与测试用例的静态代码分析,还充分利用了动态历史数据,为测试用例排序提供

了一个全面的评估视角。

架可分为粗粒度过滤和细粒度排序这, 如图 1 所示。

2 本文方法

2.1 整体框架

基于代码变更和历史信息的测试用例优先级排序框

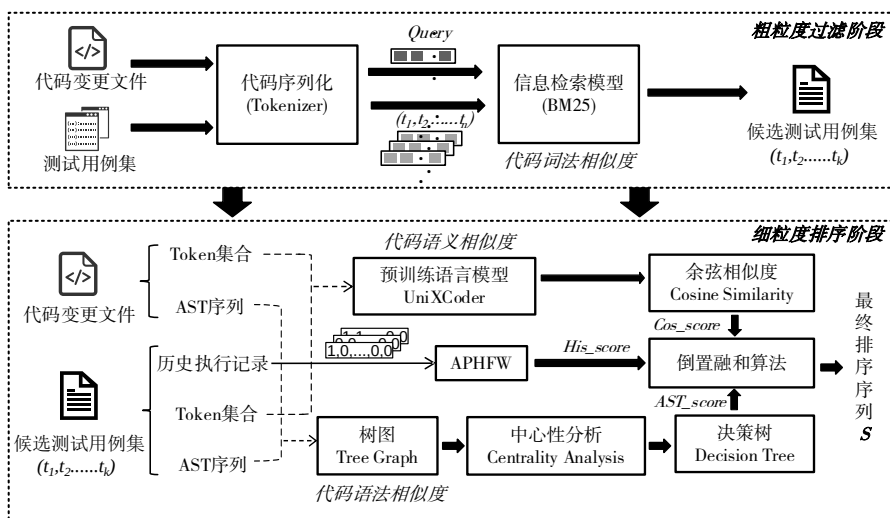


Fig. 1 Test case prioritization framework based on code changes and historical information

图 1 基于代码变更和历史信息的测试用例优先级排序框架

(1)粗粒度过滤阶段。代码变更与测试用例的代码序列化处理后,通过信息检索模型 BM25 进行初步排序以筛选出候选测试用例集,旨在初步确定相关的测试用例集合,缩减后续模型训练规模。

(2)细粒度排序阶段。首先,采用预训练语言模型对前一阶段生成的 token 序列进行深入地语义分析,使用余弦相似度计算得到余弦相似度得分 Cos_Score ;其次,将提取代码变更与测试用例的抽象语法树 (Abstract Syntax Tree, AST) 转换为树图,并视为社交网络进行中心性分析;再次,将复杂的树结构转换为 83 维的向量,经过决策树模型处理计算得出 AST 的得分 AST_Score ,并将测试用例的平均历史失败率作为其历史得分 His_Score ;最后,通过倒秩融合算法综合余弦相似度得分、AST 得分和历史得分,形成最终的测试用例优先级排序结果。

2.2 粗粒度过滤阶段

2.2.1 代码序列化阶段

代码序列化阶段的目的是从测试用例和代码变更中提取有意义的 token 序列,为后续筛选、排序阶段提供必要的输入。首先,使用 Javalang 库从 Java 源代码中提取标识符^[15];其次,使用 Word Ninja 英文分词工具将长标识符进行分割,以获得更细粒度的词汇 (例如,将“databaseName”分割为“database”“name”),并将细粒度词汇转换为小写便于后续评估;最后,删除 Java 关键词与其他保留字 (“public”“class”),过滤掉停用词和短词 (“the”“a”等),因为这些词汇通常不携带有关功能实现的具体信息。

2.2.2 BM25 排序阶段

代码序列化后提取的 token 序列被用于 BM25 排序算法,以评估测试用例与代码变更之间的相关性。BM25 是一种基于概率的排序算法,用于估计文档与用户查询之间的相关性^[16]。在测试用例优先级排序的背景下,该算法将代码变更视为查询,将每个测试用例视为包含代码变更 token 的“文档”。

BM25 算法从词汇的相似性角度出发,通过衡量查询 token 在各个测试用例“文档”中的出现频率和分布,计算每个测试用例与查询的相关性得分。具体公式为:

$$score(D, Q) = \sum_{i=1}^n IDF(q_i) \cdot \frac{f(q_i, D) \cdot (k+1)}{f(q_i, D) + k \left(1 - b + b \cdot \frac{|D|}{avgdl}\right)} \quad (1)$$

$$IDF(q_i) = \ln \left(\frac{N - n(q_i) + 0.5}{n(q_i) + 0.5} \right) \quad (2)$$

式中: D 表示文档 (测试用例); Q 表示查询 (代码变更); $IDF(q_i)$ 为查询词 q_i 的逆文档频率; $f(q_i, D)$ 为查询词 q_i 在文档 D 中的出现频率; $|D|$ 为文档 D 的长度 (即文档中的词汇总数); $avgdl$ 为所有文档的平均长度; N 为文档总数; $n(q_i)$ 为包含词 q_i 的文档数量; k 、 b 为可调参数,通常设定为 $k=1.5$ 、 $b=0.75$ 。

实验表明,通过 BM25 算法能有效捕捉代码变更和测试用例在词法层面的匹配度,得分较高的测试用例被认为与代码变更具有更高的相关性。为了保证模型的适应性, Li 等^[12]选择得分最高的前 50% 测试用例和得分最低的后

5% 测试用例作为细粒度过滤阶段的训练集,以降低后续模型训练的时间开销,显著减少因训练不相关测试用例而导致的时间和资源浪费,确保资源集中在最有可能揭示问题的测试用例上。

2.3 细粒度排序阶段

2.3.1 基于 token 的语义相似度检测阶段

本文在基于 token 的语义相似度检测阶段,采用预训练语言模型 UniXCoder 分析代码变更与测试用例之间的深层语义关系^[17]。UniXCoder 是一种专门处理编程语言设计的预训练语言模型,在大规模代码数据集上进行预训练,具备强大的泛化能力,在代码克隆检测、缺陷检测、代码生成等任务中表现出色。首先,从上一阶段筛选出的候选数据集中,提取代码变更的 token 序列及相应的测试用例 token 序列;其次,将两者分别输入 UniXCoder 模型以提取该

模型第一层与最后一层的隐藏状态的平均向量,即得到测试用例、代码变更的代码语义特征向量 $\vec{x}, \vec{y}$;最后,向量之间的相似度通过余弦相似度计算得到余弦相似度得分 Cos_Score 。

$$Cos_Score = \frac{\vec{x} \cdot \vec{y}}{\|\vec{x}\| \|\vec{y}\|} = \frac{\sum_{i=1}^n (x_i \times y_i)}{\sqrt{\sum_{i=1}^n x_i^2} \times \sqrt{\sum_{i=1}^n y_i^2}} \quad (3)$$

2.3.2 基于 AST 的语法相似度检测阶段

通过 Javalang 库分别从代码变更和测试用例的代码中提取 AST,根据节点类型将 AST 转换为简单的图表示(树图),如图 2 所示^[18]。其中,树图中最多包含 68 种类型节点(图 2 中通过单层矩形表示)和 15 种标记节点(图 2 中通过双层矩形表示),父子节点关系转化为图中的边,边的权重基于出现频率。

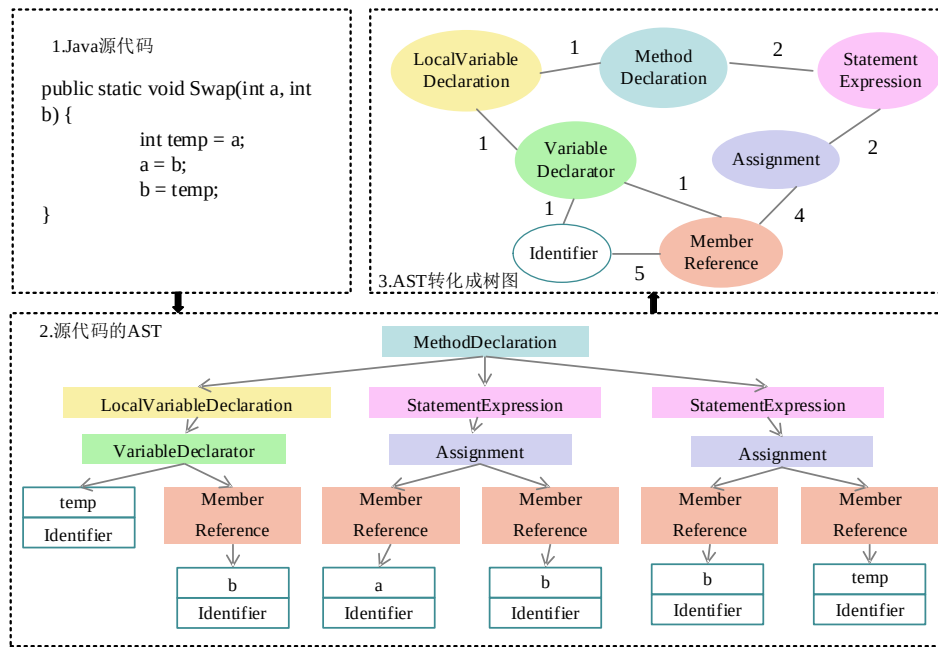


Fig. 2 Example of converting AST to tree graph

图 2 AST 转换为树图示例

本文将树图视为社交网络,对每个节点进行度中心性分析以保留树的细节。对于无向图,度中心性根据每个节点的邻接节点数来分配重要性分数,通过除以图中最大可能度数 $N-1$ 来标准化。度中心性的计算公式如式(4)所示。

$$C_d(v) = \frac{\deg(v)}{N-1} \quad (4)$$

式中: N 表示图中的节点数; $\deg(v)$ 为节点 v 的度数。

本文通过中心性分析将复杂的树结构转换为 83 维的中心性向量,拼接代码变更与测试用例的中心性向量,每个测试用例的 token 序列被分配一个二元标签,通过决策树模型来输出表示代码结构相似度的分数 AST_Score 。其中,标签 1 表示该测试用例在上一版本中检测到的故障,并显示与代码变更的高度语义相似性;标签 0 表示测试用

例在上一周期执行成功,与代码变更无直接相关性。

2.3.3 历史信息处理机制

考虑到使用全部历史执行信息可能会引入冗余和过时信息,本文采用基于时序滑动窗口的平均历史执行失败率(Average Historical Execution Failure Rate, APHFW)计算测试用例的历史得分 His_Score ^[19]。APHFW 通过时序滑动窗口筛选测试用例最近的历史执行数据,以评估 TCP 的历史维度。APHFW 的具体计算公式为:

$$APHFW = 1 - \frac{\sum_{j=1}^m R_j}{\text{win}(n, W) \times m} + \frac{1}{\text{win}(n, W) \times 2} \quad (5)$$

式中: $\text{win}(n, W)$ 表示窗口大小,受文献[20]的启发将窗口大小设定为 16,代表仅考虑最近 16 次的执行历史; m 表示测试用例在 $\text{win}(n, W)$ 次执行中的失效总次数; R_j 表

示测试用例最近第 j 次失效在历史执行的次序; $APHFW \in (0, 1)$, 值越高表示测试用例在临近的历史执行中失效概率越大, 越可能在下一次执行中检测出错误。

2.3.4 基于倒秩融合算法合并排序结果

将余弦相似度得分 Cos_Score 、AST 得分 AST_Score 、历史得分 His_Score 分别转换为相应的排名, 利用倒秩融合算法 (Reciprocal Rank Fusion, RRF) 对其进行融合, 以生成最终的排序结果。其中, RRF 根据每种排名的可靠性和权重进行精细调整和优化, 以避免直接进行得分加权可能导致的权重分配不均匀问题。

$$Score(q) = \frac{1}{R_{Cos}(q) + M} + \frac{1}{R_{AST}(q) + M} + \frac{1}{R_{His}(q) + M} \quad (6)$$

式中: M 为平滑因子; $R_{Cos}(q)$ 代表测试用例在余弦相似度得分中的排名; $R_{AST}(q)$ 代表在语法相似度得分中的排名; $R_{His}(q)$ 则代表在历史平均失败率中的排名。

随后, 基于倒秩融合算法计算出的分数 $Score(q)$ 对测试用例集进行二次精确排序, 以进一步提升测试用例优先级排序效果。

3 实验结果与分析

3.1 实验对象与评估指标

本文利用文献[11]的数据集进行实验。该数据集源自 Github 上公开托管的项目, 包含 123 个项目, 2 980 个 Travis 构建作业。为了确保数据的可靠性与分析准确性, 剔除了构建次数和测试用例执行次数较少的项目, 最终选出 44 个项目。由于项目数量众多, 文中仅展示 5 个具有代表性的项目, 如表 1 所示。

Table 1 Experimental subject information

表 1 实验对象信息

项目名称	代码构建次数	测试用例数	测试用例执行次数	失败率/%
gchq-Gaffer	41	534	2 078	9.14
codelibs-fess	90	45	3 257	6.60
ebean-orm-ebean	44	1 441	30 432	1.63
apache-incubator-druid	93	925	30 332	0.50
resteasy-Resteasy	101	743	54 365	0.35

数据集为每个项目提供了两部分内容: ①代码更改文件的内容及文件名; ②执行的测试用例集, 包括每个测试用例的代码内容、状态、名称和执行时间。本文采用平均故障检测率 APFD 和考虑测试成本的平均故障检测率 APFDc 作为评估指标^[19,21]。其中, APFD 通过排序后的测试用例执行顺序来衡量模型检测到故障的速度; APFD 的范围为 0~1, 值越接近 1 表示测试用例排序效果越好, 意味着测试用例集能更早地检测到更多故障。具体计算公式为:

$$APFD = 1 - \frac{\sum_{i=1}^m TF_i}{n \times m} + \frac{1}{2n} \quad (7)$$

式中: n 为测试用例的总数; m 为故障总数; TF_i 为第 i 个故障首次被检测到的测试用例在排序中的位置; APFDc 为 APFD 的扩展版本, 不仅考虑了测试用例检测到故障的顺序, 还考虑了每个测试用例的执行成本 (通常为执行时间)。APFDc 的计算公式为:

$$APFD_c = \frac{\sum_{i=1}^m \left(\sum_{j=TF_i}^n t_j - \frac{1}{2} t_{TF_i} \right)}{\sum_{j=1}^n t_j \times m} \quad (8)$$

式中: t_j 为第 j 个测试用例的执行时间。

同时, 为了研究所提方法的时间效率, 使用平均时间减少百分比 APTR 来评估 TCP 方法相较于实际持续集成测试节省的时间, 正值的 APTR 表示 TCP 方法相较于传统的 CI 测试节省了时间, 提升了测试过程的效率^[12]; 反之则表明 TCP 方法增加了时间成本, 需要重新评估和调整 TCP 方法。具体计算公式为:

$$APTR = \frac{\sum_{i=1}^n t_i - \left(t_{TCP} + \sum_{j=1}^{TF_c} t_j \right)}{\sum_{i=1}^n t_i} \quad (9)$$

式中: t_{TCP} 表示排序所需时间; TF_c 指测试用例序列中最后一个故障的出现次序; $\sum_{j=1}^{TF_c} t_j$ 为测试用例集检测到所有故障所需的时间; $\sum_{i=1}^n t_i$ 为实际持续集成测试所需的时间。

3.2 基线模型

为了验证本文方案的有效性, 选择 SatTCP、OptIR、QTF 及 HIS 作为基线模型进行比较分析^[11,12,22]。SatTCP 是一种语义感知的两阶段 TCP 框架。首先, 解析代码变更和测试用例以获取两者的代码表示; 其次, 利用 BM25 算法对测试用例进行初步排序; 最后, 通过预训练语言模型进行精确的语义相似度排序。

OptIR 采用信息检索模型计算代码更改和每个测试用例之间的文本相似度来确定测试用例优先级, 结果显示 BM25 算法在 TCP 任务中表现出色, 结合历史失败频率与测试执行时间后的混合 OptIR 方法故障检测率更高。HIS 基于历史失败频率高的测试用例在下一周期失败可能性更大的假设, 通过收集测试用例在之前周期中的失败次数除以测试用例总的执行次数来计算历史失败频率, 并按降序排序。QTF 基于单位时间内执行更多测试用例能发现更多错误的假设, 通过测试用例的执行时间来确定优先级, 执行时间越长的测试用例优先级越低。

3.3 实验设计

本文将每个项目的数据按 8:1:1 的比例划分训练集、验证集和测试集, 保证训练集、验证集和测试集中都包含

各个项目的样本,以确保模型能学习到各项目的特征,以提升评估可靠性。

(1)基于 token 的语义相似度检测阶段。为了提升模型的准确性和可靠性,使用专门为编程语言设计的预训练模型 UniXCoder 预训练语言模型生成代码语义向量。该模型基于先进的 Transformers 架构,特别适用于编程相关的自然语言处理任务,例如代码克隆检测、代码生成、代码摘要和代码搜索等。

为了提升模型在特定数据集上的适应性和性能,在预训练的 UniXCoder 模型上进行微调,将 warm-up 步骤设置为 100,使用 AdamW 优化器,学习率设置为 $2e-6$,模型训练 20 个 epoch,每批次大小为 68,使用余弦相似度作为损失函数,每 2 000 个训练步骤评估一次 SSH-TCP 在验证集上的性能。

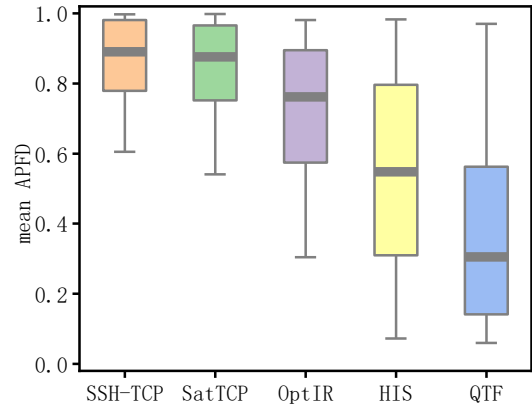
(2)基于 token 的语法相似度检测阶段。首先,采用 Javalang 库解析 Java 源代码,生成代码变更和测试用例的抽象语法树;其次,使用 networkx 库构建 AST 图结构,从中提取中心性向量^[23];最后,利用 sklearn 库实现决策树模型,以分析、预测代码变更和测试用例的语法相似性^[24]。

3.3 实验结果与分析

3.3.1 性能比较

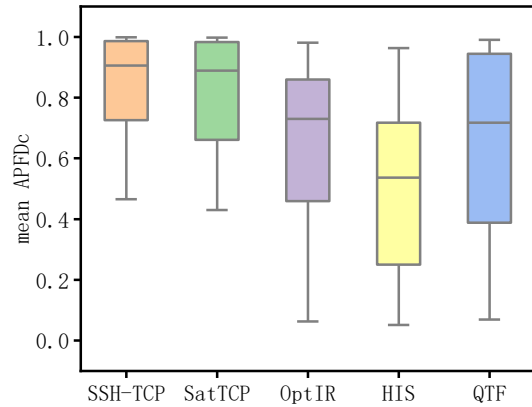
图 3 展示了 SSH-TCP 与多个基线模型在平均 APFD、平均 $APFD_c$ 两个关键性能指标上的结果。各方法的 APFD、 $APFD_c$ 及差异性如表 2 可见。由此可知,SSH-TCP 在这些指标上表现显著,SSH-TCP 的平均 APFD 为 0.871,平均 $APFD_c$ 为 0.847,在所有基线模型中均为最高,平均 $APFD_c$ 相较于其他基线模型高 3.55%~59.8%,凸显了 SSH-TCP 在测试用例排序效果上的优势,表明 SSH-TCP 通过融合先进的信息检索技术、深度学习及历史数据分析,可有效提升测试用例排序的效果。

本文为了从统计学角度验证 SSH-TCP 的优越性,采用 Wilcoxon 符号秩检验来评估 SSH-TCP 与其他基线模型之间的性能差异^[25]。在此检验中,零假设为 SSH-TCP 与其他基线模型之间在性能上没有显著差异。当 $p < 0.05$ 时,表明有足够的证据拒绝零假设,即观测结果与零假设之间存在显著差异。由表 2 可知,所有 p 值均低于阈值 0.05,表



(a) Mean APFD of each model

(a) 各模型的 mean APFD



(b) Mean APFDc of each model

(b) 各模型的 mean APFDc

Fig. 3 Box line of SSH-TCP and baseline model

图 3 SSH-TCP 与基线模型的箱线

明 SSH-TCP 在 APFD、 $APFD_c$ 两个关键性能指标上与其他基线模型存在统计学上的显著差异,进一步证实了 SSH-TCP 框架的有效性。

3.3.3 消融实验

通过消融实验以探究语义分析、语法分析和历史信息在细粒度排序阶段对 TCP 的具体贡献,具体研究了 4 种变体:①仅依赖语义相似度分析的 Sem-TCP;②仅依赖语法相似度分析的 Syn-TCP;③结合语义与语法分析的 Sem-

Table 2 APFD, $APFD_c$, and differential detection results of each method

表 2 各方法的 APFD、 $APFD_c$ 及差异性检测结果

方法	APFD-mean	APFD-median	APFD-p 值	$APFD_c$ -mean	$APFD_c$ -median	$APFD_c$ -p 值	APTR/%
SSH-TCP	0.871	0.890	-	0.847	0.905	-	68.3
SatTCP	0.855	0.876	0.038	0.818	0.889	0.026	67.9
OptIR	0.735	0.762	0.032	0.656	0.730	0.037	56.8
HIS	0.548	0.549	0.015	0.530	0.536	0.011	51.2
QTF	0.359	0.305	0.009	0.632	0.717	0.005	59.4
Sem-TCP	0.763	0.786	0.039	0.734	0.751	0.031	65.3
Syn-TCP	0.788	0.802	0.043	0.741	0.772	0.036	64.5
SemSyn-TCP	0.829	0.836	0.046	0.776	0.784	0.045	66.7
His-TCP	0.763	0.826	0.020	0.742	0.745	0.017	62.6

Syn-TCP;④只依赖历史信息的His-TCP。这些变体分别代表了TCP可能依赖的不同信息源或处理策略,与SSH-TCP在各个项目中的比较结果如表2所示。

由此可知,SSH-TCP在平均APFD_c指标上达到了0.847,显著优于其他单一维度的变体模型,提升幅度介于9.15%~15.40%,平均APFD指标也存在类似趋势。同时,SemSyn-TCP在APFD、APFD_c指标上的表现优于单独依赖语义分析的Sem-TCP与单独依赖语法分析的Syn-TCP,证明了在TCP中从多维度获取代码特征的方法具有明显优势,能全面评估代码变更和测试用例的相关性。此外,结合历史执行信息的SSH-TCP在所有指标上的表现优于SemSyn-TCP,进一步证实了SSH-TCP在综合多种数据源和分析技术后,在测试用例优先级排序效果方面具有显著优势。

为了研究SSH-TCP与各变体间的显著性差异,还进行了Wilcoxon符号秩检验,实验结果如表2所示。由此可知,所有 p 值均低于0.05,表明各变体对SatTCP的性能均具有积极作用。

3.3.4 时间效率分析

由表2中APTR的数据可知,SSH-TCP在此指标上表现出较优的性能,其值达到68.3%,略高于目前最佳的基线方法SatTCP。进一步研究SSH-TCP的实现细节发现,SSH-TCP在测试用例排序的时间相较于SatTCP稍长,原因在于SSH-TCP综合考虑了测试用例与代码变更之间的语法相似度、语义相似度及测试用例的历史执行信息等因素,虽增加了排序时间,但降低了测试用例集检测所有故障所需的总执行时间。

此外,Sem-TCP、Syn-TCP、SemSyn-TCP和His-TCP的APTR值介于62.6%~66.7%之间,均低于SSH-TCP,进一步证明了SSH-TCP在综合多种分析技术后,在持续集成测试中的时间效率更高。

3.3.5 项目比较

为了减少算法随机性造成的影响,在表1的5个项目数据集上进行30次实验取其平均值,这5个项目的APFD、APFD_c性能指标情况如表3所示。由此可知,gchq-Gaffer项目的平均APFD、平均APFD_c分别达到0.892和0.874,显示出较高的有效性;resteas-esteasy项目的平均APFD、平均APFD_c在5个项目中的最低,分别为0.791和0.795。

结合表1的项目信息,gchq-Gaffer项目中测试用例失败率为9.14%,高于其他项目。然而,resteas-esteasy项目的测试用例失败率为0.35%,分析发现该项目中大部分测试用例的执行历史均为0,无法有效根据历史信息来评估测试用例在下个周期失效的可能性。

进一步观察表3的数据发现,随着项目测试用例失败率的提升,APFD、APFD_c的平均值和中位数总体呈上升趋势,证明了在测试用例失败率较高的项目中,SSH-TCP方法能更有效地利用历史数据进行优先级排序,可提升测试

用例集的故障检测能力。

然而,在测试用例失败率低的项目中,由于缺乏足够的有效信息,导致其在项目中的表现不如在失败率较高的项目中更优秀。

Table 3 APFD and APFD_c for different projects

表3 不同项目的APFD、APFD_c

项目名称	APFD- mean	APFD- median	APFD _c - mean	APFD _c - median	失败率 /%
gchq-Gaffer	0.892	0.936	0.874	0.905	9.14
codelibs-fess	0.867	0.862	0.826	0.886	6.60
ebean-orm-ebean	0.871	0.821	0.853	0.838	1.63
apache-incubator- druid	0.830	0.854	0.821	0.847	0.50
resteas-esteasy	0.791	0.833	0.795	0.804	0.35

4 结语

本文为了更全面地分析影响测试用例重要性的多种因素,实现更优的排序效果,提出一个基于代码变更和历史信息的测试用例优先级排序框架SSH-TCP。在粗粒度过滤阶段,利用BM25进行测试用例初筛,有效缩减测试用例候选集;在细粒度排序阶段,引入UniXCoder、AST来提升模型对代码语义和结构的理解能力,以精确识别与代码变更相关的测试用例。

同时,历史执行信息的融合为测试用例提供了更全面的评估视角,进一步提升了排序效果。实验表明,SSH-TCP框架能显著提升测试用例排序效果,相较于传统方法和其他单一维度的分析技术具有明显优势,证明了在复杂的测试环境中综合多种数据源与分析技术的重要性。

本文在设计、实施SSH-TCP框架时虽取得了一定成效,但仍存在一些局限性,主要体现为:实验部分仅选取数据集的部分项目进行测试,未充分验证SSH-TCP框架在其他项目上的泛化能力;目前考虑的因素主要包括代码变更和历史信息,但测试用例的执行时间、其他多种特征(测试用例的依赖关系、测试用例与需求的关联度等)也对测试用例的重要性具有显著影响。

未来,将在更广泛的数据集上进行实验,以验证框架在不同类型、规模的软件项目中的适应性和有效性,并探索测试用例执行时间等更多特征,以全面评估测试用例。

参考文献:

- [1] YANG L, CHEN J, YOU H, et al. Can code representation boost IR-based test case prioritization? [C]// 34th IEEE International Symposium on Software Reliability Engineering, 2023: 240-251.
- [2] 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: 74-93.
- [3] YOO S, HARMAN M. Regression testing minimization, selection and pri-

- oritization: a survey[J]. *Software: Testing, Verification and Reliability*, 2012, 22(2): 67–120.
- [4] 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*, 2022: 122–129.
- [5] 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.
- [6] MONDAL S, NASRE R. Colosseum: regression test prioritization by delta displacement in test coverage[J]. *IEEE Transactions on Software Engineering*, 2021, 48(10): 4060–4073.
- [7] YARAGHI A S, BAGHERZADEH M, KAHANI N, et al. Scalable and accurate test case prioritization in continuous integration contexts[J]. *IEEE Transactions on Software Engineering*, 2022, 49(4): 1615–1639.
- [8] ZHAO Y F, HAO D, ZHANG L. Revisiting machine learning based test case prioritization for continuous integration[C]// *IEEE International Conference on Software Maintenance and Evolution*, 2023: 232–244.
- [9] SAHA R K, ZHANG L, KHURSHID S, et al. An information retrieval approach for regression test prioritization based on program changes[C]// *37th IEEE International Conference on Software Engineering*, 2015: 268–279.
- [10] AZIZI M. QRTest: automatic query reformulation for information retrieval based regression test case prioritization[C]// *IEEE International Conference on Software Testing, Verification and Validation Workshops*, 2021: 254–262.
- [11] PENG Q, SHI A, ZHANG L. Empirically revisiting and enhancing IR-based test-case prioritization[C]// *29th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2020: 324–336.
- [12] LI Y, WANG Z, WANG J, et al. Semantic-aware two-phase test case prioritization for continuous integration[J]. *Software Testing, Verification and Reliability*, 2024, 34(1): e1864.
- [13] MARIJAN D. Comparative study of machine learning test case prioritization for continuous integration testing[J]. *Software Quality Journal*, 2023, 31(4): 1415–1438.
- [14] 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*, 2022: 141–148.
- [15] THUNES C. C2nes/javalang[EB/OL]. <https://github.com/c2nes/javalang>.
- [16] JONES K S, WALKER S, ROBERTSON S E. A probabilistic model of information retrieval: development and comparative experiments: part 2[J]. *Information Processing & Management*, 2000, 36(6): 809–840.
- [17] GUO D, LU S, DUAN N, et al. UniXcoder: unified cross-modal pre-training for code representation[DB/OL]. <https://arxiv.org/abs/2203.03850>.
- [18] HU Y, ZOU D, PENG J, et al. TreeCen: building tree graph for scalable semantic code clone detection[C]// *37th IEEE/ACM International Conference on Automated Software Engineering*, 2022: 1–12.
- [19] 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.
- [20] XIAO L, MIAO H, SHI T, et al. LSTM-based deep learning for spatial-temporal software testing[J]. *Distributed and Parallel Databases*, 2020, 38: 687–712.
- [21] PAN R, BAGHERZADEH M, GHALEB T A, et al. Test case selection and prioritization using machine learning: a systematic literature review[DB/OL]. <https://arxiv.org/abs/2106.13891>.
- [22] CHEN J, LOU J, ZHANG L, et al. Optimizing test prioritization via test distribution analysis[C]// *26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2018: 656–667.
- [23] HAGBERG A, MILLMAN J, SCHULT D, et al. Networkx/networkx[EB/OL]. <https://github.com/networkx/networkx>.
- [24] BRUCHER M. Sklearn[EB/OL]. <https://scikit-learn.org/stable>.
- [25] REY D, NEUHÄUSER M. Wilcoxon-signed-rank test[M]. Berlin: Springer, 2011.

(责任编辑:刘嘉文)