软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2024,35(6):2821−2843 [doi: 10.13328/j.cnki.jos.006906]

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

基于多样性 SAT 求解器和新颖性搜索的软件产品线测试*

向 毅[1], 黄 翰[1], 罗 川[2], 杨晓伟[1]

[1](华南理工大学 软件学院, 广东 广州 510006)

[2](北京航空航天大学 软件学院, 北京 100191)

通信作者: 黄翰, E-mail: hhan@scut.edu.cn

摘　要: 软件产品线测试是一项非常具有挑战性的工作. 基于相似性的测试方法通过提升测试集的多样性以达到提高测试覆盖率和缺陷检测率的目的. 因其具有良好的可拓展性和较好的测试效果, 目前已成为软件产品线测试的重要手段之一. 在该测试方法中, 如何产生多样化的测试用例和如何维护测试集的多样性是两个关键问题. 针对以上问题, 提出一种基于多样性可满足性 (SAT) 求解器和新颖性搜索 (novelty search, NS) 的软件产品线测试算法. 具体地, 所提算法同时采用两类多样性 SAT 求解器产生多样化的测试用例. 特别地, 为了改善随机局部搜索 SAT 求解器的多样性, 提出一种基于概率向量的通用策略产生候选解. 此外, 为同时维护测试集的全局和局部多样性, 设计并运用两种基于 NS 算法思想的归档策略. 在 50 个真实软件产品线上的消融和对比实验验证多样性 SAT 求解器和两种归档策略的有效性, 以及所提算法较其他主流算法的优越性.

关键词: 软件产品线测试; 可满足性求解器; 新颖性搜索

中图法分类号: TP311

中文引用格式: 向毅, 黄翰, 罗川, 杨晓伟. 基于多样性SAT求解器和新颖性搜索的软件产品线测试. 软件学报, 2024, 35(6): 2821–2843. http://www.jos.org.cn/1000-9825/6906.htm

英文引用格式: Xiang Y, Huang H, Luo C, Yang XW. Software Product Line Testing Based on Diverse SAT Solvers and Novelty Search. Ruan Jian Xue Bao/Journal of Software, 2024, 35(6): 2821–2843 (in Chinese). http://www.jos.org.cn/1000-9825/6906.htm

Software Product Line Testing Based on Diverse SAT Solvers and Novelty Search

XIANG Yi[1], HUANG Han[1], LUO Chuan[2], YANG Xiao-Wei[1]

[1](School of Software Engineering, South China University of Technology, Guangzhou 510006, China)

[2](School of Software, Beihang University, Beijing 100191, China)

Abstract: Software product line testing is challenging. The similarity-based testing method can improve testing coverage and fault detection rate by increasing the diversity of test suites. Due to its excellent scalability and satisfactory testing effects, the method has become one of the most important test methods for software product lines. How to generate diverse test cases and how to maintain the diversity of test suites are two key issues in this test method. To handle the above issues, this study proposes a software product line test algorithm based on diverse SAT solvers and novelty search (NS). Specifically, the algorithm simultaneously uses two types of diverse SAT solvers to generate diverse test cases. In particular, in order to improve the diversity of stochastic local search SAT solvers, the study proposes a general strategy that is based on a probability vector to generate candidate solutions. Furthermore, two archiving strategies inspired by the idea of the NS algorithm are designed and applied to maintain both the global and local diversity of the test suites. Ablation and comparison experiments on 50 real software product lines verify the effectiveness of both the diverse SAT solvers and the two archiving strategies, as well as the superiority of the proposed algorithm over other state-of-the-art algorithms.

Key words: software product line testing; satisfiability solver; novelty search

* 基金项目: 广东省科技攻关重点项目 (2020B0303300001); 2018–2019 年度广东省“新一代人工智能”重大项目 (2020AAA0108404); 国家自然科学基金 (61906069, 62276103, 62202025, 61876207); 广东省基础与应用基础研究基金 (2019A1515011700, 2020A1515010696, 2022A1515011491); 中央高校基本科研业务费专项资金 (2020ZYGXZR014); 广东省财税大数据重点实验室开放基金 (2022kyc021)
收稿时间: 2022-07-12; 修改时间: 2022-09-19; 采用时间: 2023-01-19; jos 在线出版时间: 2023-07-05
CNKI 网络首发时间: 2023-07-07

1 引 言

软件产品线 (software product line, SPL)[1]是一种有效的软件开发方法, 它在满足软件产品共性的基础上, 可灵活地根据用户需求实现软件定制. 国内外众多知名企业, 如华为 (Huawei)、波音 (Boeing)、西门子 (Siemens)、东芝 (Toshiba) 等, 均有采用软件产品线进行产品研发的经历[2,3]. 该技术一方面可更好地满足终端用户多样化的需求, 实现产品定制; 另一方面又可节约开发成本、降低维护工作量以及缩短产品上市周期等[4]. 例如, 著名的 Linux 操作系统、Eclipse IDE、Drupal 网站开发系统、Amazon 等均是借助产品线技术开发的软件产品. 事实上, 软件产品线技术采用可复用的模块化软件构件实现软件产品集的开发, 该集合中的软件产品通常由特征模型 (feature model, FM) 进行描述[5,6]. 其中, 一个特征通常指系统的某个功能[1], 而一个产品则是一个特征的集合. 特征模型明确了特征之间的约束关系, 进而定义构成有效 (或可行) 软件产品的所有特征组合.

软件产品线带来诸多益处的同时, 也带来一些挑战, 其中之一便是如何保障软件产品线的可靠性. 显然, 软件产品线测试是降低缺陷 (faults) 在产品间传播风险的关键所在[7]. 事实上, 单一特征中的缺陷有可能存在于成千上万个产品之中[2]. 若不进行充分的测试, 那么将可能出现大量有缺陷的产品. 然而, 软件产品线测试又是一项非常困难的工作, 其原因在于待测试的软件产品数随特征数的增加而呈指数式增长. 理想情况下, 所有有效软件产品均需被测试, 但这在实际中几乎不可行 (因为待测试的软件产品数量极大). 此外, 即使对小规模软件产品线而言, 测试所有软件产品是可行的, 但这将导致效率低下. 现实中, 软件测试工程师的测试资源, 如时间和成本等, 往往是有限的.

因此, 软件测试工程师正不断寻找能减少测试用例个数的测试方法[8]. 其中, 组合交互测试[9]是一种既能减小测试集规模, 同时又能保证覆盖一定特征组合的测试方法[10,11]. 该测试方法的基本假设是: 绝大多数缺陷是由少量特征的交互触发的[12]. 特别地, t-组合测试要求找到一个可覆盖任意 t 个特征所有可能组合的最小测试集[13]. 这是一个 NP 难问题, 研究者们提出了众多贪心和启发式方法予以求解. 典型算法有 CASA[14], ICPL[15], ACTS[16]和 IncLing[17]等. 然而, 现有的 t-组合测试算法面临可扩展性差的问题[18]. 对大规模的真实 SPLs, 这些方法经常遇到内存占用率过高、无法停止和运行时间过长等难题[18−20]. 虽然 t-组合测试较穷举测试可显著地减少待测试的软件产品个数, 但这个数可能还是过大而无法满足实际需求. 例如, 由文献 [15] 知, 为特征数为 6888 的 Linux 内核特征模型 (即 2.6.28.6-icse11)[21]实现完全 2-组合覆盖, 需产生至少 480 个软件产品. 事实上, n 个特征的最大 t-组合数为 $C_n^t \cdot 2^t$[12]. 显然, 当 n 和 t 至少一个较大时, 覆盖所有特征组合需产生大量的软件产品. 然而, 这在现实世界中非常容易发生: 一方面, 绝大多数真实 SPLs 是大规模的, 包含成千上万个特征 (即 n 很大)[21]; 另一方面, 为了发现更多缺陷, 高交互强度 ($t>2$) 往往是必要的[22−24]. 例如, Kuhn 等人[12]指出, 若实现完全 6-组合覆盖, 则可检测出几乎所有缺陷. 考虑到以上事实, t-组合测试在实际应用中可能会受到一定限制, 因为它难以有效处理所涉及的大规模和高交互强度的情形.

解决 t-组合测试可扩展性差的可行途径之一是引入基于相似性的测试方法[25,26], 其基本假设是: 不相似的测试用例较相似的测试用例能检测到更多的软件缺陷[27,28]. 它与前面提及的 t-组合测试显著不同. 具体地, t-组合测试运用测试端标准, 即 t-组合覆盖率, 来决定何时停止增加新的测试用例. 当测试集生成后, 必然获得完全 t-组合覆盖. 相反, 基于相似性的测试则运用相似性 (或距离) 度量作为选择下一个测试用例的标准. 在生成测试集的任何时候, 需增强当前测试集的多样性以实现特定目标, 如最大化缺陷检测率或覆盖率等. 因此, 基于相似性的测试无法确保 100% 的 t-组合覆盖率, 可能会在一定程度上影响测试集的缺陷检测能力[29]. 此外, 在 t-组合测试中, 测试用例个数及算法的终止均无法人为控制. 相反, 基于相似性的测试则可灵活地设置待生成的测试用例个数及运行时间[20]. 在实际应用中测试资源有限的情形下, 上述灵活性大有裨益. 当前, 已存在一些基于相似性的软件产品线测试方法[20,29−33]. 有关这些方法的详细介绍将在第 2 节给出.

在基于相似性的软件产品线测试方法中, 如何产生可行且多样的测试用例以及如何维护所产生测试用例的多样性是两个关键问题. 针对前者, 实践表明[20,33,34]: 采用可满足性 (satisfiability, SAT) 求解器是生成可行测试用例的一种有效方法. 数学上, 特征之间的约束可表达为合取范式 (conjunction normal form, CNF)[5], 这正是 SAT 求解

器所处理对象的标准输入形式. 为了提高所生成测试用例的多样性, 研究人员针对 SAT 求解器本身的改进或者其灵活运用开展了持续性研究. 例如, Henard 等人[20,35]提出了 SAT4J[36]求解器内部参数的随机化策略 (本文称这种改造后的求解器为 rSAT4J); Luo 等人[34]提出了一种面向软件产品线测试的随机局部搜索 SAT 求解器, 即 LS-Sampling; Xiang 等人[33]建议同时运用两类求解器, 即冲突驱动的子句学习 (conflict-driven clause learning, CDCL) 求解器和随机局部搜索 (stochastic local search, SLS) 求解器. 文献 [33] 所采用的求解器为 rSAT4J (属于 CDCL 求解器) 和 ProbSAT[37] (属于 SLS 求解器), 但仅前者是一种多样化的 SAT 求解器. 若将后者也进行多样化处理, 是否有助于进一步改善测试效果? 这是一个有待探索的问题. 针对后者, 既往工作[20,34]并未予以足够重视. 事实上, 若采用多样化 SAT 求解器生成的测试用例无法在最终测试集得以保留, 那么多样性 SAT 求解器的作用也将大打折扣. 文献 [33] 首次采用新颖性搜索 (novelty search, NS) 算法[38]中的归档策略维护测试集的多样性, 取得了较好的效果. 但是, 该归档策略仅从全局的角度出发考虑测试集的多样性, 而忽略了新测试用例所在区域的局部信息 (如新测试用例与其最近邻的距离等). 同时提升测试集的全局和局部多样性对测试效果带来怎样的影响? 这也是一个有待回答的问题.

在既往工作[33]的基础上, 本文提出了一种基于多样性 SAT 求解器和新颖性搜索的软件产品线测试算法 (命名为 dSATNS), 它同时采用了两种不同类型的多样化 SAT 求解器产生测试用例, 并且运用 NS 算法的两种 (全局和局部) 归档策略维护测试集的多样性. 为了提高 SLS 求解器的多样性, 本文提出了一种基于概率向量的通用策略为该类求解器产生多样的候选解/种子 (详细介绍见第 4.2 节). 在 50 个真实软件产品线上的实验结果表明, 采用两类多样性 SAT 求解器的确有助于提高软件产品线测试的覆盖率和缺陷检测率. 实验结果还发现, 仅采用全局和局部归档策略分别适合测试集规模较大和较小的情形, 而同时采用两种策略则能获得折衷效果. 最后, 对比实验结果揭示, 本文所提出的算法较主流算法具有显著优势: 无论测试集规模的大小, dSATNS 总是排名第一. 特别地, 当测试集规模较小时, dSATNS 的优势愈明显. 在测试资源 (时间、成本等) 有限的情况下, 测试人员往往希望通过运行尽可能少的测试用例以发现尽可能多的缺陷, 而本文所提出的算法为生成这样的测试用例的集合提供了一种行之有效的解决方案.

本文的主要贡献包括: 1) 提出了一种基于概率向量的通用策略为 SLS 求解器生成多样化的种子. 该策略简单易实现, 可在不改变现有 SLS 求解器内部搜索机理的情形下, 辅助求解器生成多样化的测试用例. 2) 根据 NS 算法的思想, 设计了一种局部归档策略维护测试集的多样性. 该策略与现有的全局归档策略[33]形成互补效应. 3) 综合运用两类多样性求解器和两种归档策略, 提出了一种面向软件产品线测试的 dSATNS 算法, 其性能显著优于当前主流算法.

本文第 2 节回顾基于相似性的软件产品线测试方法. 第 3 节介绍本文所需的基础知识, 包括特征模型和软件产品线测试相关概念. 第 4 节详细介绍本文提出的软件产品线测试算法 dSATNS. 第 5 节通过消融和对比实验验证各算法构件的有效性以及本文算法较其他主流算法的优越性, 并探讨关键参数对算法性能的影响. 第 6 节总结全文.

2 基于相似性的软件产品线测试相关工作

Henard 等人[20]通过最大化测试用例的多样性来模拟 *t*-组合测试. 具体地, 他们提出了一种基于相似性的适应度函数并采用遗传算法 (genetic algorithm, GA) 进行优化. 这样做的目的是生成一组尽可能不相似 (多样) 的测试用例, 从而潜在地覆盖更多的特征组合 (或软件缺陷). 此外, 为提高 SAT4J 求解器[36]所生成测试用例的多样性, 作者建议随机化该求解器的内部参数, 即为变量赋值的策略 (包括正文字优先、反文字优先和随机文字). 实验结果表明, Henard 等人的方法可处理大规模和高交互强度的情形, 是 *t*-组合测试的一种可拓展的、灵活的替代方案. 测试用例生成工具 PLEDGE[30]实现了上述方法. 随后, Fischer 等人[29]的实验结果表明, Henard 等人[20]的方法能有效地检测出软件产品线 Drupal[39]的真实缺陷. 上述方法可取得与 CASA[14]相当的 *t*-组合覆盖率. 这些研究结果充分说明, 相似性指标是 *t*-组合覆盖率的一种恰当的替代.

Al-Hajjaji 等人[2,7]系统地研究了基于相似性的测试用例排序, 即确定测试用例的顺序从而尽可能快地发现缺陷. 他们将相似性定义为待选测试用例与所有已选测试用例的最小距离, 而非 Henard 等人[20]所定义的距离之和 (某些情形下具有一定误导性[2]). 随后, Al-Hajjaji 等人[31]又探讨了如何利用不同类型的信息度量软件产品之间的相似性. 特别地, 他们将计算相似性的输入信息分为问题空间信息 (如特征选择相似性、属性相似性和实例相似性) 和解空间信息 (如家族模型相似性). 此外, 他们还探讨了结合不同类型的信息以构造产品综合相似性指标的可能性.

针对行为 SPLs, Devroey 等人[32]研究了基于相似性的测试用例生成问题. 结果表明, 基于相似性的测试用例生成与行为 SPL 模型是有关联的. 此外, 作者还比较了多个度量测试用例相似性的距离, 发现 Hamming 和 Jaccard 距离最为有效. 从既往工作的实验结果看, 相似性指标的改进的确能获得不错的 t-组合覆盖率或缺陷检测率. 相似性指标与 t-组合覆盖密切相关也符合人们的直觉. 但是, 基于相似性的软件产品线测试方法为什么有效并未得到强有力的解释.

最近, Xiang 等人[33]借助相关性分析发现, 相似性指标与 t-组合覆盖率之间是呈显著正相关的. 该发现有利于从统计学角度阐明基于相似性的软件产品线测试的基本原理: 相似性指标的改善可潜在地提升 t-组合覆盖率. 此外, 为维护并提升测试集的多样性, 他们采用 NS 作为搜索算法. 作为进化算法的一种新范式, 该算法不断奖励“新颖”个体, 即与已发现的个体不同的个体. 上述算法机理高度契合基于相似性的软件产品线测试的目标, 故采用 NS 算法是非常恰当的. 文献 [33] 的实验结果也证实了上述断言. 最后, 他们还从实验上论证, 针对软件产品线测试用例的生成问题, 将 CDCL 求解器与 SLS 求解器相结合是一种非常有前景的方法.

与文献 [33] 一样, 本文也采用了两类 SAT 求解器和 NS 算法中的归档策略. 不同的是, 本文采用的是两类多样化的 SAT 求解器, 其中多样化的 SLS 求解器是由本文提出的一种通用策略实现的. 此外, 本文同时运用了文献 [33] 中的全局归档策略和我们提出的一种局部归档策略, 兼顾了测试集的全局和局部多样性, 形成了互补效应, 而文献 [33] 仅考虑了测试集的全局多样性. 从第 5 节的实验结果将看到, 多样化 SAT 求解器和两种归档策略的综合运用显著改进了文献 [33] 中的算法.

3 基础知识

本文研究软件产品线的测试问题, 下面就相关概念和基本知识予以介绍.

3.1 特征模型

如第 1 节所述, 软件产品线采用可复用的软件模块系统式地组装软件产品, 这些产品具有共性, 也有特性. 特征模型 (FM) 是一种表示软件产品线共性与特性的直观、简洁方法. 视觉上, 特征模型是一种树状结构, 其中每个节点代表一个特征, 代表系统的某个功能模块, 边则明确了特征之间的约束关系. 例如, 图 1 所示的特征模型表示的是一个简化后的移动手机软件产品线, 共包含 10 个特征 (模块). 一般地, 特征模型中存在 6 大类约束关系, 分别为强制的 (mandatory), 可选的 (optional), 或 (or), 排斥或 (xor), 需要 (require) 和排斥 (exclude). 研究表明[5], 上述所有约束关系均可由命题公式表示, 进一步可转换为等价的合取范式 CNF. 图 1 中的特征模型所描述的约束关系可转换为以下 CNF.

$$\begin{aligned} FM &= x_1 \wedge (\neg x_1 \vee x_2) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_4) \wedge (x_1 \vee \neg x_4) \wedge (x_1 \vee \neg x_3) \wedge (x_1 \vee \neg x_5) \wedge (x_4 \vee \neg x_6) \\ &= \wedge (x_4 \vee \neg x_7) \wedge (x_4 \vee \neg x_8) \wedge (\neg x_4 \vee x_6 \vee x_7 \vee x_8) \wedge (\neg x_6 \vee \neg x_7) \wedge (\neg x_6 \vee \neg x_8) \wedge (\neg x_7 \vee \neg x_8) \\ &= \wedge (x_5 \vee \neg x_9) \wedge (x_5 \vee \neg x_{10}) \wedge (\neg x_5 \vee x_9 \vee x_{10}) \wedge (x_8 \vee x_9) \wedge (\neg x_3 \vee \neg x_6) \end{aligned} \tag{1}$$

其中, 每个析取式 (如 $\neg x_1 \vee x_2$ 等) 称为子句. 所有子句的集合称为特征模型的功能性约束集. 值得说明的是, CNF 是 SAT 问题的标准形式. 因此, 为变量赋值使得它们满足所有功能性约束本质上是求解一个 SAT 问题.

更正式地, 若采用 $x_1, x_2, \ldots, x_n$ 表示特征模型的 n 个特征, 用 $l_1, l_2, \ldots, l_m$ 表示特征模型对应 CNF 约束中的 m 个子句. 我们有以下定义:

定义 1. 特征模型可看作一个二元组 (X, L), 其中 $X = \{x_1, x_2, \ldots, x_n\}$ 表示 n 个布尔型特征 (即变量), $L = \{l_1, l_2, \ldots, l_m\}$ 表示特征之间的 m 个约束关系. 称 L 是满足的当且仅当所有 l_i $(i = 1, 2, \ldots, m)$ 均是满足的.

定义 2. 特征模型的一个配置 (configuration) 或产品 (product) 是一个集合 $cf = \{\pm x_1, \pm x_2, \ldots, \pm x_n\}$, 其中 $+x_i$ 和 $-x_i$ 分别表示第 i 个特征 x_i 被选中和未被选中. 注意一个特征仅有被选中和未被选中两种状态, 且必居其一. 这种集合式表示的产品可自然地转换成二元向量 $b = [b_1, b_2, \ldots, b_n]$, 其中 b_i 取 1 (或 true) 当且仅当 x_i 被选中; 否则, b_i 取 0 (或 false). 一个配置 cf 是有效 (或可行) 的当且仅当特征模型的约束集 L 均得以满足. 否则, 称该配置是无效的 (或不可行的). 集合形式的定义便于解释如何计算配置之间的距离, 而向量形式的定义则与本文所提算法的实现直接相关.

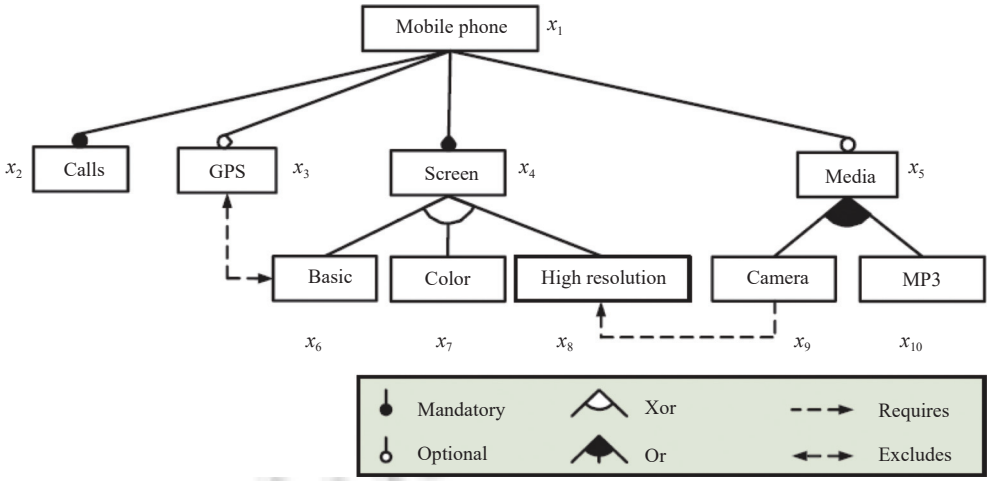


图 1 简化的移动手机软件产品线所对应的特征模型[40]

3.2 软件产品线测试相关定义

关于软件产品线测试有以下定义.

定义 3. 软件产品线的测试集 (test suite) 是一个列表 $TS = [tc_1, tc_2, \ldots, tc_N]$, 其中 tc_i 表示一个测试用例 (test case), 即某个有效的软件配置, N 表示测试集规模.

软件产品线测试旨在覆盖所有 (或尽可能多) 的 t 个特征的组合, 其中 t=1, 2,..., 于是, 我们有:

定义 4. 若 $\pi_1, \pi_2, \ldots, \pi_t$ 表示集合 $\{1, 2, \ldots, n\}$ 的任意 t ($\leqslant n$) 个不同元素, 则称集合 $\{\pm x_{\pi_1}, \pm x_{\pi_2}, \ldots, \pm x_{\pi_t}\}$ 为一个 t-集合 (t-set). 事实上, t-集合是一个部分配置的产品, 它是有效的当且仅当它未违反 L 中的任何约束.

定义 5. 软件配置 cf 覆盖某个有效的 t-集合 (记为 $tset$), 当且仅当 $cf \supseteq tset$ 成立. 注意, 测试时我们无需覆盖无效的 t-集合.

定义 6. 一个测试集 TS 的 t-组合覆盖率 (简称覆盖率) 定义为比率:

$$\frac{\left|\bigcup_{i=1}^{N} VT_{tc_i}\right|}{|VT_{fm}|} \times 100\% \tag{2}$$

其中, VT_{fm} 表示给定特征模型 (记作 fm) 的所有有效 t-集合的集合; VT_{tc_i} 表示测试用例 tc_i 所覆盖的所有有效 t-集合的集合; $|\cdot|$ 表示集合的势. 显然, 覆盖率最大为 100%, 是测试所追求的最高目标.

4 基于多样性 SAT 求解器和新颖性搜索的软件产品线测试方法

本文所提出的 dSATNS 算法的核心思想是: 采用多样性 SAT 求解器产生测试用例, 运用 NS 算法的归档策略维护测试集的多样性, 并以迭代改进的方式不断提升当前测试集的多样性, 进而覆盖更多特征组合或发现更多缺陷. 如算法 1 所示, 首先采用 rSAT4J 随机地生成一个初始测试集 tc_{worst} (第 1 行); 然后, 初始化概率向量 $p = [p_1, p_2, \ldots, p_n]$ (第 2 行); 接着, 算法进入迭代循环阶段 (第 3–10 行): 只要终止条件未满足, 则采用函数 DiverseSATSolving 生成测试用例 c 并随机地选择归档策略 GlobalArch 或 LocalArch 将 c 加入测试集 TS. 事实上, 算

法 1 遵循了 NS 算法的基本框架, 其主要特色在于运用领域知识生成新解 (即新测试用例) 和采用两种归档策略. 接下来, 本文将详细介绍该算法的一些关键细节.

算法 1. dSATNS 算法.

输入: 测试集规模 N;
输出: 测试集 TS.

1. 采用 rSAT4J 随机地产生初始测试集 $TS=[tc_1,tc_2,\ldots,tc_N]$ // N 为测试集规模
2. 根据 TS 中的个体初始化概率向量 $p=[p_1,p_2,\ldots,p_n]$ // n 为特征数
3. **while** 终止条件未满足 **do**
4. $c \leftarrow$ DiverseSATSolving(p)
5. **if** $rand(0,1)<0.5$
6. GlobalArch(TS, c)
7. **else**
8. LocalArch(TS, c)
9. **end if**
10. **end while**
11. **return** TS

4.1 概率向量及其初始化

本文采用了分布估计算法中概率向量[41,42]的概念. 概率向量 $p=[p_1,p_2,\ldots,p_n]$ 的第 j 个分量 p_j 的含义为第 j 个变量取值为 1 (或 true) 的概率. 显然, 该概率事先未知, 但可用当前种群 (即测试集) 的分布进行估计. 具体地, 令 tc_{ij} 表示第 i 个测试用例第 j 个变量的取值, 则可根据公式 (3) 初始化 p_j:

$$p_j=\frac{\sum_{i=1}^{N}tc_{ij}}{N} \tag{3}$$

也就是说, p_j 被初始化为当前测试集 TS 中第 j 个变量取值为 1 的所有测试用例所占的比例. 概率向量主要用于生成多样化的 (候选) 测试用例. 在算法的迭代过程中, 概率向量会根据当前测试集动态地更新 (第 4.3 节将予以详细介绍).

4.2 多样性 SAT 求解器

如第 1 节所述, dSATNS 采用了两类多样性求解器, 它们的结合方式如算法 2 所示. DiverseSATSolving 以概率 $1-P_r$ 调用 rSAT4J, 而以概率 P_r 调用多样化版本的 ProbSAT, 即 tc (见算法 3). 如前所述, 作为一种 CDCL 求解器, rSAT4J[20,35]是随机化版本的 SAT4J, 旨在产生尽可能不相似的 SAT 解; 而 dProbSAT 则是本文提出的 ProbSAT[37]求解器的一种改进版本. 值得说明的是, 算法 2 同时采用了两种类型的 SAT 求解器, 其原因在于: 既往研究工作已表明, 无论是针对软件产品线配置问题[40]和还是测试问题[33], 这都是一种生成新解/个体的有效策略. 此外, 与文献 [33] 一样, 调用 CDCL 和 SLS 求解器的概率分别设置为 $1-P_r$ 和 P_r. 但与文献 [33] 直接采用原始的 ProbSAT 求解器不同的是, 算法 2 中的 SLS 类型求解器 (即 dProbSAT) 也是一种多样性求解器. 它的基本思想是: 借助概率向量产生一个多样化的初始解 (称为“种子”); 若该初始解不可行, 则调用原始的 ProbSAT 予以修复.

算法 2. $c \leftarrow$ DiverseSATSolving(p).

1. $r \leftarrow rand$ (0, 1) // r 是位于 [0, 1] 区间的随机数
2. **if** $r<1-P_r$ **do** // P_r 为控制参数
3. $c \leftarrow$ rSAT4J() //随机化的 SAT4J 求解器 (CDCL 类型)

4. **else**
5. $c \leftarrow$ dProbSAT (p) //多样性 ProbSAT 求解器 (SLS 类型)
6. **end if**
7. **return** c

算法 3 给出了 dProbSAT 的详细流程. 首先将测试用例 c 初始化为空 (null). 然后, 根据第 i 个特征的类型为变量 c_i 赋初值. 具体地, 若第 i 个特征是强制性的, 则将 c_i 赋值为 1; 若第 i 个特征是废弃的, 则将 c_i 赋值为 0. 所谓强制特征指的是该特征必须出现在任何有效软件配置, 而废弃特征任何时候均不能被选中[30]. 强制和废弃特征对应变量的赋值是固定的, 这是因为任何违反都将直接导致不可行测试用例. 若非上述两种情形, 则将 c_i 赋值为 1 的概率为 $1-p_i$, 赋值为 0 的概率为 p_i (算法 3 的第 8–12 行). 最后, 若以上规则生成的 c 不可行, 则调用原始的 ProbSAT 求解器修复 c (第 15 行). 针对第 5 节中的所有特征模型, 本文统计了 c 为不可行解的百分比, 其最大值为 99.89%, 最小值为 97.56%, 平均值为 99.79%. 以上现象的解释并非难事. 事实上, 特征模型的约束众多且复杂, 对各变量进行独立赋值几乎必然会导致不可行解. 因此, 采用 ProbSAT 等求解器修复 c 是必要的. 值得一提的是, 除了 ProbSAT 求解器外, 还可以采用其他任意 SLS 类型的 SAT 求解器. 作为一个通用框架, 算法 3 适用于任何 SLS 类型的 SAT 求解器.

算法 3. $c \leftarrow$ dProbSAT(p).

1. $c \leftarrow$ null
2. **for** $i \leftarrow 1$ to n **do**
3. **if** 第 i 个特征是强制的
4. $c_i \leftarrow 1$
5. **elseif** 第 i 个特征是废弃的
6. $c_i \leftarrow 0$
7. **else**
8. **if** $rand(0,1) < 1-p_i$
9. $c_i \leftarrow 1$
10. **else**
11. $c_i \leftarrow 0$
12. **end if**
13. **end if**
14. **end for**
15. 若 c 不可行, 则运用 ProbSAT 求解器修复 c
16. **return** c

算法 3 的第 8–12 行是提升测试用例多样性的关键所在. 在当前测试集 TS 中, 第 i 个变量取值为 1 的测试用例的比例为 p_i, 则在种子 c 中第 i 个变量取值为 1 的概率为 $1-p_i$, 而非 p_i. 事实上, p_i 越大 (小), 说明当前测试集中第 i 个变量取值为 1 的测试用例的个数越多 (少). 相反地, 种子中第 i 个变量取值为 1 的概率就越小 (大). 这样处理的目的是形成一种“对抗”, 迫使 c_i 取 1 和取 0 的测试用例的个数不至于相差太大, 进而避免产生聚集的测试用例. 相反, 若采用与分布估计算法相同的做法, 即以概率 p_i 将 c_i 赋值为 1, 则当 p_i 较大时, c_i 赋值为 1 的概率也较大. 这样有可能导致最终测试集中绝大多数甚至全部测试用例的第 i 个变量都取值为 1, 必将对测试用例的多样性产生不利影响. 值得说明的是, 本质上文献 [34] 也采用了概率向量产生种子. 但与本文不同的是, 文献 [34] 的概率向量也参与了后续的随机搜索, 为选择待翻转的变量提供重要启发式信息. 本文的概率向量仅用于生成种子, 这样做的目的是将种子生成阶段与随机局部搜索阶段进行分离, 从而提高本文方法的灵活性. 例如, 本文将基于概率

向量的种子生成策略与 ProbSAT 相结合, 当然也可以与其他任何 SLS 求解器相结合. 另外, 第 5.5.1 节的实验结果表明, 在多数情况下, 仅引入基于概率向量的种子生成策略即可显著提升测试效果. 因此, 在算法设计时, 本文以简洁性和灵活性为重要原则, 仅关注基于概率向量种子生成策略, 而保持 ProbSAT 的内部随机搜索机理机制不变.

4.3 测试集的归档策略

算法 2 产生的多样化的测试用例将被加入测试集 TS. 为维护测试集的多样性, 本文采用了两种归档策略 GlobalArch (见算法 4) 和 LocalArch (见算法 5), 它们均是根据 NS 算法的思想设计的, 分别考虑了测试集的全局和局部多样性. 我们从详细介绍全局归档策略开始. 如算法 4 的第 1 行所示, 合并新产生的测试用例 $\{tc\}$ 与测试集 TS 得到扩展测试集, 记作 TS'. 接着计算个体 $c \in TS'$ 的新颖得分 (novelty score)[38], 其定义如下:

$$\rho(c, TS') = \frac{1}{k}\sum_{i=1}^{k} d(c, tc'_i) \tag{4}$$

其中, tc'_i 表示集合 TS' 中与 c 第 i 近的个体, 称为 c 的第 i 个“邻居”. 超参数 k 明确了参与新颖得分计算的邻居个数. $d(c, tc'_i)$ 表示个体 c 与 tc'_i 之间的距离. 本文采用公式 (5) 定义的 anti-Dice 距离. 研究表明[33], 对于软件产品线测试问题, anti-Dice 是一种有效的距离测度. 注意公式 (5) 的测试用例采用集合形式的表示方法 (参考定义 2).

$$d(c, tc'_i) = 1 - \frac{|c \cap tc'_i|}{2|c \cup tc'_i| - |c \cap tc'_i|} \tag{5}$$

新颖得分是 NS 算法中的核心概念, 它度量了个体所在区域的拥挤程度. 新颖得分越高, 说明个体所在区域越稀疏, 反之则越密集. 通过不断搜索更新颖的个体 (即新颖得分更高的个体), 可逐步提高档案的整体多样性, 这正好契合了基于相似性的软件产品线测试所追求的目标.

算法 4. $TS \leftarrow \text{GlobalArch}(TS, tc)$.

1. $TS' \leftarrow TS \cup \{tc\}$
2. **for** each $c \in TS'$ **do**
3. 根据公式 (4) 计算测试用例 c 的新颖得分 $\rho(tc, TS')$
4. **end for**
5. 找出测试集 TS 中新颖得分最小 (最差) 的个体, 记为 tc_{worst}
6. **if** $\rho(tc, TS') > \rho(tc_{\text{worst}}, TS')$
7. 用 tc 替换 TS 中的 tc_{worst}
8. 更新概率向量: $p_i \leftarrow p_i + (tc_i - tc_{\text{worst},i})/N$, 其中 $i = 1, 2, \ldots, n$
9. **end if**
10. **return** TS

算法 5. $TS \leftarrow \text{LocalArch}(TS, tc)$.

1. 找到 TS 中与 tc 距离最近的测试用例, 记为 tc_{closest}
2. 根据公式 (4) 计算测试用例 tc 的新颖得分 $\rho(tc, TS \setminus \{tc_{\text{closest}}\})$
3. 根据公式 (4) 计算测试用例 tc_{closest} 的新颖得分 $\rho(tc_{\text{closest}}, TS)$
4. **if** $\rho(tc, TS \setminus \{tc_{\text{closest}}\}) > \rho(tc_{\text{closest}}, TS)$
5. 用 tc 替换 TS 中的 tc_{closest}
6. 更新概率向量: $p_i \leftarrow p_i + (tc_i - tc_{\text{closest},i})/N$, 其中 $i = 1, 2, \ldots, n$
7. **end if**
8. **return** TS

新个体 tc 是否能加入档案 TS 是由其新颖得分决定的. 如算法 4 的第 5 行所示, 找到当前测试集 TS 中新颖得分最小 (最差) 的个体 tc_{worst}. 若 $\rho(tc, TS') > \rho(tc_{\text{worst}}, TS')$ 成立, 则用 tc 替换 tc_{worst} 并更新概率向量; 否则, 舍弃新个体 tc. 更新概率向量的操作也非常简单 (第 8 行), 只需将当前 p_i 加上 $(tc_i - tc_{\text{worst},i})/N$. 根据上述操作, 若 tc_i 与 $tc_{\text{worst},i}$ 取值相同, 则 p_i 保持不变; 若 tc_i 与 $tc_{\text{worst},i}$ 取值不同, 则 p_i 更新为 $p_i + 1/N$ 或 $p_i - 1/N$. 具体而言, 存在两种情形.

- 当 $tc_i = 1$ 且 $tc_{\text{worst},i} = 0$ 时, 这意味着第 i 个变量取值为 1 的测试用例个数增 1, 自然地 p_i 应更新为 $p_i + 1/N$.
- 当 $tc_i = 0$ 且 $tc_{\text{worst},i} = 1$ 时, 这意味着第 i 个变量取值为 1 的测试用例个数减 1, 自然地 p_i 应更新为 $p_i - 1/N$.

这两种情形都可用公式 $p_i = p_i + (tc_i - tc_{\text{worst},i})/N$ 进行表达. 从时间复杂度的角度看, 这种更新效率非常高. 相反地, 文献 [34] 的算法更新概率向量时, 每次均需对当前测试集中第 i 个变量取 *true* 或 *false* 的测试用例重新计数, 效率相对较低.

算法 4 将新个体与当前测试集的最差个体进行比较. 若新个体更优, 则执行替换操作. 上述归档策略有利于从全局的角度提升测试集的多样性, 但却忽略了新个体所在区域的局部信息. 例如, 新个体可能与档案中的某个个体距离非常近, 但由于与其他邻居的距离较远, 导致新个体的新颖得分并不低. 在归档时, 新个体有可能会被加入档案. 在这种情况下, 新个体的加入会导致局部聚集现象. 为缓解上述现象, 本文提出了一种基于局部信息的归档策略 (见算法 5). 首先, 在 TS 中找到与新测试用例 tc 最近的测试用例 tc_{closest}, 然后按照公式 (4) 分别计算 tc 和 tc_{closest} 的新颖得分. 值得说明的是, 计算二者的新颖得分时均将对方排斥在外. 例如, 计算 tc 的新颖得分时, 仅考虑 TS 中除 tc_{closest} 外的个体, 即 $TS \backslash \{tc_{\text{closest}}\}$. 类似地, 计算 tc_{closest} 的新颖得分时, 并不将 tc 纳入考虑范围. 最后, 如果 tc 较 tc_{closest} 具有更高的新颖得分, 则用 tc 替换 tc_{closest} 并更新概率向量 p. 由此可见, 在局部归档策略中, 新测试用例及与其最近的测试用例不能同时存在, 有助于改善测试集的局部多样性.

同时采用全局和局部归档策略有望充分利用各自优势, 达到互补效果. 在第 5.5.2 节本文将从实验上予以充分论证.

4.4 终止条件

dSATNS 算法终止条件的设置较为灵活. 一般地, 当算法的运行时间达到预设的最大值时, 算法终止并返回最终的测试集. 由于测试集的多样性是以迭代的方式改进的, 用户完全可以根据自己的需要设置最大运行时间. 例如, 若用户希望覆盖更多特征组合或发现更多缺陷, 则可将运行时间设置为某个较大值. 在实际应用中, 终止条件的灵活性非常有益, 因为不同用户可根据实际需要自主地确定何时终止算法.

5 实验部分

为验证 dSATNS 各算法构件的有效性以及评估该算法的性能, 本文将回答以下研究问题.

- RQ1: 在 dSATNS 算法中, 采用 dProbSAT 求解器是否有助于改善覆盖率和缺陷检测率?
- RQ2: 在 dSATNS 算法中, 是否有必要采用两种归档策略?
- RQ3: 与主流算法相比, dSATNS 的性能如何?
- RQ4: 参数 P_r 如何影响 dSATNS 算法的性能?

RQ1 目的在于验证多样性求解器的有效性. 为回答 RQ1, 将 dSATNS 算法中的 dProbSAT 替换为普通版本的 ProbSAT 得到变体算法 SATNS. 具体地, SATNS 生成种子时随机地将 c_i 赋值为 1 或 0, 而 dSATNS 则是以概率 $1 - p_i$ 将变量 c_i 赋值为 1. 注意是否采用多样化的 ProbSAT 求解器是 dSATNS 与 SATNS 的唯一差别. 本文并不对 rSAT4J 是否有效进行消融实验, 其原因在于该求解器的有效性已在既往工作[20,33]得到充分验证.

RQ2 旨在验证同时采用两种归档策略的优势. 为回答 RQ2, 将 dSATNS 算法中的两种归档策略分别统一为 GlobalArch 和 LocalArch 得到两个变体算法, 记为 GloArch 和 LocArch. 此外, 在有关 RQ2 的实验与讨论中将 dSATNS 重命名为 TwoArch 以凸显 RQ2 的主题.

RQ3 是为了将本文算法与其他主流算法进行比较. 为回答 RQ3, 将 dSATNS 与 TSENS[33], TSEGA[33]和

rSAT4J[20]进行比较. 其中, TSENS 是本文算法的直接基础, 它与 dSATNS 的区别在于: TSENS 未采用多样化的 SAT 求解器且仅使用了全局型档案. 值得注意的是, 在生成“种子”时, TSENS 随机地为每个变量赋值 0 或 1. 将这种以随机方式生成种子的策略替换成遗传算子 (选择、交叉和变异) 即得到算法 TSEGA. 最后, rSAT4J 是一个被广泛采用的基准算法[20].

RQ4 是为了探索参数 P_r 对 dSATNS 算法性能的影响. P_r 是本文所提算法的一个重要参数, 它控制了调用两类求解器的比例. 将 P_r 以一定间隔 (如 0.1) 从 0 增加到 1, 然后比较每个可能值对应的算法性能, 进而确定其最优取值.

5.1 实验数据

在实验部分, 本文采用 50 个公开的软件产品线/特征模型 (见表 1 的第 1 列) 进行仿真实验. 在既往工作中[20,33,34,43], 它们也被广泛遴选为测试问题. 除了 SPLOT-FM-5000 之外的所有 SPLs 均代表真实软件产品线或高度可配置系统, 而 SPLOT-FM-5000 则是人为构造的一个软件产品线, 被 Henard 等人[20]用于基于相似性的软件产品线测试的研究. 对所有特征模型, 特征个数 (即 n) 的最小值为 24, 最大值为 14 910; 约束个数最小值为 35, 最大值为 343 944. 我们将这些软件产品线分为 3 类: 小规模软件产品线 ($n < 1000$), 中等规模软件产品线 ($1000 < n \leqslant 5000$) 和大规模软件产品线 ($n > 5000$). 表 1 所展示的软件产品线已按 n 值升序排列, 其中前 23 个是小规模的, 后 4 个是大规模的, 中间 23 个是中等规模的. 所有软件产品线对应的特征模型均采用 DIMACS 这种 SAT 求解器的标准格式进行表示.

表 1 dSATNS 和 SATNS 两算法覆盖率的比较 (%)

FM	N=4		N=6		N=10		N=50		N=100	
	dSATNS	SATNS	dSATNS	SATNS	dSATNS	SATNS	dSATNS	SATNS	dSATNS	SATNS
CounterStrikeSFM	**87.52**	87.52 ±	**95.86**	95.74 ±	99.52	**99.52 ±**	100.00	**100.00 ±**	100.00	**100.00 ±**
HiPAcc	73.31	**74.27** ◦	**84.93**	84.49 ±	**92.84**	92.76 ±	99.88	**99.88 ±**	**100.00**	100.00 •
SPLSSimuelESPnP	**87.78**	87.78 ±	**96.69**	96.27 ±	**99.72**	99.65 ±	**100.00**	100.00 ±	**100.00**	100.00 ±
JavaGC	**72.16**	72.16 ±	81.12	**81.85 ±**	**89.91**	89.83 ±	**99.75**	99.75 ±	100.00	**100.00 ±**
Polly	**76.35**	76.35 ±	**84.47**	84.39 ±	**93.03**	92.92 ±	99.92	**99.92 ±**	100.00	**100.00 ±**
DSSample	60.32	**60.49 ±**	**68.25**	68.25 ±	**77.78**	77.78 ±	96.82	**96.87 ±**	99.19	**99.19 ±**
VP9	75.70	**75.94 ±**	**85.25**	85.19 ±	**92.73**	92.62 ±	**99.96**	99.96 ±	**100.00**	100.00 ±
WebPortal	**80.43**	80.19 ±	**90.54**	89.88 •	**97.00**	96.87 ±	**100.00**	100.00 ±	**100.00**	100.00 ±
JHipster	**76.55**	76.25 ±	**85.53**	84.99 ±	93.30	**93.34 ±**	**99.81**	99.81 ±	99.94	**99.95 ±**
Drupal	**83.04**	82.38 •	**92.99**	92.38 •	**98.64**	98.51 ±	**100.00**	100.00 ±	100.00	**100.00 ±**
SmartHomev2.2	**81.93**	80.68 •	**92.15**	91.48 •	**98.03**	97.79 •	**100.00**	100.00 ±	**100.00**	100.00 ±
VideoPlayer	**85.00**	84.16 •	**94.23**	93.43 •	**99.14**	98.94 •	**100.00**	100.00 ±	**100.00**	100.00 ±
Amazon	**53.68**	53.67 ±	60.42	**60.42 ±**	**67.58**	67.38 ±	**89.92**	89.59 ±	94.73	**94.81** ◦
ModelTransformation	**77.37**	76.72 •	**87.57**	86.83 •	**95.29**	95.06 •	99.99	**99.99 ±**	100.00	**100.00 ±**
CocheEcologico	**78.22**	78.18 ±	**84.81**	84.18 •	**90.80**	90.80 ±	99.29	**99.31 ±**	**99.77**	99.77 ±
Printers	**75.73**	74.73 •	**82.67**	82.01 •	**88.67**	88.43 •	97.54	**97.58 ±**	98.80	**98.81 ±**
fiasco_17_10	**59.79**	59.60 ±	**64.98**	64.70 ±	**71.90**	71.76 ±	82.72	**82.81 ±**	85.47	**85.72 ±**
uClibc-ng_1_0_29	**63.46**	62.70 •	**70.46**	69.97 •	**76.42**	76.37 ±	**88.02**	87.92 ±	91.57	**91.69 ±**
E-shop	**77.35**	75.26 •	**88.17**	86.18 •	**95.71**	95.58 •	99.96	**99.96 ±**	100.00	**100.00 ±**
toybox	**95.04**	94.65 •	**97.92**	97.74 •	**99.48**	99.44 ±	100.00	**100.00 ±**	**100.00**	100.00 ±
axTLS	91.15	**91.19 ±**	**95.59**	95.51 ±	**98.56**	98.45 •	100.00	**100.00 ±**	**100.00**	100.00 ±
financial	47.10	**47.18 ±**	**49.76**	49.76 ±	53.96	**54.02 ±**	**66.38**	66.28 ±	71.44	**71.49 ±**
busybox_1_2_8_0	**73.81**	69.64 •	**85.90**	84.03 •	**94.86**	94.04 •	99.72	**99.83** ◦	99.90	**99.93** ◦
mpc50	**75.31**	75.02 •	**84.51**	84.24 ±	92.08	**93.05** ◦	**99.55**	99.54 ±	**99.83**	99.82 ±
ref4955	**75.50**	75.03 ±	**84.54**	84.34 ±	92.27	**92.77 ±**	**99.54**	99.53 ±	**99.82**	99.82 ±
Linux	74.77	**74.79 ±**	**84.10**	83.87 ±	91.67	**91.90 ±**	**99.42**	99.41 ±	**99.79**	99.78 ±
csb281	**74.87**	74.20 •	**84.05**	83.86 ±	91.60	**91.62 ±**	**99.44**	99.44 ±	99.78	**99.79 ±**
ecos-icse11	**74.67**	74.28 •	**84.01**	83.74 ±	**91.87**	91.54 ±	**99.48**	99.48 ±	**99.81**	99.80 ±

表 1 dSATNS 和 SATNS 两算法覆盖率的比较 (%)(续)

FM	*N*=4		*N*=6		*N*=10		*N*=50		*N*=100	
	dSATNS	SATNS	dSATNS	SATNS	dSATNS	SATNS	dSATNS	SATNS	dSATNS	SATNS
ebsa285	**74.52**	74.18 ±	**83.87**	83.58 ±	**91.58**	91.29 ±	99.41	**99.42** ±	**99.78**	99.76 ±
vrc4373	**75.00**	74.70 ±	**84.36**	84.17 ±	**91.93**	91.70 ±	**99.54**	99.51 •	99.81	**99.81** ±
pati	**74.99**	74.73 •	**84.52**	84.27 •	**92.96**	92.85 ±	**99.54**	99.53 ±	**99.82**	99.82 ±
dreamcast	**75.02**	74.81 ±	**84.21**	83.99 ±	91.97	**92.42** ±	**99.44**	99.44 ±	**99.79**	99.78 ±
pc i82544	74.30	**74.48** ±	**83.96**	83.85 ±	**91.66**	91.39 ±	**99.45**	99.44 ±	99.79	**99.80** ±
XSEngine	**74.64**	74.39 •	**84.04**	83.99 ±	**91.91**	91.73 ±	99.50	**99.50** ±	99.81	**99.81** ±
refidt334	**74.12**	73.91 ±	**84.03**	83.75 •	**91.45**	91.19 •	99.43	**99.45** ±	99.78	**99.78** ±
ocelot	**74.95**	74.70 ±	**84.34**	84.17 ±	**91.80**	91.65 ±	**99.48**	99.46 ±	**99.81**	99.80 •
integrator arm9	74.32	**74.49** ±	**83.93**	83.29 •	**91.30**	91.14 ±	**99.39**	99.39 ±	**99.78**	99.76 ±
olpcl2294	74.45	**74.50** ±	**84.09**	83.9 ±	91.86	**92.26** ±	**99.47**	99.44 ±	**99.80**	99.79 ±
olpce2294	**74.63**	74.51 ±	**84.09**	83.85 •	**92.61**	91.76 •	**99.50**	99.44 •	**99.80**	99.79 ±
phycore.	74.62	**74.72** ±	**84.39**	84.12 •	**91.96**	91.73 ±	**99.53**	99.51 ±	**99.81**	99.81 ±
hs7729pci	**74.18**	74.18 ±	**83.68**	83.48 ±	**91.39**	91.33 ±	**99.26**	99.24 ±	99.70	**99.70** ±
freebsd-icse11	**71.55**	68.45 •	**81.53**	79.26 •	**90.73**	89.52 •	98.83	**98.94** ◦	99.43	**99.45** ±
fiasco	**82.73**	82.43 •	86.29	**86.41** ±	**90.62**	90.62 ±	**95.37**	95.28 ±	96.38	**96.47** ±
uClinux	**94.00**	92.06 •	**97.47**	96.57 •	**99.53**	99.29 •	**100.00**	100.00 ±	**100.00**	100.00 ±
Automotive01	**62.79**	62.62 ±	**70.47**	70.21 •	78.32	**78.39** ±	93.46	**93.47** ±	96.51	**96.54** ±
SPLOT-FM-5000	**62.61**	61.08 •	**71.03**	70.05 •	**79.93**	79.53 •	95.89	**95.91** ±	98.23	**98.25** ±
busybox-1.18.0	**89.84**	87.97 •	**94.41**	93.37 •	**97.97**	97.63 •	**99.97**	99.96 •	**99.99**	99.99 ±
2.6.28.6-icse11	69.46	**70.35** ◦	79.79	**80.58** ◦	89.70	**89.82** ◦	**98.35**	98.35 ±	**99.21**	99.20 ±
uClinux-config	**89.32**	87.07 •	**93.72**	92.57 •	**97.71**	97.41 •	99.95	**99.96** ◦	99.99	**99.99** ±
buildroot	**84.73**	84.72 ±	**91.09**	90.93 ±	95.33	**95.35** ±	99.63	**99.65** ±	**99.83**	99.83 ±
假设检验符号	•/◦/±	20/2/28	•/◦/±	22/1/27	•/◦/±	14/2/34	•/◦/±	3/3/44	•/◦/±	2/2/46

注: 粗体标注最优结果, •、◦、±分别表示第1个算法显著地优于、差于和等同于第2个算法

5.2 性能指标

本文采用 *t*-组合覆盖率和缺陷检测率作为性能指标, 其中 *t*-组合覆盖率的定义见公式 (2). 为计算该指标, 我们需要知道特征模型的所有有效 *t*-集合. 本文主要考虑 *t*=2 的情形, 即 2-组合覆盖率, 这是因为它是一个有关软件产品线测试的重要性能指标[3], 且与 *t*-组合覆盖率 ($t \geqslant 3$) 高度正相关[34,43]. 对于小规模特征模型, 我们首先枚举出所有可能的 2-集合, 然后调用 SAT4J 求解器判断这些 2-集合是否有效, 进而构造有效 *t*-集合的集合. 对于中等和大规模的特征模型, 采用枚举法计算所有有效 2-集合极为费时, 甚至不现实. 在这种情况下, 按照文献 [33] 的建议, 我们随机地产生 10 个规模为 100 000 的有效 *t*-集合的集合. 构造每个这样的集合的流程如下: 首先随机地从集合 X 选择 2 个特征, 然后对所选的每个特征, 随机地决定其是否被选中. 接着, 运用 SAT4J 求解器判断上述特征组合是否有效. 重复上述过程直到产生 100 000 个有效 2-集合. 最后, 值得说明的是, 若 2-集合的集合是随机产生的, 则某个测试集的覆盖率是这 10 个集合上覆盖率的均值. 这样做的目的是降低随机性带来的影响[33].

第 2 个指标是缺陷检测率, 定义为测试集所发现的缺陷占所有缺陷的比例. 这里的缺陷特指由特征之间的交互而触发的缺陷[44]. 本文按照 Al-Hajjaji 等人[2]所描述的方法为每个特征模型植入缺陷. 首先, 随机地在区间 [2,6] 选择一个整数作为 *t* 的值; 然后, 随机地挑选 *t* 个特征并随机地决定各特征是否出现; 接着, 运用 SAT4J 求解器判断上述特征组合是否有效. 注意我们仅接受有效的特征组合, 认为它们是构成能触发错误的缺陷. 本质上, 缺陷也是 *t*-集合, 只不过这里的 *t* 是随机取值的. 关于缺陷数, 根据 Al-Hajjaji 等人[2]的建议, 一般可取值 *n*/10. 某个测试集的至少一个测试用例覆盖了某个缺陷, 我们就认为该测试集发现了这个缺陷. 为降低随机性带来的影响, 本文为每个特征模型产生 100 个随机缺陷集, 然后取所有缺陷集上缺陷检测率的平均值作为最终的缺陷检测率. 注意以上构造及计算缺陷检测率的方法被广泛应用于既往工作[2,33].

5.3 实验设置

本文的主要实验设置如下.

• 测试集规模 N: 将 N 取 4, 6, 10, 50 和 100 等 5 个值. 一方面, 上述设置涵盖了小、中和大规模的测试集规模, 使得本文的实验和结论更具说服力; 另一方面, 该设置有助于探究各算法性能随着 N 的变化趋势.

• 算法的终止条件: 采用最大运行时间作为算法的终止条件. 与文献 [33] 的设置保持一致, 对于小规模软件产品线, 最大运行时间为 6 s; 中等规模软件产品线, 最大运行时间为 30 s; 大规模软件产品线, 最大运行时间为 200 s.

• 参数 P_r: 对 RQ1–RQ3, 其取值固定为 0.1, 与既往工作[33]保持一致. 对 RQ4, 本文将该参数取不同值以探讨它们对算法性能的影响.

• 软硬件环境: 本文的实验在个人台式机上进行, 其硬件配置为: Intel i7-7700 处理器, 3.60 GHz 频率, 8.00 GB 内存, 8 线程; 软件配置为: Windows 10 操作系统, Java 编程语言, Eclipse 集成开发环境.

• 独立运行次数: 由于本文算法及对比算法均是随机算法, 故本文将各算法独立运行 30 次, 采用性能指标的中位数作为分析与比较的标准.

5.4 假设检验

采用 Mann-Whitney U 检验 (简称 U 检验)[45]判断两个算法之间是否有显著差异 (显著性水平 $\alpha = 0.05$). 假设检验结果由 3 个符号•, ◦和±标注, 分别表示第 1 个算法显著地优于、差于和等同于第 2 个算法 (通常为第 1 个算法的变体或其他对比算法).

采用 Vargha 等人的 $\hat{A}_{12}$ 统计量[46]判断两个算法之间差异程度的大小. $\hat{A}_{12} \in [0,1]$ 统计量是一种非参数效应量 (effect size) 度量方法. $\hat{A}_{12}$ 度量了第 1 个算法较第 2 个算法获得更高指标值的概率. 具体地, 若 $\hat{A}_{12} > 0.5$, 则表示第 1 个算法比第 2 个算法具有更高的概率获得更优指标值 (用符号↑表示); $\hat{A}_{12} = 0.5$ 表示两算法是相当的 (用符号～表示); $\hat{A}_{12} < 0.5$ 表示第 1 个算法比第 2 个算法具有更低的概率获得更优指标值 (用符号↓表示). 根据 Vargha 等人[46], 效应量的幅度可定性地评估为 large (l), medium (m), small (s) 和 negligible (n). 本文采用 R 平台的 effsize 软件包计算 $\hat{A}_{12}$ 统计量.

考虑所有的特征模型, 采用 Friedman 检验[47]计算各对比算法的平均排名. 该检验法首先在单个特征模型上独立地对各算法进行排序, 然后计算这些算法在所有特征模型上的平均排名. 显然, 排名越小表示算法越优. 事实上, Friedman 检验非常适合用于评估与比较算法的整体性能表现.

5.5 实验结果与分析

本节将展示 4 个研究问题的实验结果, 并对结果进行分析与解释, 最后给出各研究问题的答案.

5.5.1 采用多样性求解器有助于改善覆盖率和缺陷检测率 (RQ1)

表 1 给出了 dSATNS 与 SATNS 两个算法的覆盖率 (%) 以及成对比较的 U 检验结果. 如表 1 所示, 对所有 N 值而言, dSATNS 在大多数特征模型的覆盖率是高于 SATNS 的. 具体地, 图 2 (a) 给出了 dSATNS 显著优于 (•)、差于 (◦) 和等同于 (±) SATNS 的特征模型所占百分比. 例如, 当 N=4 时, dSATNS 在 40% 的模型上显著优于 SATNS. 从图 2 (a) 可看出, 当 N 较小时 ($N \leqslant 10$), dSATNS 较 SATNS 具有明显优势: 前者表现更优的特征模型百分比远高于后者表现更优的特征模型百分比. 当 N 较大时 (N=50 和 N=100), 二者的性能差异不明显. 在绝大多数 (超过 88%) 的特征模型上, dSATNS 和 SATNS 的覆盖率不具有统计意义上的差异. 由此可见, 当 N 较小时, 采用多样化的 ProbSAT 求解器能显著提升算法的覆盖率.

类似地, dSATNS 与 SATNS 的缺陷检测率以及 U 检验结果汇总在表 2 和图 2(b). 如图 2(b) 所示, 虽然在超过 66% 的特征模型上, 两算法的缺陷检测率并无显著差异. 但是, 不论 N 取何值, dSATNS 表现更优的特征模型所占百分比始终高于其表现更差的特征模型所占百分比, 且相应百分比的差异最高可达 9 倍 (即 N=10 的情形). 因此, 我们可以认为: dSATNS 较 SATNS 发现更多软件缺陷的潜力更大.

进一步地, 为比较 dSATNS 和 SATNS 两个算法差异程度的大小, 表 3 按照效应量幅度汇总了关于覆盖率和缺陷检测率的 $\hat{A}_{12}$ 比较结果. 例如, 表 3 中第 2 行第 3 列的数字“28%”表示: 在 28% 的特征模型上, dSATNS 就覆

盖率的表现优于 SATNS(↑), 且二者的差异幅度被评估为 large (*l*). 从表 3 可看出, 对覆盖率指标和 *N*=4, 6 和 10 而言, 在至少 60% 的特征模型上 dSATNS 与 SATNS 具有不可忽视的差异. 此外, 无论差异幅度是 *l*, *m* 还是 *s*, dSATNS 表现更优的模型所占百分比始终 (远) 高于其表现更差的模型所占百分比. 当 *N*=50 和 100 时, 在大多数模型上 (60%+), 两算法的差异是可忽略的. 考虑缺陷检测率, dSATNS 表现更优的模型所占百分比始终高于 SATNS 表现更优的模型所占百分比. 由此可见, 在更多情形下, dSATNS 改进了 SATNS, 而且这种改进在相当比例的特征模型上是本质性的 (即效应量幅度为 *l* 和 *m*).

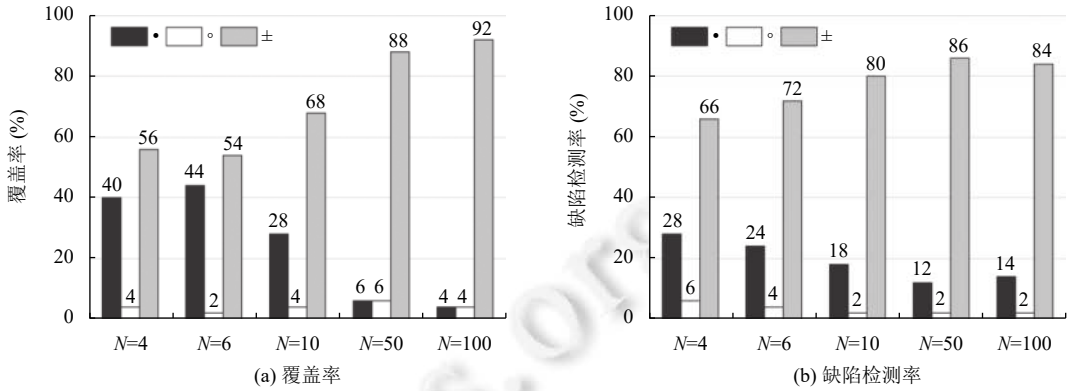


图 2 根据 U 检验, dSATNS 显著优于 (•)、差于 (◦) 和等同于 (±)SATNS 的特征模型所占百分比

表 2 dSATNS 和 SATNS 两算法缺陷检测率的比较 (%)

FM	*N*=4		*N*=6		*N*=10		*N*=50		*N*=100	
	dSATNS	SATNS	dSATNS	SATNS	dSATNS	SATNS	dSATNS	SATNS	dSATNS	SATNS
CounterStrikeSFM	66.33	**66.50** ±	73.67	**75.00** ◦	83.33	**84.00** ±	97.33	**97.83** ±	**99.33**	99.33 ±
HiPAcc	55.50	**55.75** ◦	**65.50**	64.38 ±	**75.00**	74.75 ±	**93.38**	93.13 ±	96.63	**96.75** ±
SPLSSimuelESPnP	**64.75**	64.25 ±	**74.25**	73.25 ±	**84.00**	83.75 ±	**98.50**	98.25 ±	**99.75**	99.75 ±
JavaGC	54.75	**55.75** ±	62.75	**63.25** ±	**71.75**	71.13 ±	**90.63**	90.50 ±	**95.00**	94.88 ±
Polly	**57.70**	57.60 ±	**65.80**	65.60 ±	**75.50**	75.30 ±	**93.50**	93.20 ±	**96.80**	96.60 ±
DSSample	47.90	**48.40** ±	**55.20**	55.00 ±	**63.70**	63.50 ±	**86.60**	86.60 ±	**91.80**	91.80 ±
VP9	54.50	**54.70** ±	**63.20**	62.80 ±	**71.90**	71.40 ±	**91.60**	91.40 ±	95.80	**96.00** ±
WebPortal	**58.50**	58.40 ±	**67.10**	66.70 ±	**76.80**	76.40 ±	**94.60**	94.20 ±	97.20	**97.30** ±
JHipster	62.50	**62.70** ±	**70.60**	70.40 ±	79.40	**79.70** ±	**96.00**	95.80 ±	**98.20**	98.00 ±
Drupal	**56.80**	56.20 ±	**65.00**	64.40 ±	**74.30**	74.30 ±	**94.50**	94.00 •	**97.90**	97.80 ±
SmartHomev2.2	**54.21**	53.64 •	**62.57**	61.86 •	**72.36**	71.79 ±	**93.14**	92.79 ±	97.21	**97.29** ±
VideoPlayer	**62.63**	62.13 ±	70.13	**70.19** ±	**79.50**	79.13 ±	**96.63**	96.44 ±	**98.94**	98.88 ±
Amazon	47.25	**47.38** ±	52.94	**53.31** ±	60.19	**60.25** ±	**80.94**	80.63 ±	86.81	**86.94** ±
ModelTransformation	**53.17**	53.00 ±	**61.94**	61.50 ±	**71.33**	71.22 ±	92.33	**92.44** ±	**96.44**	96.44 ±
CocheEcologico	**65.45**	65.45 ±	71.95	**72.15** ±	79.20	**79.35** ±	93.60	**93.70** ±	**96.35**	96.30 ±
Printers	**59.89**	59.03 •	**66.03**	65.69 ±	**73.31**	73.22 ±	**89.89**	89.83 ±	93.81	**93.89** ±
fiasco_17_10	**54.29**	53.85 •	**59.38**	59.35 ±	**66.81**	66.67 ±	80.08	**80.23** ±	83.50	**83.81** ◦
uClibc-ng_1_0_29	**44.28**	43.83 •	**50.07**	49.52 •	**56.80**	56.30 ±	**74.78**	74.74 ±	80.70	**81.04** ±
E-shop	**50.45**	49.53 •	**58.50**	57.30 •	67.75	**67.82** ±	**90.68**	90.23 •	**95.58**	95.57 ±
toybox	**90.10**	89.52 •	**93.96**	93.64 •	96.76	**96.79** ±	**99.82**	99.82 ±	**99.96**	99.96 ±
axTLS	**83.24**	83.14 ±	**88.67**	88.49 ±	**93.38**	93.26 ±	**99.23**	99.20 ±	**99.77**	99.76 ±
financial	46.21	**46.41** ◦	49.14	**49.23** ±	**53.51**	53.43 ±	**65.72**	65.69 ±	**71.19**	71.17 ±
busybox_1_28_0	**43.90**	42.48 •	**51.45**	50.65 •	**61.16**	60.42 •	**86.90**	86.57 •	**93.57**	93.33 •
mpc50	51.12	**51.14** ±	**58.28**	58.16 ±	66.89	**67.54** ◦	**88.12**	88.03 ±	**93.13**	92.97 ±
ref4955	50.69	**50.73** ±	**57.95**	57.82 ±	67.01	**67.04** ±	**87.61**	87.61 ±	**92.88**	92.83 ±
Linux	51.40	**51.51** ±	**58.52**	58.31 ±	66.82	**67.17** ±	87.43	**87.46** ±	**92.52**	92.51 ±

表 2 dSATNS 和 SATNS 两算法缺陷检测率的比较 (%)(续)

FM	*N*=4		*N*=6		*N*=10		*N*=50		*N*=100	
	dSATNS	SATNS	dSATNS	SATNS	dSATNS	SATNS	dSATNS	SATNS	dSATNS	SATNS
csb281	**49.49**	49.33 ±	**56.75**	56.61 ±	**65.14**	65.12 ±	86.75	**86.80 ±**	**92.24**	92.20 ±
ecos-icse11	**49.68**	49.49 ±	56.75	**56.78 ±**	**65.64**	65.38 ±	**86.76**	86.70 ±	**92.24**	92.15 ±
ebsa285	48.90	**48.98 ±**	**56.08**	55.92 ±	**64.62**	64.43 ±	**86.34**	86.34 ±	**91.95**	91.88 ±
vrc4373	**51.07**	50.80 ±	58.07	**58.08 ±**	**66.64**	66.35 •	**87.63**	87.60 ±	92.81	**92.86 ±**
pati	**50.96**	50.81 ±	**58.05**	57.82 •	**66.89**	66.73 ±	**87.60**	87.56 ±	**92.73**	92.65 ±
dreamcast	**50.77**	50.63 ±	**57.95**	57.89 ±	66.57	**66.83 ±**	**87.54**	87.49 ±	**92.71**	92.70 ±
pc i82544	50.59	**50.62 ±**	57.69	**57.71 ±**	**66.30**	65.94 •	**87.42**	87.38 ±	**92.60**	92.53 ±
XSEngine	**49.69**	49.39 •	**56.78**	56.65 ±	**65.54**	65.38 ±	**86.94**	86.93 ±	92.35	**92.41 ±**
refidt334	**50.27**	50.05 ±	**57.43**	57.34 ±	**65.91**	65.63 •	**87.00**	86.79 •	**92.21**	92.20 ±
ocelot	51.08	**51.09 ±**	**58.24**	58.11 ±	**66.51**	66.51 ±	87.30	**87.35 ±**	**92.56**	92.51 •
integrator arm9	49.84	**50.03 ±**	**57.04**	56.77 •	**65.34**	65.22 ±	**86.63**	86.60 ±	**92.01**	91.94 ±
olpcl2294	49.99	**50.14 ±**	**57.17**	57.05 ±	**65.90**	65.77 ±	**86.93**	86.89 ±	**92.29**	92.19 •
olpce2294	**50.39**	50.34 ±	**57.45**	57.31 ±	**66.14**	66.04 ±	**86.88**	86.80 ±	**92.17**	92.17 ±
phycore.	50.63	**50.75 ±**	**57.96**	57.81 ±	**66.38**	66.11 ±	**87.29**	87.25 ±	**92.60**	92.55 ±
hs7729pci	50.24	**50.35 ±**	**57.41**	57.26 ±	**65.89**	65.83 ±	**86.48**	86.44 ±	**91.88**	91.78 •
freebsd-icse11	**44.43**	43.01 •	**51.50**	50.30 •	**60.14**	59.45 •	84.16	**84.33** ◦	**90.91**	90.91 ±
fiasco	**81.18**	80.90 •	85.10	**85.17 ±**	**89.77**	89.77 ±	**95.20**	95.12 ±	96.29	**96.41 ±**
uClinux	**87.98**	86.17 •	**92.23**	91.40 •	**96.01**	95.67 •	**99.79**	99.78 ±	99.97	**99.97 ±**
Automotive01	**44.13**	44.06 ±	**50.07**	49.98 ±	57.02	**57.19 ±**	76.10	**76.13 ±**	**82.40**	82.32 ±
SPLOT-FM-5000	**38.98**	38.23 •	**44.91**	44.47 •	**52.46**	52.22 •	**74.43**	74.35 •	**81.63**	81.55 ±
busybox-1.18.0	**81.36**	79.84 •	**86.55**	85.58 •	**91.84**	91.37 •	**98.86**	98.81 •	**99.61**	99.61 ±
2.6.28.6-icse11	42.11	**42.75** ◦	48.93	**49.44** ◦	58.08	**58.13 ±**	82.05	**82.11 ±**	**88.87**	88.75 •
uClinux-config	**80.07**	78.31 •	**85.23**	84.16 •	**90.87**	90.48 •	**98.58**	98.56 ±	**99.51**	99.49 •
buildroot	**72.84**	72.82 ±	**79.43**	79.28 ±	85.27	**85.32 ±**	**96.51**	96.50 ±	**98.20**	98.16 •
	•/◦/±	14/3/33	•/◦/±	12/2/36	•/◦/±	9/1/40	•/◦/±	6/1/43	•/◦/±	7/1/42

注: 粗体标注最优结果, •、◦、±分别表示第1个算法显著地优于、差于和等同于第2个算法

表 3 dSATNS 和 SATNS 的 $\hat{A}_{12}$ 比较结果汇总 (%)

覆盖率		*N*=4	*N*=6	*N*=10	*N*=50	*N*=100	缺陷检测率		*N*=4	*N*=6	*N*=10	*N*=50	*N*=100
l	↑	28	32	14	0	0	*l*	↑	16	16	12	4	4
	↓	2	2	0	4	2		↓	2	2	0	0	0
m	↑	6	8	12	4	2	*m*	↑	10	4	6	6	10
	↓	2	0	6	2	2		↓	0	0	2	2	2
s	↑	22	42	26	14	8	*s*	↑	20	30	34	24	22
	↓	4	2	2	14	22		↓	12	4	10	2	10
n	～	36	14	40	62	64	*n*	～	40	44	36	62	52

注: ↑: dSATNS更优; ↓: SATNS更优, ～: 二者性能相当

根据 dSATNS 与 SATNS 的 $\hat{A}_{12}$ 比较结果, 若忽略效应量幅度, 可分 3 种情形: 改进情形 (即 dSATNS 表现优于 SATNS), 退化情形 (即 dSATNS 表现差于 SATNS) 和持平情形 (即 dSATNS 与 SATNS 的表现相当). 将 *l*, *m* 和 *s* 这 3 种幅度所对应的百分数累加即可得到“改进情形”与“退化情形”所占的总的百分比, 而 *n* 对应的百分数即为“持平情形”所占的百分比. 图 3 以柱状图的形式给出了各种情形所占的百分比, 其中, ↑表示改进情形、↓表示退化情形、～表示持平情形. 如图 3(a) 所示, 就覆盖率而言: 当 *N* 小于等于 10 时, dSATNS“改进情形”的百分比明显高于“退化情形”的百分比, 也高于“持平情形”的百分比; 当 *N*=50 和 100 时, 虽然“退化情形”的百分比高于“改进情形”, 但是在绝大多数模型上 (62%+), 两算法的性能是持平的. 由图 3(b) 可知, 就缺陷检测率而言, 不论 *N* 取何值, “改进情

形”始终远高于“退化情形”.

从以上实验结果及分析, 可得出以下结论.

• U 检验和 $\hat{A}_{12}$ 效应量均表明, 当 N 较小时, 采用了多样化求解器的 dSATNS 算法更倾向于获得更高的覆盖率; 而当 N 较大时, 在绝大多数模型上覆盖率的差别并不显著. 上述现象可解释如下: 随着 N 的增大, 相应测试集能潜在地覆盖更多的 t-集合. 当 N 足够大时, 任何随机产生的测试集都有可能覆盖 (将近) 所有的 t-集合. 此时, 是否采用多样性求解器仅会带来很小的差异.

• U 检验和 $\hat{A}_{12}$ 效应量均表明, 无论 N 取何值, 采用了多样化求解器的 dSATNS 算法都更倾向于获得更高的缺陷检测率.

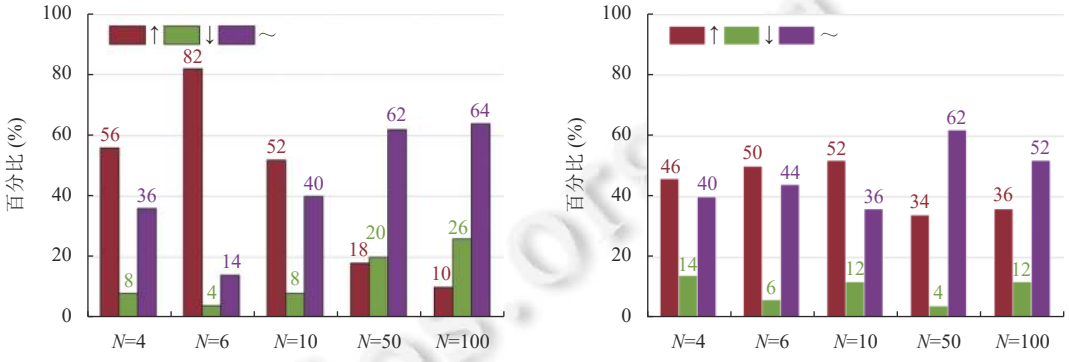


图 3 dSATNS 算法不同情形所占的百分比

因此, RQ1 的答案已明确: 在本文所提算法中, 采用多样化的 SAT 求解器的确有助于提高软件产品线测试的覆盖率和缺陷检测率.

5.5.2 采用两种归档策略的必要性 (RQ2)

图 4 给出了 TwoArch 与 GloArch 比较的 U 检验结果. 如图所示, 无论是覆盖率还是缺陷检测率, TwoArch (显著) 优于 GloArch (•) 的百分比始终大于等于 TwoArch 表现更差 (◦) 的百分比. 当 p_j 时, TwoArch 的优势非常明显; 而当 $N \geqslant 50$ 时, 这两个算法在绝大多数 (90%+) 模型上并未有显著的性能差异. 由此可见, 全局档案策略虽然可在 N 较大时取得较好的效果, 但当 N 较小时, 其性能具有很大的改善空间. 换言之, 采用两种档案策略一方面保持了全局档案在 N 较大时的优良性能, 另一方面又有效规避了该档案在 N 较小时的性能退化.

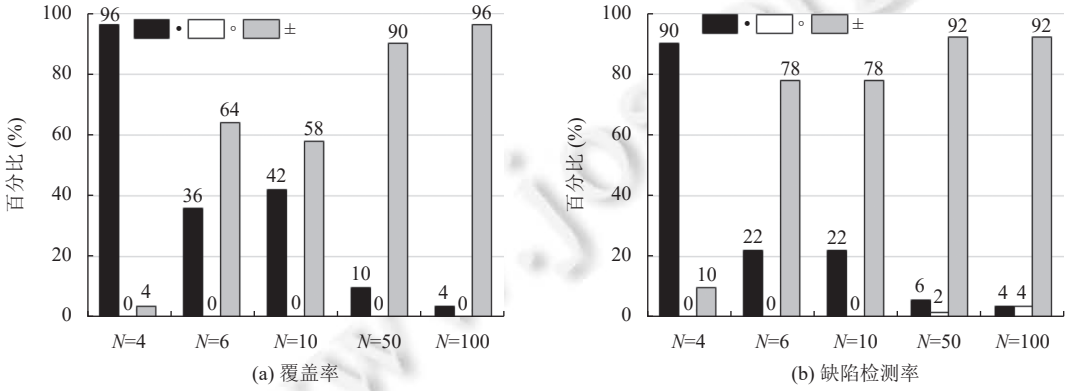


图 4 TwoArch 与 GloArch 比较的 U 检验结果

类似地, 图 5 给出了 TwoArch 与 LocArch 比较的 U 检验结果. 与图 4 正好相反, 当 $N \leqslant 10$ 时, TwoArch 的整体性能表现稍逊于 LocArch; 但当 $N \geqslant 50$ 时, TwoArch 整体要优于 LocArch. 由此可见, 局部档案策略对较小的 N

有效, 但当 N 较大时, 其性能可进一步改善. 换言之, 当 N 较小时, 采用两种档案策略虽然会带来轻微的性能退化, 但是当 N 较大时能较为明显地弥补局部档案的不足.

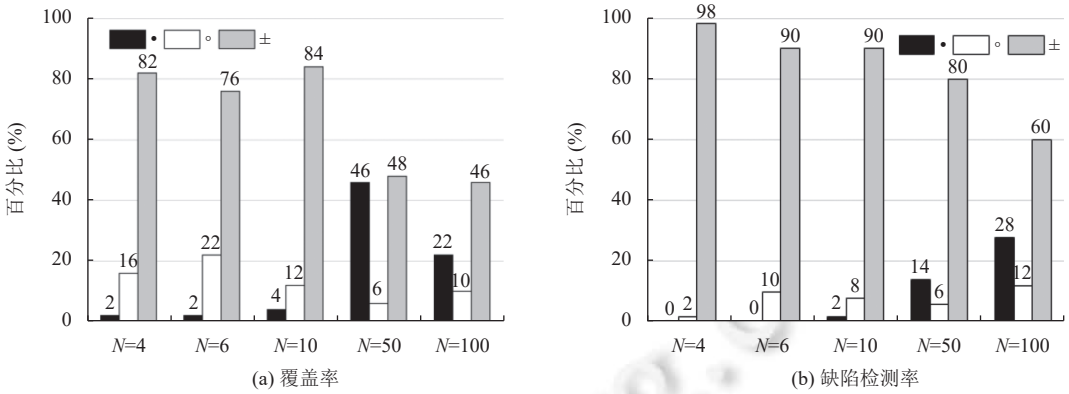


图 5 TwoArch 与 LocArch 比较的 U 检验结果

表 4 给出了 TwoArch 与 GloArch 关于覆盖率及缺陷检测率的 $\hat{A}_{12}$ 比较结果. 为便于对结果进行分析, 图 6 展示了这两个算法进行比较时“改进情形”与“退化情形”所占百分比随 N 的变化趋势. 如图 6 所示, 与 GloArch 相比, TwoArch 覆盖率和缺陷检测率的“改进情形”所占百分比随着 N 的增大呈下降趋势. 相反地, “退化情形”则呈上升趋势. 同时, 由表 4 知, “改进情形”所占百分比最高可达 96%, 而“退化情形”最高仅为 22%. 以上结果表明, 当 N 越小时, 两个归档策略较全局归档策略的优势越明显.

表 4 TwoArch 与 GloArch 的 $\hat{A}_{12}$ 比较结果汇总 (%)

覆盖率		N=4	N=6	N=10	N=50	N=100	缺陷检测率		N=4	N=6	N=10	N=50	N=100
l	↑	94	12	6	0	2	l	↑	78	2	2	0	2
	↓	0	0	0	0	0		↓	0	0	0	0	0
m	↑	0	14	26	6	2	m	↑	8	18	14	4	0
	↓	0	0	0	0	0		↓	0	0	0	0	4
s	↑	2	38	34	16	6	s	↑	10	20	36	14	12
	↓	0	2	2	8	22		↓	2	10	14	12	12
汇总	↑	96	64	66	22	10	汇总	↑	96	40	52	18	14
	↓	0	2	2	8	22		↓	2	10	14	12	16
	～	4	34	32	70	68		～	2	50	34	70	70

注: ↑: TwoArch更优; ↓: GloArch更优, ～: 二者性能相当

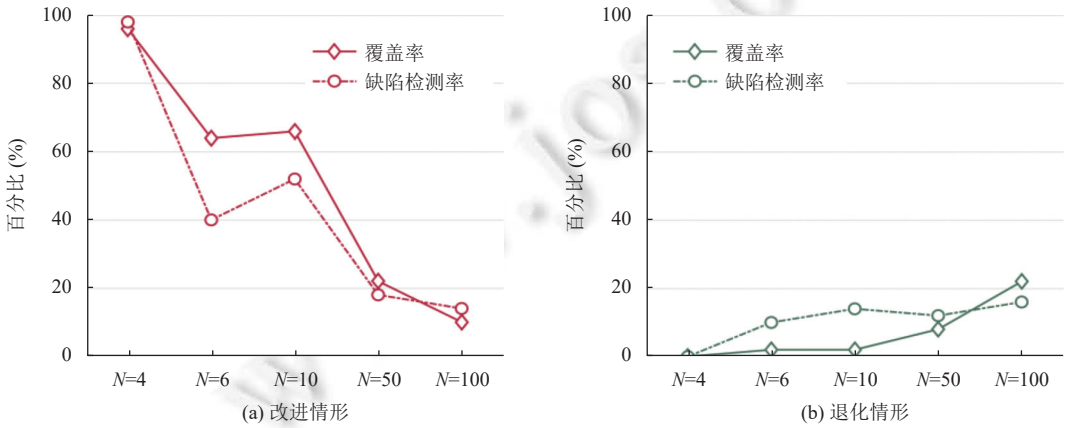


图 6 TwoArch 较 GloArch 的“改进情形”与“退化情形”随 N 的变化趋势

类似地, 表 5 给出了 TwoArch 与 LocArch 关于覆盖率及缺陷检测率的 $\hat{A}_{12}$ 比较结果, 图 7 则展示了相应的改进和退化情形随 N 的变化趋势. 与图 6 的趋势正好相反, “改进情形”整体呈上升趋势, 而“退化情形”则大体呈下降趋势. 由此可知, 当 N 较大时, 两个归档策略较局部档案策略的优势更明显. 值得一提的是, 虽然表 5 的数据表明, 当 N 较小时 TwoArch 较 LocArch 的“退化情形”所占百分比远高于“改进情形”, 但是这些差异的主要贡献来源于 s 幅度的效应量.

表 5 TwoArch 与 LocArch 的 $\hat{A}_{12}$ 比较结果汇总 (%)

覆盖率		N=4	N=6	N=10	N=50	N=100	缺陷检测率		N=4	N=6	N=10	N=50	N=100
l	↑	0	0	2	38	42	l	↑	0	0	2	10	16
	↓	0	2	2	6	10		↓	0	2	0	6	8
m	↑	0	2	0	8	2	m	↑	0	0	0	2	8
	↓	6	14	2	0	0		↓	2	2	4	0	2
s	↑	8	6	12	8	4	s	↑	2	4	16	20	30
	↓	32	38	32	2	12		↓	26	38	14	14	8
汇总	↑	8	8	14	54	48	汇总	↑	2	4	18	32	54
	↓	38	54	36	8	22		↓	28	42	18	20	18
	～	54	38	50	38	30		～	70	54	64	48	28

注: ↑: TwoArch更优; ↓: LocArch更优, ～: 二者性能相当

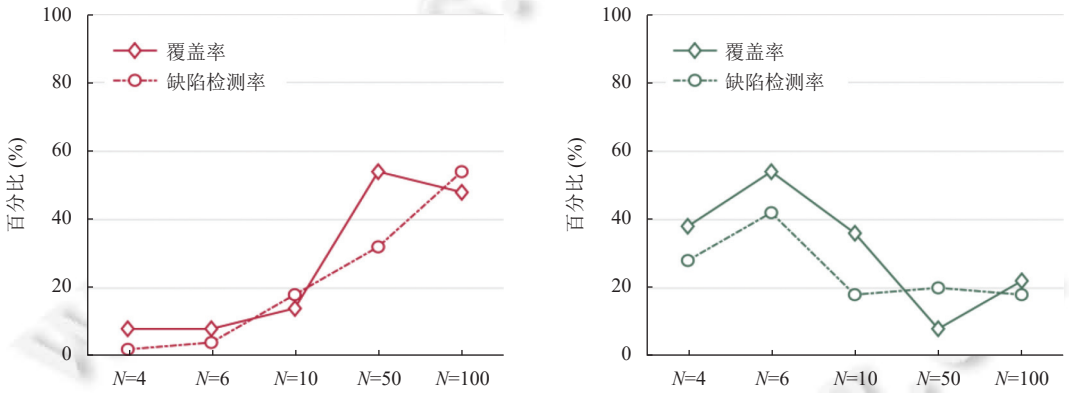


图 7 TwoArch 较 LocArch 的“改进情形”与“退化情形”随 N 的变化趋势

从以上实验结果及讨论得出以下结论.

• U 检验和 $\hat{A}_{12}$ 效应量的比较结果均表明, 仅采用全局档案策略适合 N 较大的情形, 而当 N 较小时, 其性能有待进一步提升. 相反地, 仅采用局部档案策略适合 N 较小的情形.

• 同时采用两种档案策略可获得折衷效果, 即无论 N 取大或小, 其性能都不至于太差.

因此, RQ2 的答案也明确了: 在本文所提算法中, 采用两种归档策略是非常有必要的, 它能折衷仅使用全局或局部归档策略的效果, 进而使得测试效果对测试集规模大小不敏感.

5.5.3 与主流算法的比较 (RQ3)

图 8 给出了 dSATNS 与各主流算法关于缺陷检测率的 U 检验结果. 这里仅给出缺陷检测率的实验结果, 主要原因在于覆盖率和缺陷检测率得出的结论相似. 如图 8(a) 和图 8(b) 所示, 无论 N 取何值, dSATNS 始终优于 TSEGA 和 TSENS. 而且, 当 N 较小时, dSATNS 的优势非常明显: dSATNS 表现显著更优的特征模型百分比远高于其表现显著更差的特征模型百分比. 当 N 较大时 (如 N=50 和 100), 在 80% 以上的特征模型上, dSATNS 与其对比算法并无显著的统计差异. 与 SAT4J 相比, 图 8(c) 表明, 无论 N 的取值, dSATNS 始终具有压倒性优势: 在 90% 及以上的特征模型上, dSATNS 较 SAT4J 的性能改进是统计显著的.

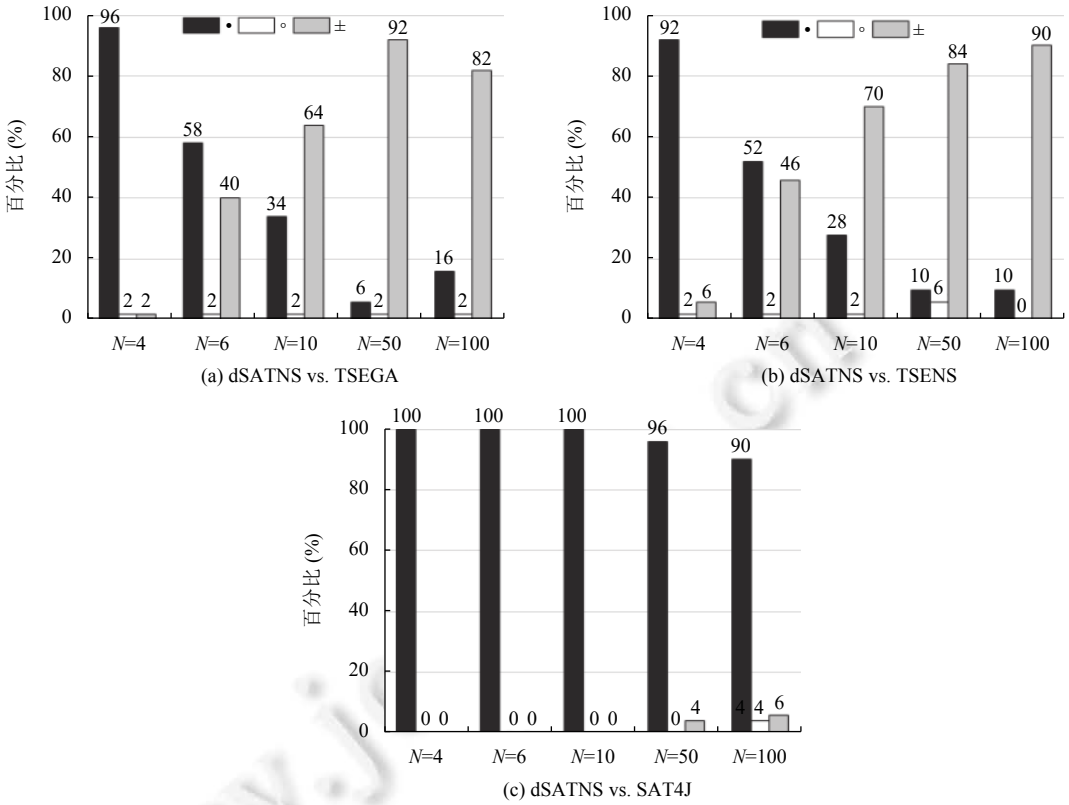


图 8 dSATNS 与主流算法缺陷检测率的 U 检验结果

为对各算法的性能有更直观的认识, 图 9 展示了由 Friedman 检验得到的各算法的平均排名 (越小越好). 可以看到, 对所有 N 值而言, dSATNS 始终获得最佳的平均排名, 其次是 TSEGA 或 TSENS (二者的性能差别很小), 最后是 SAT4J. 进一步地, 表 6 给出了 dSATNS 与各算法成对比较的 p 值 (Friedman 检验). 除了 N=50 外, dSATNS 与 TSEGA 的差异在其他情形均是统计显著的. 与 TSENS 相比, 统计不显著的情形还包括 N=100. 最后, dSATNS 与 SAT4J 的差异对所有 N 值而言均是统计显著的. 以上实验结果说明, 与主流算法相比, dSATNS 不仅获得了最佳排名, 而且与其他算法的性能差异在多数情形下是统计显著的. 特别地, 当 N 较小时, dSATNS 展现出卓越的性能表现, 这对于该算法的实际应用是非常有利的. 如前所述, 软件测试人员的时间、精力及成本往往都是有限的, 这使得他们不断寻求既能减小测试集规模, 又能提高测试效果的方法. 在软件产品线测试领域, 本文所提出的算法为实现上述目标提供了行之有效的解决方案.

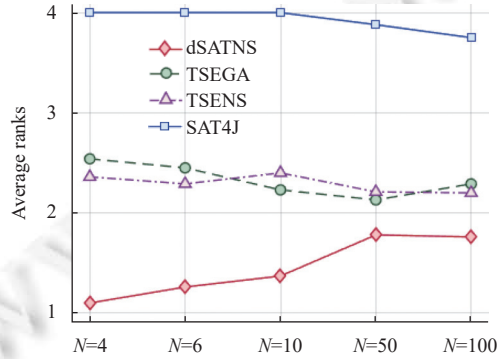


图 9 Friedman 检验得到的各算法的平均排名 (average ranks)

表 6 dSATNS 与主流算法比较的 p 值 (Friedman 检验). 加粗表示无显著差异 (显著性水平 $\alpha = 0.05$)

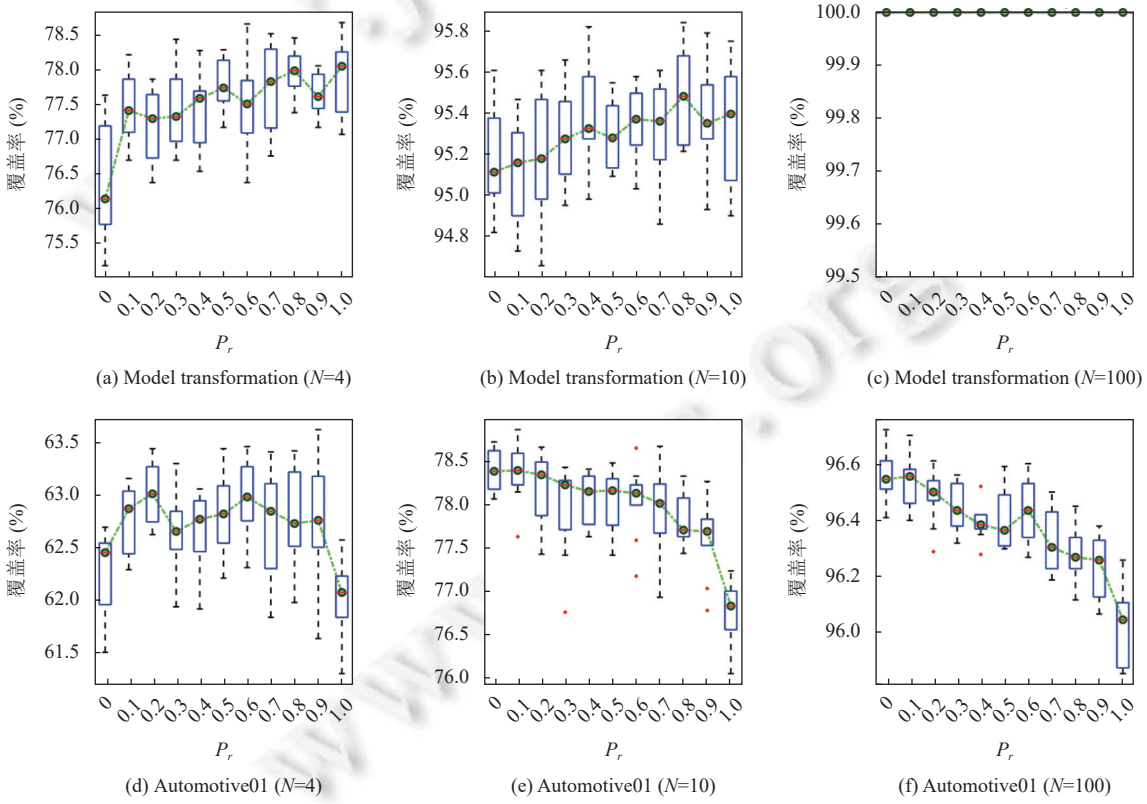
对比算法	N=4	N=6	N=10	N=50	N=100
dSATNS vs. TSEGA	≈0	0.000 004	0.000 066	**0.095 836**	0.040 104
dSATNS vs. TSENS	0.000 001	0.000 066	0.000 866	**0.175 244**	**0.088 36**
dSATNS vs. SAT4J	≈0	≈0	≈0	≈0	≈0

RQ3 的答案如下: 与主流算法相比, 本文提出的 dSATNS 取得了最优的整体性能表现. 尤其是, 当测试集规模较小时, 本文算法具有明显优势, 故非常适合于测试资源有限的情形.

5.5.4 参数 P_r 对算法性能的影响 (RQ4)

以 0.1 为步长, 将 P_r 的值从 0.0 增加到 1.0. 对 P_r 的每个值, 考虑 N=4, 10 和 100 这 3 种情形, 在 ModelTransformation, Automotive01 和 2.6.28.6-icse11 这 3 个特征模型上独立运行 dSATNS 算法 30 次. 注意以上模型分别可视为小规模、中等规模和大规模特征模型的代表. 实验结果 (即覆盖率) 以箱线图的形式给出, 见图 10. 如图所示, 对小规模的 ModelTransformation: 当 N=4 时, P_r 取 0.0 的效果最差; 当 N=10 时, 随着 P_r 的增大, 覆盖率略呈上升趋势, 但整体相差不大; 当 N=100 时, 不同 P_r 值对应的算法性能几乎无差异. 对中等规模的 Automotive01, 当 N=4 时, P_r 取 0.0 或 1.0 的效果要差于其他取值; 而当 N=10 或 100 时, 随着 P_r 的增大, 覆盖率整体呈下降趋势. 对于大规模的 2.6.28.6-icse11, 当 N=4 时, P_r 取 1.0 的效果最差, 取其他值的效果则相差不大; 当 N=10 和 100 时, P_r 的最优取值分别为 0.1 和 0.0, 且当 P_r 增大时, 算法的性能呈下降趋势. 综上, 虽然 P_r 的最优取值同时受到 N 值及待求解特征模型的影响, 但是我们可总结出以下规律.

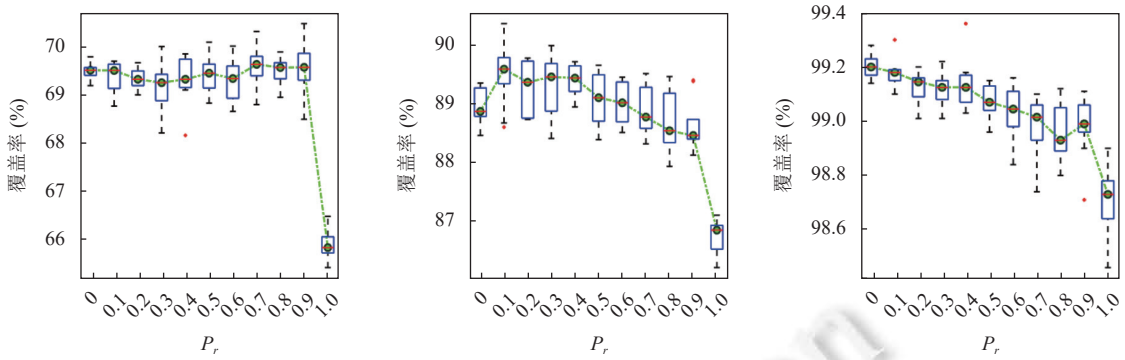
- 不建议将 P_r 取为两个极端值, 即 0.0 和 1.0. 这是因为 P_r 取这两个值时, 总会在某些情形下获得较 (最) 差效果.
- 当 P_r 从 0.1 增加到 0.9 时, 算法的性能或者差别不大, 或者呈下降趋势.
- 综合考虑上述两点, 为了折衷不同情形下的算法性能, 本文建议将 P_r 取为一个非零的较小值, 如 0.1 等.



(a) Model transformation (N=4) (b) Model transformation (N=10) (c) Model transformation (N=100)

(d) Automotive01 (N=4) (e) Automotive01 (N=10) (f) Automotive01 (N=100)

图 10 参数 P_r 对算法性能的影响



(g) 2.6.28.6-icse11 (N=4) (h) 2.6.28.6-icse11 (N=10) (i) 2.6.28.6-icse11 (N=100)

图 10 参数 P_r 对算法性能的影响 (续)

综上, RQ4 的答案如下: 参数 P_r 对 dSATNS 算法性能的影响因 N 值及待求解特征模型的不同而有所不同. 一般地, P_r 取较小值 (如 0.1) 时效果较好. 这与既往工作[33]的结论是一致的.

6 结 论

本文研究基于相似性的软件产品线测试方法, 提出了一种基于多样性 SAT 求解器和新颖搜索的算法. 该算法的目的在于产生并维护一组多样化的测试用例, 进而覆盖更多的特征组合、发现更多的软件缺陷. 为了生成多样化的测试用例, 同时采用了两类 (CDCL 和 SLS) 多样性 SAT 求解器. 实验结果表明, 多样性求解器的运用确实有助于提升多样性, 进而提高覆盖率和缺陷检测率. 为维护测试集的多样性, 同时采用了全局和局部归档策略, 它们均以 NS 算法的思想为基础. 其中, 局部归档策略是本文提出的一种新方法. 实验结果揭示, 全局和局部档案策略分别适用于测试集规模较大和较小的情形, 而两个档案策略的同时运用则能获得折衷效果. 最后, 对比实验表明本文所提算法的整体性能优于其他主流算法. 特别是当测试集规模较小时, 本文算法具有明显优势.

本文所建议的多样性 SAT 求解框架 (即算法 2) 是通用的. 现在所采用的 SAT 求解器完全可以直接替换成任何其他求解器, 如 Glucose[48], NLocalSAT[49], LS-Sampling[34], SamplingCA[50]等. 在将来, 比较这些求解器的性能表现并给出关于如何选择求解器的合理建议是一项有意义的工作. 在多样性 SLS 求解器的设计方面, 将概率向量与 ProbSAT 所采用的随机机制深度结合是一个值得尝试的方向. 最后, 我们也将探索把本文所采用的双归档策略应用于与软件产品线相关的其他任务, 如多样性采样[51]等.

References:

[1] Clements PC, Northrop LM. Software Product Lines: Practices and Patterns. Boston: Addison-Wesley Professional, 2001. 1–10.

[2] Al-Hajjaji M, Thüm T, Lochau M, Meinicke J, Saake G. Effective product-line testing using similarity-based product prioritization. Software & Systems Modeling, 2019, 18(1): 499–521. [doi: 10.1007/s10270-016-0569-2]

[3] Hierons RM, Li MQ, Liu XH, Parejo JA, Segura S, Yao X. Many-objective test suite generation for software product lines. ACM Trans. on Software Engineering and Methodology, 2020, 29(1): 2. [doi: 10.1145/3361146]

[4] Lian XL, Zhang L. Multi-objective optimization algorithm for feature selection in software product lines. Ruan Jian Xue Bao/Journal of Software, 2017, 28(10): 2548–2563 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5130.htm [doi: 10.13328/j.cnki.jos.005130]

[5] Batory D. Feature models, grammars, and propositional formulas. In: Proc. of the 9th Int'l Conf. on Software Product Lines (SPLC 2005). Rennes: Springer, 2005. 7–20. [doi: 10.1007/11554844_3]

[6] Benavides D, Segura S, Ruiz-Cortés A. Automated analysis of feature models 20 years later: A literature review. Information Systems, 2010, 35(6): 615–636. [doi: 10.1016/j.is.2010.01.001]

[7] Al-Hajjaji M, Thüm T, Meinicke J, Lochau M, Saake G. Similarity-based prioritization in software product-line testing. In: Proc. of the 18th Int'l Software Product Line Conf. (Vol. 1). Florence: Association for Computing Machinery, 2014. 197–206. [doi: 10.1145/2648511.

2648532]

[8] Varshosaz M, Al-Hajjaji M, Thüm T, Runge T, Mousavi MR, Schaefer I. A classification of product sampling for software product lines. In: Proc. of the 22nd Int'l Systems and Software Product Line Conf. (Vol. 1). Gothenburg: Association for Computing Machinery, 2018. 1–13. [doi: 10.1145/3233027.3233035]

[9] Cohen MB, Dwyer MB, Shi JF. Constructing interaction test suites for highly-configurable systems in the presence of constraints: A greedy approach. IEEE Trans. on Software Engineering, 2008, 34(5): 633–650. [doi: 10.1109/TSE.2008.50]

[10] Johansen MF, Haugen Ø, Fleurey F. Properties of realistic feature models make combinatorial testing of product lines feasible. In: Proc. of the 14th Int'l Conf. on Model Driven Engineering Languages and Systems. Wellington: Springer, 2011. 638–652. [doi: 10.1007/978-3-642-24485-8_47]

[11] Lopez-Herrejon RE, Fischer S, Ramler R, Egyed A. A first systematic mapping study on combinatorial interaction testing for software product lines. In: Proc. of the 8th IEEE Int'l Conf. on Software Testing, Verification and Validation Workshops (ICSTW). Graz: IEEE, 2015. 1–10. [doi: 10.1109/ICSTW.2015.7107435]

[12] Kuhn DR, Wallace DR, Gallo AM. Software fault interactions and implications for software testing. IEEE Trans. on Software Engineering, 2004, 30(6): 418–421. [doi: 10.1109/TSE.2004.24]

[13] Cohen MB, Dwyer MB, Shi JF. Interaction testing of highly-configurable systems in the presence of constraints. In: Proc. of the 2007 Int'l Symp. on Software Testing and Analysis. London: Association for Computing Machinery, 2007. 129–139. [doi: 10.1145/1273463.1273482]

[14] Garvin BJ, Cohen MB, Dwyer MB. Evaluating improvements to a meta-heuristic search for constrained interaction testing. Empirical Software Engineering, 2011, 16(1): 61–102. [doi: 10.1007/s10664-010-9135-7]

[15] Johansen MF, Haugen Ø, Fleurey F. An algorithm for generating T-wise covering arrays from large feature models. In: Proc. of the 16th Int'l Software Product Line Conf. (Vol. 1). Salvador: Association for Computing Machinery, 2012. 46–55. [doi: 10.1145/2362536.2362547]

[16] Borazjany MN, Yu LB, Lei Y, Kacker R, Kuhn R. Combinatorial testing of ACTS: A case study. In: Proc. of the 5th IEEE Int'l Conf. on Software Testing, Verification and Validation. Montreal: IEEE, 2012. 591–600. [doi: 10.1109/ICST.2012.146]

[17] Al-Hajjaji M, Krieter S, Thüm T, Lochau M, Saake G. IncLing: Efficient product-line testing using incremental pairwise sampling. In: Proc. of the 2016 ACM SIGPLAN Int'l Conf. on Generative Programming: Concepts and Experiences. Amsterdam: Association for Computing Machinery, 2016, 144–155. [doi: 10.1145/2993236.2993253]

[18] Pett T, Thüm T, Runge T, Krieter S, Lochau M, Schaefer I. Product sampling for product lines: The scalability challenge. In: Proc. of the 23rd Int'l Systems and Software Product Line Conf. (Vol. A). Paris: Association for Computing Machinery, 2019. 78–83. [doi: 10.1145/3336294.3336322]

[19] Medeiros F, Kästner C, Ribeiro M, Gheyi R, Apel S. A comparison of 10 sampling algorithms for configurable systems. In: Proc. of the 38th IEEE/ACM Int'l Conf. on Software Engineering (ICSE). Austin: IEEE, 2016. 643–654. [doi: 10.1145/2884781.2884793]

[20] Henard C, Papadakis M, Perrouin G, Klein J, Heymans P, Le Traon Y. Bypassing the combinatorial explosion: Using similarity to generate and prioritize T-wise test configurations for software product lines. IEEE Trans. on Software Engineering, 2014, 40(7): 650–670. [doi: 10.1109/TSE.2014.2327020]

[21] She S, Lotufo R, Berger T, Wasowski A, Czarnecki K. Reverse engineering feature models. In: Proc. of the 33rd Int'l Conf. on Software Engineering. Waikiki: ACM. 2011. 461–470. [doi: 10.1145/1985793.1985856]

[22] Kuhn R, Lei Y, Kacker R. Practical combinatorial testing: Beyond pairwise. IT Professional, 2008, 10(3): 19–23. [doi: 10.1109/MITP.2008.54]

[23] Petke J, Yoo S, Cohen MB, Harman M. Efficiency and early fault detection with lower and higher strength combinatorial interaction testing. In: Proc. of the 9th Joint Meeting on Foundations of Software Engineering. Saint Petersburg: Association for Computing Machinery, 2013. 26–36. [doi: 10.1145/2491411.2491436]

[24] Reisner E, Song C, Ma KK, Foster JS, Porter A. Using symbolic evaluation to understand behavior in configurable software systems. In: Proc. of the 32nd ACM/IEEE Int'l Conf. on Software Engineering (Vol. 1). Cape Town: Association for Computing Machinery, 2010. 445–454 [doi: 10.1145/1806799.1806864]

[25] 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]

[26] Hemmati H, Briand L. An industrial investigation of similarity measures for model-based test case selection. In: Proc. of the 21st IEEE Int'l Symp. on Software Reliability Engineering. San Jose: IEEE, 2010. 141–150. [doi: 10.1109/ISSRE.2010.9]

[27] Hemmati H, Arcuri A, Briand L. Achieving scalable model-based testing through test case diversity. ACM Trans. on Software

Engineering and Methodology, 2013, 22(1): 6. [doi: 10.1145/2430536.2430540]

[28] Mondal D, Hemmati H, Durocher S. Exploring test suite diversification and code coverage in multi-objective test case selection. In: Proc. of the 8th IEEE Int'l Conf. on Software Testing, Verification and Validation (ICST). Graz: IEEE, 2015. 1–10. [doi: 10.1109/ICST.2015.7102588]

[29] Fischer S, Lopez-Herrejon RE, Ramler R, Egyed A. A preliminary empirical assessment of similarity for combinatorial interaction testing of software product lines. In: Proc. of the 9th Int'l Workshop on Search-based Software Testing. Austin: ACM, 2016. 15–18. [doi: 10.1145/2897010.2897011]

[30] Henard C, Papadakis M, Perrouin G, Klein J, Le Traon Y. PLEDGE: A product line editor and test generation tool. In: Proc. of the 17th Int'l Software Product Line Conf. Co-located Workshops. Tokyo: Association for Computing Machinery. 2013. 126–129. [doi: 10.1145/2499777.2499778]

[31] Al-Hajjaji M, Schulze M, Ryssel U. Similarity analysis of product-line variants. In: Proc. of the 22nd Int'l Systems and Software Product Line Conf. (Vol. 1). Gothenburg: Association for Computing Machinery, 2018. 226–235. [doi: 10.1145/3233027.3233044]

[32] Devroey X, Perrouin G, Legay A, Schobbens PY, Heymans P. Search-based similarity-driven behavioural SPL testing. In: Proc. of the 10th Int'l Workshop on Variability Modelling of Software-intensive Systems. Salvador: Association for Computing Machinery, 2016. 89–96. [doi: 10.1145/2866614.2866627]

[33] Xiang Y, Huang H, Li MQ, Li SZ, Yang XW. Looking for novelty in search-based software product line testing. IEEE Trans. on Software Engineering, 2022, 48(7): 2317–2338. [doi: 10.1109/TSE.2021.3057853]

[34] Luo C, Sun BQ, Qiao B, Chen JJ, Zhang HY, Lin JK, Lin QW, Zhang DM. LS-sampling: An effective local search based sampling approach for achieving high T-wise coverage. In: Proc. of the 29th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. Athens: ACM, 2021. 1081–1092. [doi: 10.1145/3468264.3468622]

[35] Henard C, Papadakis M, Harman M, Le Traon Y. Combining multi-objective search and constraint solving for configuring large software product lines. In: Proc. of the 37th IEEE Int'l Conf. on Software Engineering. Florence: IEEE, 2015. 517–528. [doi: 10.1109/ICSE.2015.69]

[36] Le Berre D, Parrain A. The SAT4J library, release 2.2: System description. Journal on Satisfiability, Boolean Modeling and Computation, 2010, 7(2–3): 59–64. [doi: 10.3233/SAT190075]

[37] Balint A, Schöning U. Choosing probability distributions for stochastic local search and the role of make versus break. In: Proc. of the 15th Int'l Conf. on Theory and Applications of Satisfiability Testing. Trento: Springer, 2012. 16–29. [doi: 10.1007/978-3-642-31612-8_3]

[38] Lehman J, Stanley KO. Abandoning objectives: Evolution through the search for novelty alone. Evolutionary Computation, 2011, 19(2): 189–223. [doi: 10.1162/EVCO_a_00025]

[39] Sánchez AB, Segura S, Parejo JA, Ruiz-Cortés A. Variability testing in the wild: The Drupal case study. Software & Systems Modeling, 2017, 16(1): 173–194. [doi: 10.1007/s10270-015-0459-z]

[40] Xiang Y, Zhou YR, Zheng ZB, Li MQ. Configuring software product lines by combining many-objective optimization and SAT solvers. ACM Trans. on Software Engineering and Methodology, 2018, 26(4): 14. [doi: 10.1145/3176644]

[41] Baluja S. Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning. Schenley Park Pittsburgh: Carnegie Mellon University, 1994.

[42] Mühlenbein H, Paaß G. From recombination of genes to the estimation of distributions I. Binary parameters, In: Proc. of the 4th Int'l Conf. on Parallel Problem Solving from Nature. Berlin: Springer, 1996. 178–187. [doi: 10.1007/3-540-61723-X_982]

[43] Baranov E, Legay A, Meel KS. Baital: An adaptive weighted sampling approach for improved T-wise coverage. In: Proc. of the 28th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. New York: ACM, 2020. 1114–1126. [doi: 10.1145/3368089.3409744]

[44] Bagheri E, Ensan F, Gasevic D. Grammar-based test generation for software product line feature models. In: Proc. of the 2012 Conf. of the Center for Advanced Studies on Collaborative Research. Toronto: Association for Computing Machinery, 2012. 87–101.

[45] Mann HB, Whitney DR. On a test of whether one of two random variables is stochastically larger than the other. The Annals of Mathematical Statistics, 1947, 18(1): 50–60. [doi: 10.1214/aoms/1177730491]

[46] Vargha A, Delaney HD. A critique and improvement of the CL common language effect size statistics of mcgraw and wong. Journal of Educational and Behavioral Statistics, 2000, 25(2): 101–132. [doi: 10.3102/10769986025002101]

[47] Friedman M. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. Journal of the American Statistical Association, 1937, 32(200): 675–701. [doi: 10.1080/01621459.1937.10503522]

[48] Audemard G, Simon L. Predicting learnt clauses quality in modern SAT solver. In: Proc. of the 21st Int'l Joint Conf. on Artificial Intelligence. Pasadena: Morgan Kaufmann Publishers Inc., 2009. 399–404.

[49] Zhang W, Sun Z, Zhu Q, Li G, Cai SW, Xiong YF, Zhang L. NLocalSAT: Boosting local search with solution prediction. In: Proc. of the 29th Int'l Joint Conf. on Artificial Intelligence. Yokohama: IJCAI, 2020. 1177–1183. [doi: 10.24963/ijcai.2020/164]

[50] Luo C, Zhao QY, Cai SW, Zhang HY, Hu CM. SamplingCA: Effective and efficient sampling-based pairwise testing for highly configurable software systems. In: Proc. of the 30th ACM Joint European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. Singapore: ACM, 2022, 1185–1197. [doi: 10.1145/3540250.3549155]

[51] Xiang Y, Huang H, Zhou Y, Li S, Luo C, Lin Q, Li M, Yang X. Search-based diverse sampling from real-world software product lines. In: Proc. of the 2022 Int'l Conf. on Software Engineering (ICSE). New York: ACM, 2022. 1945–1957.

附中文参考文献:

[4] 连小利, 张莉. 面向软件产品线中特征选择的多目标优化算法. 软件学报, 2017, 28(10): 2548–2563. http://www.jos.org.cn/1000-9825/5130.htm [doi: 10.13328/j.cnki.jos.005130]

向毅(1986－), 男, 博士, 副教授, 博士生导师, 主要研究领域为智能化软件工程, 演化计算, 软件产品线.

罗川(1991－), 男, 博士, 副教授, CCF 专业会员, 主要研究领域为约束求解, 软件测试, 云计算.

黄翰(1980－), 男, 博士, 教授, CCF 杰出会员, 主要研究领域为微计算理论与方法, 智能化软件工程, 数据智能工程, 生化反应计算机.

杨晓伟(1969－), 男, 博士, 教授, 博士生导师, 主要研究领域为机器学习, 模式识别, 智能化软件工程.