软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software [doi: 10.13328/j.cnki.jos.006905]

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

结合 SVM 与 XGBoost 的链式多路径覆盖测试用例生成*

钱忠胜, 俞情媛, 张 丁, 姚昌森, 秦朗悦, 成轶伟

(江西财经大学 信息管理学院, 江西 南昌 330013)

通信作者: 钱忠胜, E-mail: changesme@163.com

摘 要: 机器学习方法可很好地与软件测试相结合, 增强测试效果, 但少有学者将其运用于测试数据生成方面. 为进一步提高测试数据生成效率, 提出一种结合 SVM (support vector machine) 和 XGBoost (extreme gradient boosting) 的链式模型, 并基于此模型借助遗传算法实现多路径测试数据生成. 首先, 利用一定样本训练若干个用于预测路径节点状态的子模型 (SVM 和 XGBoost), 通过子模型的预测精度值筛选最优子模型, 并根据路径节点顺序将其依次链接, 形成一个链式模型 C-SVMXGBoost (chained SVM and XGBoost). 在利用遗传算法生成测试用例时, 使用训练好的链式模型代替插桩法获取测试数据覆盖路径 (预测路径), 寻找预测路径与目标路径相似的路径集, 对存在相似路径集的预测路径进行插桩验证, 获取精确路径, 计算适应度值. 在交叉变异过程中引入样本集中路径层级深度较大的优秀测试用例进行重用, 生成覆盖目标路径的测试数据. 最后, 保留进化生成中产生的适应度较高的个体, 更新链式模型 C-SVMXGBoost, 进一步提高测试效率. 实验表明, C-SVMXGBoost 较其他各对比链式模型更适合解决路径预测问题, 可提高测试效率. 并且通过与已有经典方法相比, 所提方法在覆盖率上提高可达 15%, 平均进化代数也有所降低, 在较大规模程序上其降低百分比可达 65%.

关键词: 测试用例; SVM; XGBoost; 链式模型; 多路径覆盖

中图法分类号: TP311

中文引用格式: 钱忠胜, 俞情媛, 张丁, 姚昌森, 秦朗悦, 成轶伟. 结合SVM与XGBoost的链式多路径覆盖测试用例生成. 软件学报. http://www.jos.org.cn/1000-9825/6905.htm

英文引用格式: Qian ZS, Yu QY, Zhang D, Yao CS, Qin LY, Cheng YW. Multi-path Coverage Test Case Generation Combining Chained SVM and XGBoost. Ruan Jian Xue Bao/Journal of Software (in Chinese). http://www.jos.org.cn/1000-9825/6905.htm

Multi-path Coverage Test Case Generation Combining Chained SVM and XGBoost

QIAN Zhong-Sheng, YU Qing-Yuan, ZHANG Ding, YAO Chang-Sen, QIN Lang-Yue, CHENG Yi-Wei

(School of Information Management, Jiangxi University of Finance and Economics, Nanchang 330013, China)

Abstract: Machine learning methods can be well combined with software testing to enhance test effect, but few scholars have applied it to test data generation. In order to further improve the efficiency of test data generation, a chained model combining support vector machine (SVM) and extreme gradient boosting (XGBoost) is proposed, and multi-path test data generation is realized by a genetic algorithm based on the chained model. Firstly, this study uses certain samples to train several sub-models (i.e., SVM and XGBoost) for predicting the state of path nodes, filters the optimal sub-models based on the prediction accuracy value of the sub-models, and links the optimal sub-models in sequence according to the order of the path nodes, so as to form a chained model, namely chained SVM and XGBoost (C-SVMXGBoost). When using the genetic algorithm to generate test cases, the study makes use of the chained model that is trained instead of the instrumentation method to obtain the test data coverage path (i.e., predicted path), finds the path set with the predicted path similar to the target path, performs instrumentation verification on the predicted path with similar path sets, obtains accurate paths, and calculates fitness values. In the crossover and mutation process, excellent test cases with a large path level depth in the sample set are introduced for reuse to generate test data covering the target path. Finally, individuals with higher fitness during the evolutionary generation are saved,

* 基金项目: 国家自然科学基金 (62262025); 江西省自然科学基金重点项目 (20224ACB202012)
收稿时间: 2022-07-04; 修改时间: 2022-10-08; 采用时间: 2023-01-11; jos 在线出版时间: 2023-08-09

and C-SVMXGBoost is updated, so as to further improve the test efficiency. Experiments show that C-SVMXGBoost is more suitable for solving the path prediction problem and improving the test efficiency than other chained models. Moreover, compared with the existing classical methods, the proposed method can increase the coverage rate by up to 15%. The mean evolutionary algebra is also reduced, and the reduction percentage can reach 65% on programs of large size.

Key words: test case; support vector machine (SVM); extreme gradient boosting (XGBoost); chained model; multi-path coverage

1 引 言

手动测试生成满足测试目标的数据会花费测试人员大量时间, 且需测试的目标路径往往有多条. 因此, 自动生成符合条件的测试数据, 并通过已有数据尝试覆盖多目标路径可有效减轻测试工程师的负担[1], 提高测试用例的生成效率, 也避免了较多重复性的工作.

在多路径覆盖测试生成中[2−6], 挖掘覆盖路径与测试用例间的关联性或分析路径间的相似性有助于提高测试用例质量. 同时, 为尽早生成覆盖目标路径的测试数据还需建立在用例多样性的基础上. 遗传算法 (genetic algorithm, GA) 具有生物进化、遗传变异、全局概率搜索等机制, 可生成丰富的测试数据, 其在自动生成测试数据方面的应用十分广泛. 不过, 传统遗传算法也存在自身缺陷, 如, 容易陷入局部最优, 且随着程序规模增大, 插桩运行被测程序的时间成本急剧增加, 但通过改进算法可有效缓解此类问题. 我们通过合理重用已有测试用例缓解遗传算法局部最优问题[7], 提高测试效率. 姚香娟等人[8]利用后向传播 (back propagation, BP) 神经网络模拟计算个体适应度, 有效地解决了遗传操作中插桩程序耗费时间较长的问题.

此外, 随着机器学习方法的不断成熟, 已有较多学者结合机器学习模型与测试理论进行测试相关领域的研究. Gong 等人[9]提出一种基于代理模型辅助进化的路径覆盖测试数据生成方法, 使用代理模型估计每个个体的适应度值. 我们在文献 [10] 提出一种支持向量机 (support vector machine, SVM) 回归模型预测适应度值并重用已有测试数据. 陈铁明等人[11]将抽象指令序列 N-Gram 编码作为随机森林 (random forest, RF) 的样本, 构建分类模型, 测试出恶意代码. Esteves 等人[12]使用 XGBoost (extreme gradient boosting) 算法预测软件是否易出现缺陷. 以上机器学习方法各有优缺点以及适用的场景, 合理利用这些方法可有效提高测试效率.

在众多机器学习模型中, SVM 和 XGBoost 具有小样本、低耗时、高准确性等优点而被广泛应用, 且它们对于处理不同数据类型具有各自优势.

在真实测试场景中, 测试目标包含的多条目标路径存在一定的联系. 每条路径的每个路径节点均可表示为两种状态 (即经过或未经过), 则节点状态的预测可看作一个二分类问题. 由于被测程序的输入数据类型各异, 使用单一的模型进行路径预测具有一定的局限性. 而 SVM 模型训练速度快, 对于数值型样本具有较好的分类效果, XGBoost 模型具有良好的可扩展性, 对于非数值型样本分类效果更佳. 此外, 它们均适用于小样本、高维度的测试数据. 因此, 融合 SVM 与 XGBoost 模型为生成覆盖目标路径的测试数据奠定基础, 该融合模型具有训练时间较短、可处理不同数据类型等优势.

鉴于此, 本文提出一种融合 SVM 与 XGBoost 模型并用于遗传算法进行多路径覆盖测试用例生成的策略, 利用融合模型代替插桩法获取测试数据覆盖的路径, 并通过分析预测路径与目标路径的相关性, 从而实现多目标路径测试. 主要完成如下几个方面的工作.

1) 构建链式模型. 利用一定数量的样本优先训练更适用的模型 (即, 预测效果更佳的模型), 若模型精度未达到较优值, 则训练另一个模型, 选取每个路径节点的最优子模型, 并根据路径节点出现的顺序将最优子模型链接起来, 构建 C-SVMXGBoost 链式模型, 将其代替插桩法来预测测试数据的覆盖路径.

2) 融入链式模型. 利用遗传算法进化生成测试用例时, 先初步筛选优秀个体, 通过训练好的链式模型预测当前测试数据的路径, 并搜索与预测路径相似的目标路径; 再进一步筛选优秀个体, 若存在相似目标路径, 则进行插桩验证, 获取精确路径, 计算适应度值. 通过计算原训练样本的覆盖路径与目标路径的路径层级深度, 在交叉、变异等进化过程中, 引入路径层级深度较大值对应的测试数据, 作为可重用的用例.

3) 更新链式模型. 在使用模型过程中, 对优秀个体进一步筛选, 可获取测试数据所对应的精确路径, 将具有精

确路径的数据作为新样本, 更新路径预测模型 C-SVMXGBoost.

4) 实验对比分析. 从模型性能与测试用例生成效率两方面设计实验. 其中, 模型性能分析主要对比单模型和融合模型的性能; 测试用例生成效率方面主要将本文方法与其他融合模型的测试方法, 以及已有经典方法的测试效率进行对比.

2 相关工作

在测试用例生成的研究中, 有不少学者从多路径角度提高测试用例效率. Sun 等人[2]利用测试用例间的路径距离设定测试用例的优先级, 进一步提高测试效率, 但该方法计算路径距离时仅考查两个路径集中某一路径集的单向影响, 并未考虑它们间的双向作用, 使得路径距离的计算不够准确. 廖伟志等人[3]改进蚁群算法以求解多路径测试数据生成问题, 但未结合单信息素与多信息素对测试用例生成的影响. Feng 等人[4]将已求解的分支路径对应的测试数据视为启发式信息设计一种覆盖多目标的测试数据生成策略, 其路径信息较为粗糙, 缺乏对相邻路径节点的权重分析. 范书平等人[5]根据测试用例覆盖分支的情况以及均衡度提取关键用例, 提高多路径测试效率, 该方法综合衡量测试用例对完整路径的均衡度, 却忽略了某一关键分支节点对整体的影响. 潘峰等人[6]为提高并行程序的多路径测试数据生成效率, 提出多路径覆盖调度序列排序方法, 仅通过随机采样方式尝试覆盖目标路径, 但并未使用启发式方法, 这对初始用例要求较高, 依赖性强.

从上述研究可发现, 为提高多路径用例生成效率, 一般可将辅助用例生成的相关理论运用到测试中, 同时保持测试用例的丰富多样性, 以尽快生成所需测试数据. 为使生成的测试数据具有丰富性和多样性, 可借助启发式算法完成. 而遗传算法利用先验知识进行遗传进化操作, 这丰富种群个体, 极大提升进化速度, 减少迭代次数.

我们通过关键字流图检测相似程序, 并将相似程序的优秀用例重用到遗传算法的进化生成中[7], 提高被测程序测试用例生成效率, 该方法仅对存在可重用测试用例的相似程序时有效, 反之则无效, 且研究的是单目标路径. Di Nucci 等人[13]提出基于超体积的遗传算法, 以解决使用多个测试覆盖准则的测试用例优先级排序问题, 对于高度冗余测试嵌套的软件系统, 该方法性能有所下降. 夏春艳等人[14]提出基于否定选择遗传算法的路径覆盖测试集生成的方法, 将否定选择的策略融入遗传算法中, 提高了路径覆盖率, 但未考虑否定选择后测试用例的优先级. 我们提出一种依据关键点概率计算种群个体贡献度的多路径覆盖策略[15], 并依此设计适应度函数, 指导个体迁移, 进化生成覆盖目标路径的测试数据, 忽略了关键边对个体贡献度的影响. 戚荣志等人[16]通过建立弹性分布数据集, 划分子种群, 并将其分布在集群的节点中, 结合遗传算法的适应度值和进化操作完成测试, 但对于高维组合测试算法的性能下降会明显.

可见, 遗传算法在测试用例生成方面具有较高的普适性, 在该算法上做相应的改进可优化测试用例, 有效地提高测试效率. 此外, 机器学习方法是辅助测试数据生成的工具之一, 其具有优秀的分类与预测能力, 可改善传统软件测试过程, 因此融入机器学习的软件测试方法, 在近些年受到越来越多学者的关注.

姚香娟等人[8]将神经网络模型运用到测试中, 减少传统方法上适应度的计算时间, 该方法仅解决单目标路径的用例生成问题, 且测试效率还有待提高. Gong 等人[9]提出一种基于代理模型辅助进化的路径覆盖测试数据生成方法, 根据多模态特性将程序样本划分为若干簇, 分别训练各自代理模型, 在对个体进行评估时, 选取合适的代理模型预测个体适应度值, 进而提高模型准确度以及测试效率, 这些代理模型对样本均衡的要求较高, 而在真实测试中往往缺乏难覆盖路径的测试样本, 进而影响代理模型的选择. Skocelas 等人[17]提出一种循环神经网络生成测试用例的方法, 验证依赖于先前网络输入的顺序或时变模式的上下文单元功能.

我们提出一种支持向量机回归模型 (SVR) 的测试用例生成与重用方法[10], 通过 SVR 模型预测适应度值并重用已有测试数据, 该方法解决单路径覆盖问题, 若求解多路径则需重复执行. 姜慧研等人[18]将 SVM 与蚁群算法结合构造软件缺陷预测模型, 但该方法在参数寻优过程中需要较长的时间. Pradhan 等人[19]提出一种基于聚类的遗传算法, 利用非支配精英选择策略减少在选择父解决方案时的随机性, 以支持多目标测试优化, 但未考虑主题对算法参数的影响.

基于树的分类模型准确率高、复杂度低, 具有较好的分类效果, 在软件缺陷检测方面应用相对广泛. Esteves 等人[12]使用 XGBoost 算法预测软件是否易出现缺陷, 并利用预测能力和模型可解释性间的联系, 提出模型采样方法, 该方法可找到更易出现缺陷的模块, 在特征组合过程中采用穷举法, 这使得在构建模型前需对数据处理花费较多时间. Bhati 等人[20]将 XGBoost 集成到 IDS 中, 为 IDS 提供防范位置攻击的安全性. 杨宏宇等人[21]将加权投票思想引入随机森林分类模型, 加强对 Android 恶意软件的区分, 但解决 APK 文件批量反编译耗时较长.

从现有研究来看, 机器学习方法可很好地与软件测试相结合, 推进测试进程. 有学者将其与遗传算法相结合, 应用于模拟计算适应度值, 完成基于路径的测试用例生成. 但是, 他们未考虑将其应用于多目标路径中, 且有些模型所需训练时间较长, 预测准确度较低, 这拖慢了测试的整体效率. 还有些学者将机器学习方法应用于二分类或多分类问题的求解, 以检测软件缺陷.

对于测试用例生成而言, 获取测试数据覆盖被测程序的路径至关重要[5]. 将路径节点状态看作分类问题, 通过机器学习方法, 可预测出一条完整的路径, 而且, 随着程序规模的增加, 该方法可有效节省程序的插桩时间. 此外, 结合遗传算法可丰富测试数据的同时, 生成更优秀的测试用例 (即, 可覆盖目标路径的测试数据), 以提高测试效率. 当被测程序较为复杂时, 测试数据的输入类型往往具有多样性. 这些种群个体作为模型的训练样本, 符合小样本、非线性与高纬度的特征.

针对上面的分析, 本文结合 SVM 模型的小样本、非线性等特性以及 XGBoost 模型的低复杂度、高准确率等优势, 构建一种基于路径预测的链式模型 C-SVMXGBoost, 代替插桩法获取测试数据的覆盖路径, 根据其路径寻找并尝试覆盖更多的相似目标路径. 为使生成的测试数据可覆盖更多路径, 合理地重用训练样本中的部分数据 (即, 适应度值较高的个体), 辅助进化生成更优个体, 并将其保留, 用于更新 C-SVMXGBoost 链式模型.

3 总体框架与相关基础

本文利用 C-SVMXGBoost 链式模型实现多路径覆盖测试用例生成 (整体框架如后文图 1 所示), 包括 C-SVM-XGBoost 链式模型构建 (算法设计见第 4.1 节)、遗传进化测试生成 (算法设计见第 4.2 节) 及 C-SVMXGBoost 链式模型更新 (算法设计见第 4.3 节) 等过程. 此外, 从模型样本选择、C-SVMXGBoost 链式模型评估等角度设计对比实验与消融实验等,结果及其分析见第 5.3 节.

1) C-SVMXGBoost 链式模型构建模块. 根据测试数据及其对应的路径节点覆盖信息, 利用 SVM 和 XGBoost 模型构建每个路径节点的状态预测子模型, 并依据路径节点出现顺序筛选每个节点精度最高的子模型, 依次将其链接构建路径预测链式模型.

2) 遗传进化测试生成模块. 将初始种群输入到链式模型中预测当前测试数据的覆盖路径, 搜索与预测路径相似的目标路径, 对存在相似路径的测试用例获取精确路径, 计算个体适应度, 同时在种群进化中重用路径层级深度较大的测试数据, 进而生成目标测试用例.

3) C-SVMXGBoost 链式模型更新模块. 在种群进化中保留优秀个体以及它们对应的精确路径, 将具有精确路径的数据作为新样本, 更新 C-SVMXGBoost 链式模型, 在后续遗传进化中引用.

3.1 预备知识

SVM 模型是按监督学习方式对数据进行二元分类以获得较好泛化能力的广义分类器. 在实际的软件测试中, 以测试用例为样本, 几乎无法获得可将路径节点状态 (即: 测试数据覆盖节点时其状态值为 1, 反之为 0) 分离的超平面, 故这里引入 *hinge* 损失函数, 利用 SVM 求解能将被覆盖节点与未被覆盖节点分离的最优超平面, 即为目标函数, 如公式 (1) 所示:

$$\begin{cases} \min \dfrac{1}{2}\| \omega \|^2 + C\sum\limits_{i=1}^{n}\xi_i \\ \text{s.t. } y_i(\omega^{\mathrm{T}}x_i + b) \geqslant 1-\xi_i,\ \xi_i \geqslant 0,\ i = 1,2,\ldots,n \end{cases} \tag{1}$$

其中, n 为测试样本数; C 为惩罚参数 ($C>0$); ξ_i 为松弛变量 ($\xi_i = \max(0, 1-y_i(\omega_i + b))$), 即为 *hinge* 损失函数; ω 为

分离超平面系数, b 为分离超平面常量参数, 这两个参数均需通过模型训练获取; x_i 为第 i 个训练样本, y_i 为第 i 个样本对应的路径节点状态.

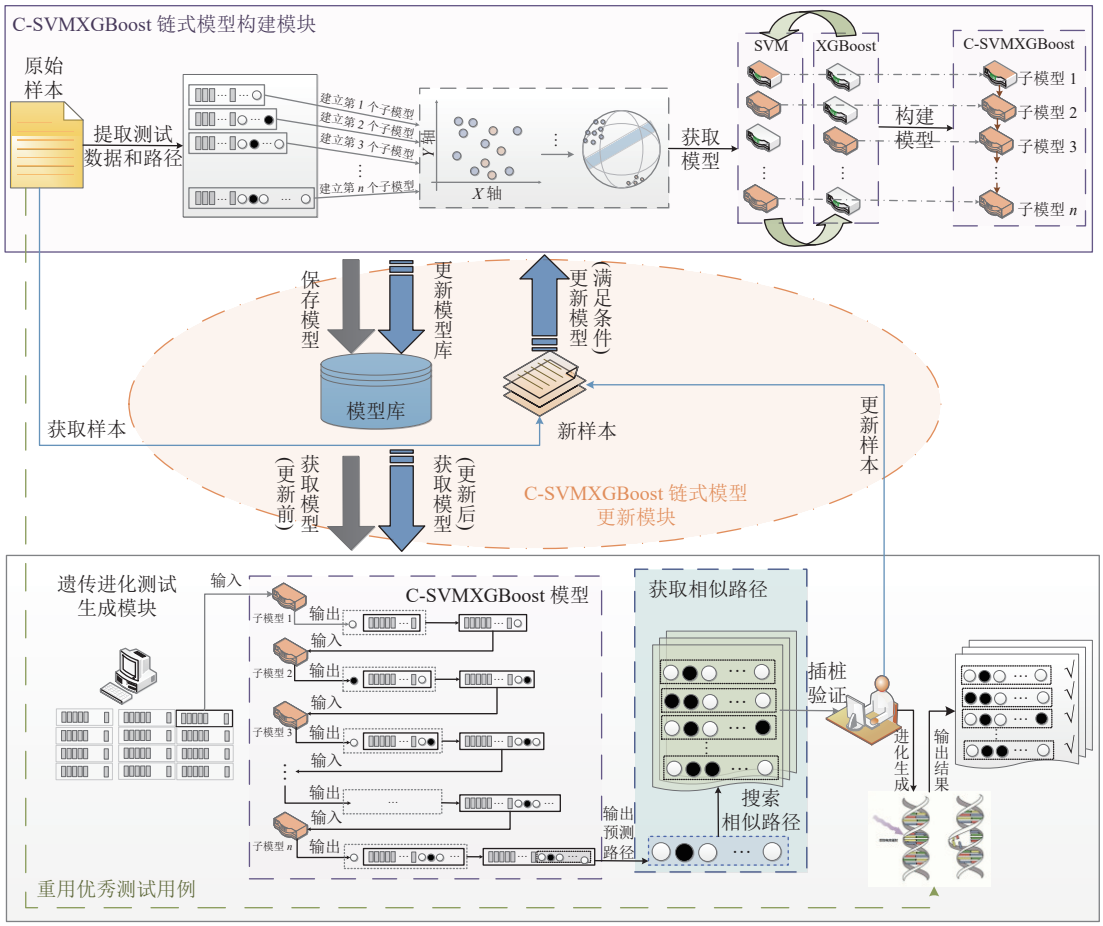


图 1　链式模型构建与用例生成框架

通过拉格朗日乘子法可将上述有约束条件的目标函数转化为无约束的, 并采用高斯核函数将非线性分类问题转换为某个空间维度的线性分类, 如公式 (2) 所示:

$$\begin{cases} \min \dfrac{1}{2}\displaystyle\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j K(x_i * x_j) - \sum_{i=1}^{n}\alpha_i \\ \text{s.t.}\ \displaystyle\sum_{i=1}^{n}\alpha_i y_i = 0,\ 0 \leqslant \alpha_i \leqslant C,\ i = 1,2,\ldots,n \end{cases} \tag{2}$$

其中, α_i 和 α_j 分别为第 i 和 j 个样本对应目标函数中的拉格朗日乘子, 且 $\alpha_i, \alpha_j \geqslant 0$; y_i 和 y_j 分别为第 i 和第 j 个样本的分类值, 即路径节点状态; $K(x_i * x_j)$ 为高斯核函数. 根据公式 (2) 即可求得最优 $\alpha^* = (\alpha_1^*, \alpha_2^*, \ldots, \alpha_n^*)^{\mathrm{T}}$.

$$\omega^* = \sum_{i=1}^{n}\alpha_i^* y_i x_i \tag{3}$$

$$b^* = y_j - \sum_{i=1}^{n}\alpha_i^* y_i (x_i * x_i) \tag{4}$$

其中, α_i^* 为上述求解最优 α^* 的一个分量. 将求解的最优 α^* 值代入公式 (3) 和公式 (4) 即可得出需求解的分离超平

面, 即系数ω和参数b, 此时模型构建完成. 当输入数据时, 根据训练的模型即可预测对应的路径节点状态.

另外, 与 SVM 一样, XGBoost 模型也可应用于求解分类问题. 针对复杂或不均衡样本, 不仅模型训练时间相比于其他模型耗时较少, 而且准确性也相对较高.

XGBoost 模型可认为是一种经过优化的分布式梯度提升库, 由t个基模型组成的一个加法模型, 利用该模型也可将路径节点状态分类, 预测节点状态值. 通过不断调整预测的路径节点状态值与真实状态值间的损失, 使其达到最小, 此时路径节点状态分类模型构建完成, 目标函数如公式 (5) 所示:

$$Obj^{(t)}=\sum_{i=1}^{n}l(y_i,y_i^p)+\sum_{i=1}^{t}\Omega(f_i) \tag{5}$$

其中, n为测试样本数; $l(y_i,y_i^p)$为路径节点状态真实值y_i与路径节点状态预测值y_i^p的损失函数; $\sum_{i=1}^{t}\Omega(f_i)$为全部$t$个基模型的复杂度$f_i$之和, 将其加入到目标函数中作为正则化项, 防止模型过拟合.

以第t个模型为例, 此模型对第i个样本x_i的预测值可表示为如公式 (6) 所示:

$$y_i^{p(t)}=y_i^{p(t-1)}+f_t(x_i) \tag{6}$$

其中, $y_i^{p(t-1)}$是第$t-1$步模型的节点状态预测值, 为常数; $f_t(x_i)$为此次需要加入新模型的预测值. 根据泰勒定理, $f(x)$对应的损失函数$l(y_i,y_i^{p(t-1)}+f_t(x_i))$如公式 (7) 所示:

$$l(y_i,y_i^{p(t-1)}+f_t(x_i))=l(y_i,y_i^{p(t-1)})+g_if_t(x_i)+\frac{1}{2}h_if_t^2(x_i) \tag{7}$$

其中, $l(y_i,y_i^{p(t-1)})$为节点状态真实值与第$t-1$步模型的节点状态预测值的损失函数; g_i为第t个模型$f(x)$对应损失函数的一阶导数, h_i为第t个模型$f(x)$损失函数的二阶导数, $\frac{1}{2}$为二阶导数的系数. 将上述公式带入目标函数, 经化简后目标函数如公式 (8) 所示:

$$Obj^{(t)}\cong\sum_{i=1}^{n}[g_if_t(x_i)+\frac{1}{2}h_if_t^2(x_i)]+\Omega(f_t) \tag{8}$$

在目标函数中, 只需求解每一步损失函数的一阶导数和二阶导数, 然后最优化目标函数, 使函数值最小, 获得每一步$f(x)$, 再根据加法模型, 即可得到一个完整的路径节点状态分类模型, 用于预测节点状态.

精度 (*Precision*) 是评估模型好坏的重要指标之一, 反应分类模型求解相关实例的能力, 如公式 (9) 所示:

$$Precision=\frac{TP}{TP+FP} \tag{9}$$

其中, *TP* 表示预测为正样本 (即节点状态值为 1), 且实际也为正样本的特征数; *FP* 表示预测为正样本, 但实际为负样本 (即节点状态值为 0) 的特征数.

SVM 模型对于数值型样本具有较好的分类效果, 且适用于小样本的测试数据. XGBoost 具有良好的可扩展性, 对于非数值型样本分类效果较好. 合理地结合这两种模型, 可满足不同程序的预测.

本文构建的 C-SVMXGBoost 链式模型即为若干个最优 SVM 与 XGBoost 子模型的链式融合. 根据公式 (9) 计算子模型的精度, 并以此筛选出最优子模型, 如公式 (10) 所示; 链式模型 C-SVMXGBoost 构建如公式 (11) 所示:

$$Sub_{Mod}{}^{m}=Max_{precision}(\mathrm{SVM}_{node_m},\mathrm{XGBoost}_{node_m}) \tag{10}$$

$$Chain_{Mod}=SubChained\left(\sum_{m=1}^{k}Sub_{Mod}{}^{m}\right) \tag{11}$$

公式 (10) 中, SVM_{node_m}为第m个路径节点对应子模型 SVM 的精度; $\mathrm{XGBoost}_{node_m}$为第$m$个路径节点对应子模型 XGBoost 的精度; $Sub_{Mod}{}^{m}$为第m个路径节点对应精度最大的子模型, 即最优子模型. 公式 (11) 中, $SubChained\left(\sum_{m=1}^{k}Sub_{Mod}{}^{m}\right)$表示所有最优子模型按照路径节点顺序依次链接的链式模型, 即 C-SVMXGBoost.

C-SVMXGBoost 模型用于模拟求解测试数据的覆盖路径, 是由若干个子模型链式融合而成, 每个子模型均需

一定数量的测试样本进行训练, 且它们均包含输入数据与预期输出结果. 每个子模型训练样本的输入数据格式不同, 其表现形式如公式 (12) 所示, 输出结果为测试数据对应的某一路径节点状态.

$$X_k = \begin{cases} (x_1, x_2, x_3, \ldots, x_n),\ k=1 \\ (x_1, x_2, x_3, \ldots, x_n, p_1, p_2, \ldots, p_{k-1}),\ k>1, k \in N \end{cases} \tag{12}$$

在公式 (12) 中, X_k 表示第 k 个子模型样本的输入. 记被测程序的输入数据为 $Inp=(x_1, x_2, \ldots, x_n)$, 其中 x_a (a=1, 2, 3,…, n) 为第 a 个输入分量; 记当前测试数据输入插桩程序后获得的覆盖路径为 $P=(p_1, p_2, \ldots, p_k)$ ($\{k|k \geqslant 1, k \in N\}$), 其中 p_b (b=1, 2, 3,…, k) 为第 b 个路径节点状态, 且 $p_b \in \{0,1\}$ (即测试数据经过节点时其状态值为 1, 否则为 0). 需注意的是, 每个预测路径节点子模型的训练样本输入均不包括预测节点本身, 例如第 k 个子模型的训练样本输入不包括第 k 个路径节点状态值, 第 k 个节点状态是由输入数据及前 $k-1$ 个路径节点状态预测所得. 所有子模型依次输出的预测结果链成一条完整的预测路径 P', 即 $P'=(p'_1, p'_2, p'_3, \ldots, p'_k)$.

为使样本具有一定的代表性, 在随机生成时会进行筛选, 尽量使其均匀分布在被测程序的输入域. 选用样本的容量不宜过大或过小. 若样本容量太小, 则无法体现模型的准确性; 若样本容量太大, 则需消耗更多计算资源以及模型构建时间, 降低了测试效率.

3.2 适应度函数设计

适应度函数是筛选优秀个体的关键性影响因素, 根据不同需求设计适应度函数是遗传算法中必不可少的步骤. 我们通过构建路径预测的链式模型, 从而找到更多相似的目标路径, 尽可能在进化生成过程中充分使用测试数据, 以覆盖更多的目标路径.

生成易覆盖目标路径的测试数据没有太大意义, 而可通过难覆盖路径节点的测试数据被认为是较优种群个体. 其中, 路径层级深度与插桩程序中路径节点出现的次序及状态有关, 是衡量路径节点关键性的因素之一.

综上所述, 适应度函数的设计需综合考虑测试数据的覆盖路径 (记为 $P=(p_1, p_2, \ldots, p_k)$) 和目标路径 (记为 $P^*=(p_1^*, p_2^*, \ldots, p_k^*)$) 间的相似度, 以及路径层级深度. 本文方法的路径相似度 (记为 Sim)、路径层级深度 (记为 $Depth_{\text{level}}$) 的计算分别如公式 (13) 和公式 (14) 所示:

$$Sim(P, P^*) = \frac{\sum_{b=1}^{k} (p_b = p_b^*)}{k} \tag{13}$$

其中, $\sum_{b=1}^{k} (p_b = p_b^*)$ 表示测试数据覆盖路径 P 中第 b 个节点与目标路径 P^*中第 b 个节点状态相同的数量. 测试数据覆盖路径与目标路径间的相似度即为两条路径对应节点状态相同的节点数与单条路径总节点数的比值.

$$Depth_{\text{level}} = \sum_{b=1}^{k} \frac{p_b + \varepsilon}{k} \times h \tag{14}$$

其中, p_b 为测试数据覆盖路径的第 b 个节点的状态; h 为当前路径节点的位置; ε ($0<\varepsilon \leqslant 0.01$) 是一个很小的常量, 以保证在节点状态为 0 时, 该节点路径层级深度不为 0, 本方法设其值为 0.01.

路径相似度从测试用例覆盖能力方面筛选目标个体, 经链式模型选取的测试用例对应的精确覆盖路径与目标路径间的相似性越高, 表明该个体越容易进化生成目标路径的用例. 同时, 相似性比较的过程涉及目标路径节点的难易覆盖性. 此外, 为避免路径相似度较低时可能丢弃覆盖目标路径中某些难覆盖节点的个体, 这里引入路径层级深度, 从路径节点关键性方面对目标个体择优, 个体穿越路径的层级越深, 表明该个体经过的路径节点越重要, 则赋予其越大的适应度.

为平衡路径相似度与层级深度对个体适应度的综合影响, 设置了权重系数 α, 由于路径相似度是判别目标路径是否被覆盖的主要因素, 其权重更大, 故有 $0.5<\alpha \leqslant 1$. 当路径相似度 Sim 为 1 时, 表示当前测试用例已覆盖目标路径, 则将 α 设置为 1, 此时个体适应度值仅受 Sim 影响, 即适应度值为 1; 当路径相似度 Sim 不为 1 时, 表示当前测试用例未覆盖目标路径, 此时个体适应度值会受路径相似度与层级深度的共同影响, 因此, 需根据适应度值继续

进化生成优秀个体.

基于此, 将适应度函数 (记为 *Fitness*) 表示为路径相似度与路径层级深度的组合, 如公式 (15) 所示:

$$Fitness = \alpha \times Sim + (1-\alpha) Depth_{\text{level}} \tag{15}$$

4 基于 C-SCMXGBoost 链式模型的测试生成

本文将 SVM 和 XGBoost 等机器学习方法与遗传算法相结合, 利用构建的链式模型获取测试数据的覆盖路径, 减少程序插桩次数, 并寻找与预测路径相似的目标路径集, 尽可能覆盖更多的目标路径, 从而减少适应度计算与重复测试消耗的时间, 提高测试效率.

4.1 C-SVMXGBoost 链式模型构建

模型构建模块主要训练用于模拟测试数据覆盖路径的 C-SVMXGBoost 链式模型, 其训练样本来源于预先生成的测试数据. 模型构建过程如图 2 所示.

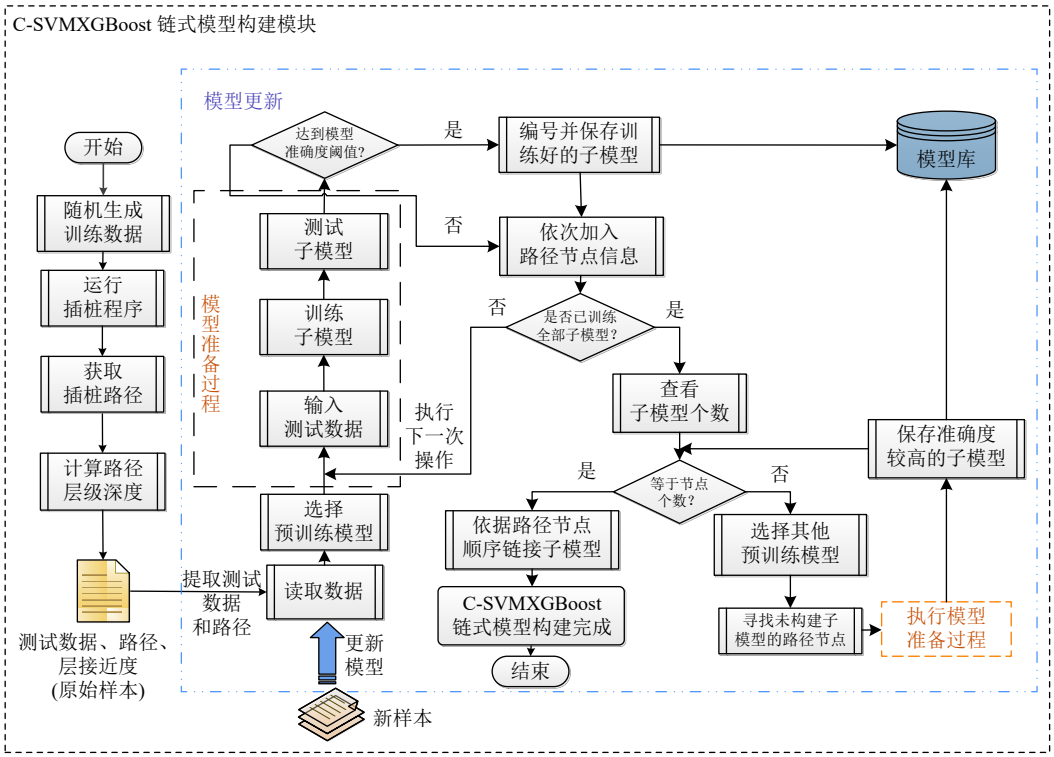


图 2 链式模型 C-SVMXGBoost 的构建过程

一条完整的路径由多个路径节点连接而成, 每个路径节点并不孤立存在, 前一个节点的状态对后一个节点具有一定的影响, 因此, 在训练链式模型的子模型时需考虑前一个路径节点的信息. 具体构建过程见算法 1.

算法 1. 链式模型 C-SVMXGBoost 构建.

输入: 子模型精度阈值 *SubModel_Threshold* (*SMT*), 路径节点个数 *Number_Node* (*NN*);

输出: C-SVMXGBoost 模型.

BEGIN

1. *test_data*←Generate a set of test data randomly;

2. *paths*, *level_depth*←Get path information and compute level depth of *test_data*;
   // 获取路径信息并利用公式 (14) 计算路径层级深度
3. Select the pre-trained model according to the input data type; // 根据数据类型选择预训练模型
4. *count* = 0; // 统计需训练的子模型数
5. *num_model* = []; // 保存子模型序列
6. *precison_lst* = []; // 保存子模型精度
7. *sample*←*test_data*; // 初始训练样本
8. **WHILE** (*count* < *NN*): // 插桩路径的所有子模型构建
9. *submodel*←modelTrain(*sample*); // 根据公式 (1)–公式 (4) 或公式 (5)–公式 (8) 训练不同子模型
10. *precision*←preciCompute(*submodel*); // 根据公式 (9) 计算模型精度
11. *precison_lst*.append(*precision*); // 保存子模型精度
12. *sample*←Update *sample* by next node; // 将下一个节点加入并更新样本
13. *count* += 1;
14. **IF** *precision* >= *SMT*:
15. Save *submodel*;
16. *num_model*.append(*count*);
17. **END IF**
18. **END WHILE**
19. **WHILE** len(*num_model*) < *NN*: // 某个路径节点还未具有较优子模型
20. **IF** *num_model*[*i*] < > *num_model*[*i*+1]–1:
21. *count* = *num_model*[*i*] + 1; // 第 *count* 个路径节点缺失较优子模型
22. *sample*←Update *sample* by *test_data* and *path node*s; // 更新样本
23. Select another pre-trained model to retrain the submodel that does not reach the threshold;
    // 选择其他预训练模型来训练未达到阈值的模型
24. Select optimal submodel of the current node by Formula(10); // 通过公式 (10) 选择该路径节点的最优子模型
25. Save *submodel*;
26. *num_model*.append(*count*);
27. **END IF**
28. **END WHILE**
29. C-SVMXGBoost←Chain the *submodels* according to the order of path nodes;
    // 根据公式 (11) 按路径节点顺序依次链接子模型
30. Output C-SVMXGBoost;

END

在算法 1 中, 第 1 行和第 2 行为模型训练样本准备阶段, 随机生成测试数据后将其输入插桩程序获取路径, 并计算其路径层级深度. 在随机生成数据的基础上进一步筛选, 确保样本均衡. 第 3–7 行, 选择预训练模型, 初始化参数及样本. 第 8–29 行, 构建 C-SVMXGBoost 模型. 其中, 第 8–18 行训练每个路径节点的子模型, 计算模型精度并更新样本, 统计子模型个数. 若其达到最优子模型阈值, 则将其保存; 若未达到, 将其丢弃. 第 19–28 行主要实现部分路径节点最优子模型的筛选. 其中, 第 20 行和第 21 行寻找需进行二次训练模型的路径节点; 第 22 行和第 23 行, 设置样本, 选择与训练该路径节点子模型; 第 24–26 行筛选、保存当前路径节点的最优子模型. 第 29 行将已完成的最优子模型依据路径节点顺序链接, 构建链式模型, 并在第 30 行输出.

4.2 融合 C-SVMXGBoost 链式模型的遗传进化测试生成

遗传算法模块主要借助已构建的链式模型生成待测程序的用例, 完成多目标路径的测试. 遗传算法生成测试用例的过程如图 3 所示.

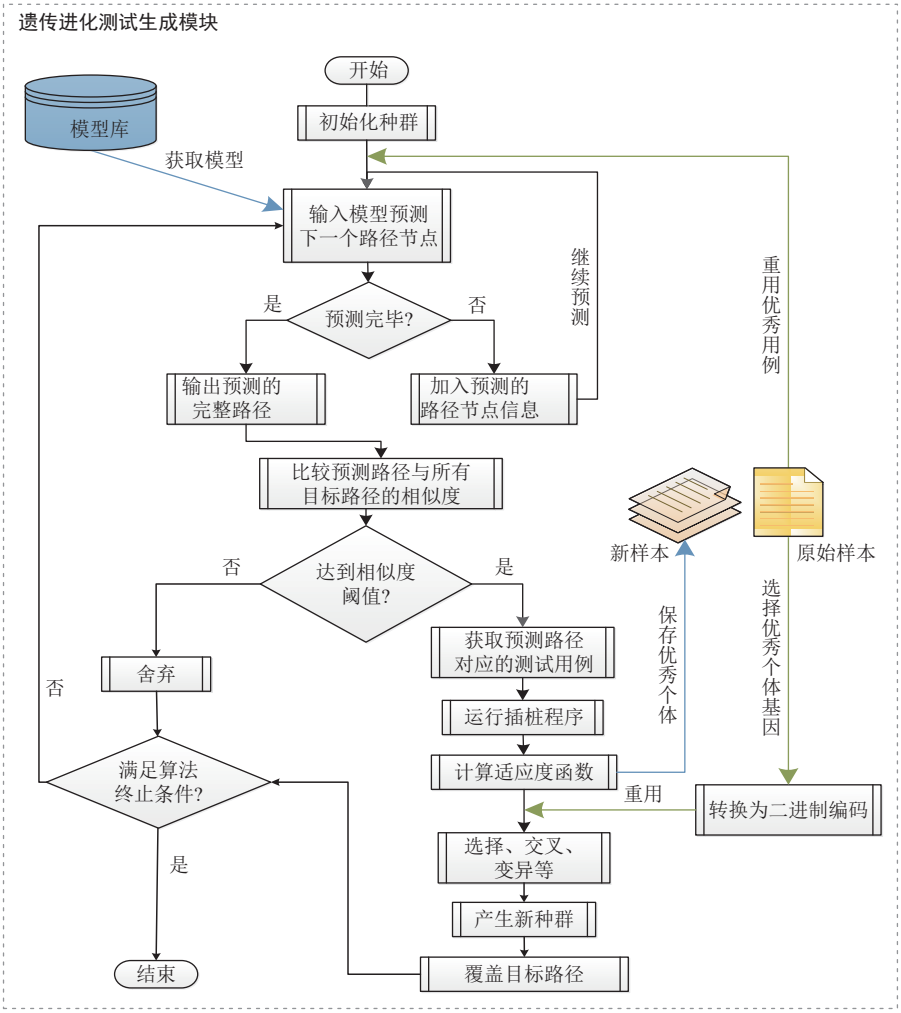


图 3 利用融合 C-SVMXGBoost 链式模型的遗传算法生成测试用例

在利用遗传算法生成测试用例的过程中, 本文采用构建的链式模型预测种群个体的覆盖路径, 再根据此路径寻找相似目标路径, 计算个体的适应度. 在交叉变异时引入原始样本中较为优秀的用例, 种群在其他过程中仍然按照传统模式下的方式进化. 测试用例生成过程见算法 2.

算法 2. 融合 C-SVMXGBoost 链式模型的遗传进化测试生成.

输入: C-SVMXGBoost 模型, 目标路径集 *Tarpath*, 路径相似度阈值 *Path_Sim_Threshold*(*PST*), 种群大小 *Pop_Size*, 个体 *Pop_Individual*, 染色体长度 *Chromosome_Length*(*CL*), 进化代数 *Evolutionary_Time*(*ET*), 交叉概率 *Cross_Probability*(*CP*), 变异概率 *Mutate_Probability*(*MP*), 优秀个体 *Excellent_Individual*(*EI*);
输出: 新测试数据 *New_testdata*, 未覆盖目标路径集 *UncoveredTarPath*.

BEGIN

```
1.  init_pop←Initialize(Pop_Size, CL);
2.  test_data←toDecimal(init_pop); // 将初始种群信息转换为十进制
3.  iteration←0;
4.  FOR iteration IN ET:
5.      new_pop_indiv ← init_pop;
6.      prepath←Input test_data into C-SVMXGBoost; // 获取预测路径
7.      sim_path←comparePath (prepath, Tarpath); // 获取与预测路径相似的目标路径
8.      IF sim_path <> ∅: // 存在相似路径
9.          precise_path←getFromInstru(test_data); // 在插桩程序中获取精确路径
10.         fit← calculateFitness(test_data); // 根据公式 (13)–公式 (15) 计算适应度值
11.         IF fit == 1:
12.             UncoveredTarPath←Delete current path from Tarpath; // 删除当前目标路径
13.             New_testdata←test_data; // 保存当前测试数据, 用于后续模型更新
14.         END IF
15.     END IF
16.     Roulette(Pop_Size, new_pop_indiv, fit); // 轮盘赌法选择个体
17.     Pop_list←Crossover (Pop_Size, CL, CP, EI); // 引入新个体进行交叉操作
18.     new_pop_indiv←Mutate (Pop_list, CL, MP); // 实施变异操作
19.     test_data←toDecimal(new_pop_indiv);
20.     iteration += 1; // 统计迭代次数
21.     IF UncoveredTarPath== ∅:
22.         BREAK;
23.     END IF
24. END FOR
25. Output New_testdata, UncoveredTarPath;
END
```

在算法 2 中, 第 1 行和第 2 行初始化种群并将其转换为十进制数. 第 3–24 行, 借助 C-SVMXGBoost 模型完成遗传操作, 生成覆盖目标路径的测试数据. 其中第 5 行和第 6 行, 将数据输入链式模型中, 获得预测路径. 第 7–10 行, 根据预测路径寻找目标路径中的相似路径, 将当前测试数据输入到插桩程序, 获取精确路径, 并计算适应度值. 第 11–14 行, 若适应度值为 1, 表明目标路径被覆盖, 将其删除, 并保存当前测试数据; 第 16–20 行, 当目标路径未被全覆盖或不存在预测路径与目标路径的相似路径时, 则进行轮盘赌选择、交叉、变异等过程, 在交叉过程中引入优秀个体基因, 通过变异生成新的种群, 并统计迭代次数. 第 21–23 行, 判断全部目标路径是否被覆盖, 若全被覆盖则完成测试; 否则在进化代数达到阈值时才终止. 第 25 行输出新测试数据 (用于更新样本) 以及目标路径的覆盖情况.

4.3 C-SVMXGBoost 链式模型更新

C-SVMXGBoost 链式模型精度是提高测试效率的关键, 为使每个路径节点的预测结果均较准确, 该模型需进一步更新, 以提高模型精度. 链式模型更新过程见算法 3.

算法 3. C-SVMXGBoost 链式模型更新.

输入: 链式模型 C-SVMXGBoost, 新测试数据 *New_testdata*, 目标路径集 *TarPath*;
输出: Updated_C-SVMXGBoost 模型.

BEGIN

1. *prepath*←Input *New_testdata* into C-SVMXGBoost; // 由算法 2 中新测试数据获取预测路径
2. *sim_path*←Search target path in *TarPath* similar to *prepath*; // 寻找预测路径的相似目标路径
3. **IF** *sim_path* < > ∅:
4. *precise_path* ← getFromInstru(*New_testdata*); // 在插桩程序中获取精确路径
5. *lev_depth* ← calculateLevDepth(*precise_path*); // 根据公式 (14) 计算精确路径的层级深度
6. *new_sample* ← Update(*New_testdata*, *precise_path*, *lev_depth*);
7. **END IF**
8. Updated_C-SVMXGBoost←Repeat the steps in Algorithm 1;
9. Output Updated_C-SVMXGBoost;

END

在算法 3 中, 第 1 行和第 2 行, 将算法 2 中获得的新测试数据输入 C-SVMXGBoost 链式模型中, 获取预测路径及其相似的目标路径; 第 3–7 行, 若存在相似路径, 将测试数据输入到插桩程序中获得精确路径, 计算该路径的层级深度, 且将测试数据、精确路径以及路径层级深度保存并更新样本; 第 8 行和第 9 行, 根据算法 1 来更新模型并输出.

5 实验设计与分析

为检验 C-SVMXGBoost 链式模型预测路径的准确性, 以及测试用例生成效率, 本节选取经典的基础程序和工业程序进行实验以展开对比分析.

5.1 评价指标

为评价提出的基于链式模型的多路径测试用例生成方法的效率, 同时降低算法随机性带来的影响, 本文定义如下几个评价指标.

1) 链式模型分类精度. 链式模型中所有子模型分类精度的均值.

2) 平均训练时间. 独立地训练每个被测程序的初始模型 C-SVMXGBoost 若干次, 计算每次训练所需时间总和的平均值.

3) 执行时间. 测试程序所使用的时间. 其中, 传统遗传法的执行时间为程序通过遗传算法完成用例生成的时间; 本文方法的执行时间为构建链式模型与生成测试数据时间之和.

4) 覆盖率. 将程序独立地运行若干次, 若每次执行程序时, 用例可在最大进化代数内覆盖全部目标路径, 则将覆盖次数加 1. 覆盖率为若干次程序运行后最终覆盖次数与总运行次数的比值.

5) 进化代数总量. 一定次数的用例生成中, 所有单条目标路径被覆盖所需进化代数的数值总和.

6) 平均进化代数. 待生成用例的程序独立地执行一定次数, 每次执行时, 若在最大进化代数内, 用例能覆盖所有目标路径, 则保存最后覆盖路径的进化代数; 若在最大进化代数内, 用例未能全覆盖, 则将最大值 (即, 进化代数阈值) 作为本次测试程序时所需的进化代数. 平均进化代数表达了这些次实验所记录代数总和的均值.

此外, 为更直观、便捷地展示本文方法与各对比方法在不同指标下的实验效果, 在执行时间、覆盖率和平均进化代数上扩展几个指标, 并在相应部分进行文字阐述, 详细说明见第 5.3.3 节、第 5.3.4 节和第 5.4 节.

5.2 实验设置

实验环境为 Win 10 操作系统、Python 编程语言、jdk 1.8 版本、PyCharm 编程工具. 为使 C-SVMXGBoost 链式模型更好地适用于遗传进化测试策略, 让模型选出的优秀个体可充分地参与进化生成, 同时确保引入的个体基因丰富多样, 避免个体在进化过程中陷入局部最优, 以生成更优的测试用例, 这里将遗传参数设置为多点交叉概率为 0.9, 单点变异概率为 0.1, 初始种群规模为 40 等. 通过从链式模型预测结果的准确性以及测试用例生成效率等

方面设计对比实验, 由实验结果可知遗传参数设置是合理的. 另外, 根据文献 [7], 遗传参数设置在合理的区间内对方法效率的影响较小.

为便于本文方法有效性验证, 下面给出所需实验程序, 包括 2 个基础程序和 11 个工业程序, 具体信息见表 1.

表 1　实验程序信息表

程序类型	程序编号	程序名称	代码行数	函数个数
基础程序	PG1	冒泡排序	8	1
	PG2	三角形判定	15	1
工业程序	PG3	LuhnCheck	24	1
	PG4	计算器	196	5
	PG5	Snake	163	11
	PG6	Gobang	470	23
	PG7	Account	1 670	36
	PG8	Nav Acc Data PP	3 990	155
	PG9	Sed	9 341	77
	PG10	Flex	14 405	162
	PG11	GO	28 547	2 982
	PG12	DepSolver	8 988	227
	PG13	ClustalW	23 265	468

在表 1 中, PG1 (https://blog.csdn.net/s1156605343/article/details/106177863) 和 PG2 (https://www.jb51.net/article/184534.htm) 为基础程序, PG3–PG13 为工业程序. 其中 LuhnCheck 为银行卡号校验算法 (https://blog.csdn.net/weixin_33716557/article/details/91539545); PG4 为简易计算器算法 (https://www.cnblogs.com/quemengqio/p/7799468.html); Snake 为贪吃蛇单人游戏; Gobang 为融入 AI 算法的五子棋赢棋算法; 程序 Account 为登录网站, 其功能包括用户登录、注册等; 程序 Nav Acc Data PP 为航海事故数据处理平台. 这几种程序 (PG5–PG8) 均来源于 https://gitee.com/explore. 程序 Sed 和 Flex 均选用其中两个版本 (https://sir.csc.ncsu.edu/php); 程序 GO 是一款抽象策略游戏 (https://blog.csdn.net/chongshangyunxiao321/article/details/50997404); 程序 DepSolver 是一种并行多媒体三维静电解算器 (https://github.com/chef-boneyard/depsolver); 程序 ClustalW 是一种常用的多基因序列比对工具 (https://github.com/coldfunction/CUDA-clustalW). 在这几种程序中, 考虑到源代码编程语言的差异, 使用 PG1–PG10 验证本文方法的有效性, 并选取其中部分程序, 以及 PG11–PG13 用于与其他方法对比.

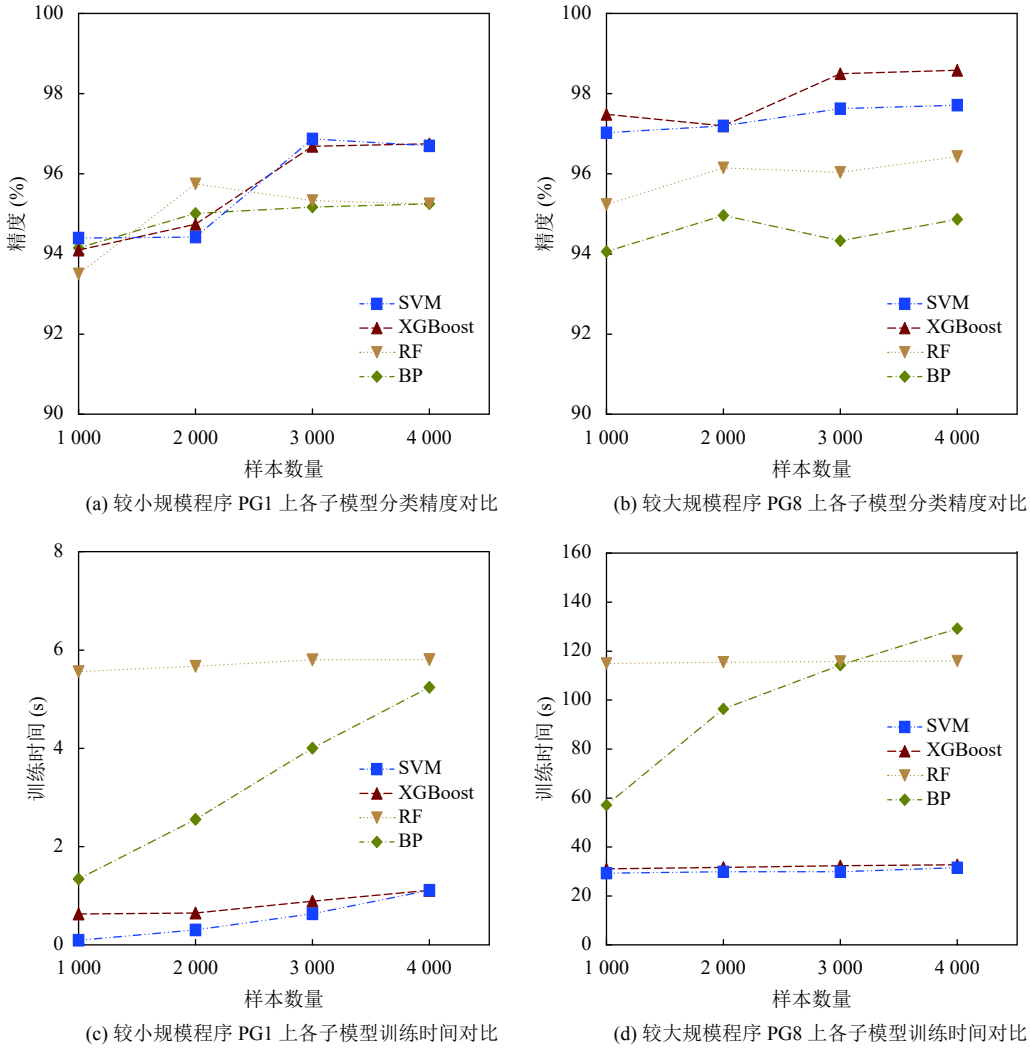
5.3　模型验证与测试分析

链式模型 C-SVMXGBoost 在测试数据生成过程中起到关键作用, 其分类精度与构建时间对提升测试效率至关重要. 此外, 在遗传算法中合理地重用优秀测试用例可提升测试数据质量. 下面先验证本文方法的有效性, 包括子模型精度、链式模型精度、模型构建的时间消耗、测试用例生成效率等方面.

5.3.1　子模型样本数量选择

SVM、XGBoost、RF (随机森林)、BP (BP 神经网络) 等模型因具有良好分类效果, 可将它们应用于测试方面. 在样本相对均衡的前提下, 其数量对模型的精度具有一定的影响. 就如何合理地选取不同程序的训练样本数量, 综合考虑程序的规模以及测试数据的复杂度 (即样本的多样性), 分别选取较小规模程序 (以基础程序 PG1 为例) 和较大规模程序 (以工业程序 PG8 为例), 对不同样本数量下的各子模型分类精度与时间进行分析. 考虑到样本数量不宜过大, 故这里分别设置为 1 000, 2 000, 3 000, 4 000. 实验结果如图 4 所示.

由图 4(a) 和图 4(b) 可知, 在较小规模程序上, 4 种子模型 SVM, XGBoost, RF, BP 的分类精度峰值分别在样本数量为 3 000, 4 000, 4 000, 4 000; 在较大规模程序上, 它们的分类精度峰值均在样本数量为 4 000 时, 若样本数量继续增加, 其精度值仍有可提升的空间. 但是, 从每个子模型在不同程序上分类精度的变化趋势可看出, 随着样本数量的增加, 它们值的变化也趋于缓慢, 部分子模型的精度还呈现下降趋势.



(a) 较小规模程序 PG1 上各子模型分类精度对比

(b) 较大规模程序 PG8 上各子模型分类精度对比

(c) 较小规模程序 PG1 上各子模型训练时间对比

(d) 较大规模程序 PG8 上各子模型训练时间对比

图 4　各子模型在不同规模程序下及不同样本数量上的分类精度、训练时间对比

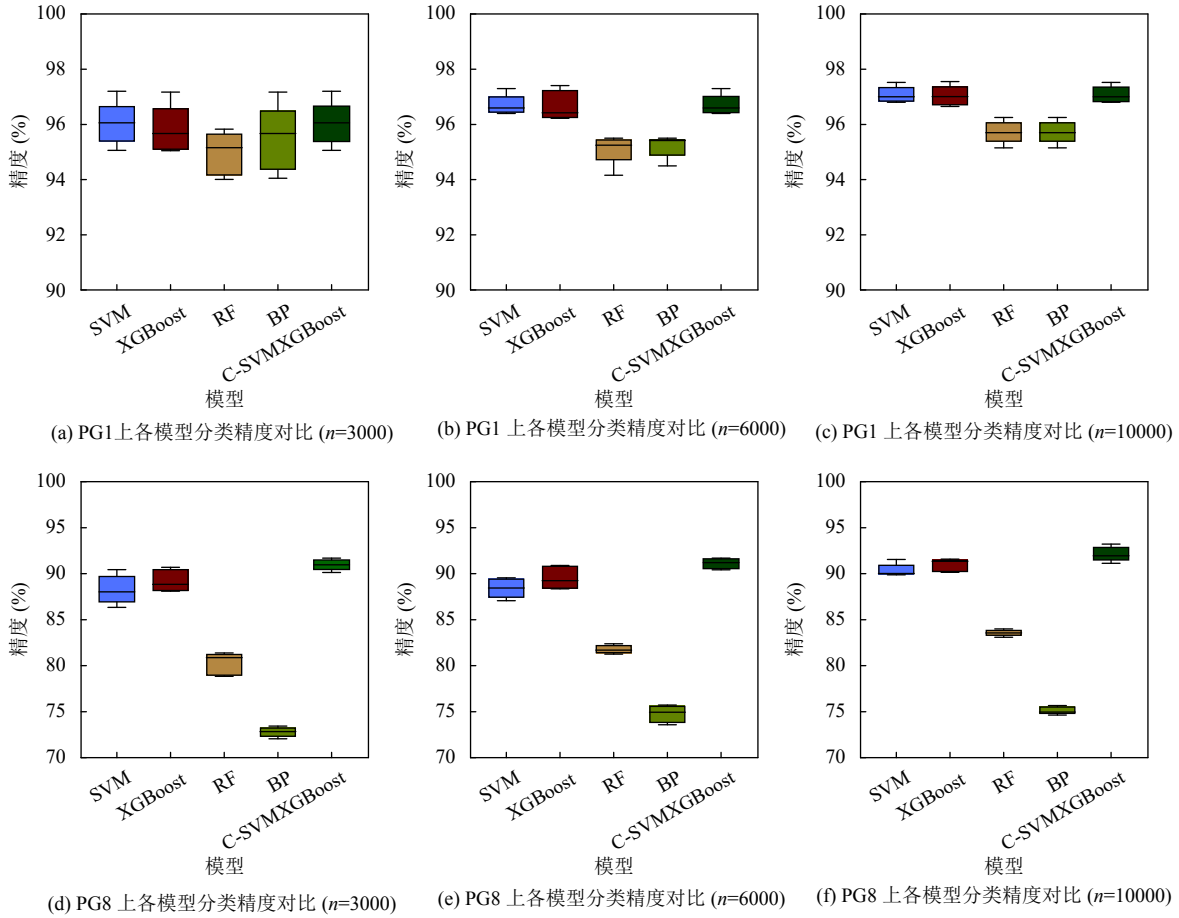
在保证子模型精度的情况下, 其训练时间应尽可能少. 由图 4(c) 和图 4(d) 可看出, 不论是较小规模还是较大规模程序, 随着样本数量增加, SVM, XGBoost, RF 这 3 个子模型的训练时间增加缓慢, BP 则增加较为明显, 且 SVM 和 XGBoost 的训练时间少于 RF 和 BP 这两个子模型.

结合图 4(a) 和图 4(b) 中的分类精度, 当样本数量从 2000 增加到 3000 时, 大部分子模型的分类精度仍处于提升阶段, 样本数量从 3000 增加到 4000 时, 各子模型精度增加不明显, 除了 BP 子模型, 其他模型的训练时间增幅较小. 因此, 综合考虑不同子模型效率, 样本数量设置为 3000 左右较为合适. 此外, 对比不同程序上各子模型分类精度和训练时间, 在小样本数量上, SVM 和 XGBoost 较其他两种子模型结果均较优. 在测试数据为数值型程序 PG1 上, SVM 模型在保证分类结果准确的同时, 训练时间也较少; 在包含非数值型测试数据的程序 PG8 上, XGBoost 模型在精度上相比于 SVM 更优, 训练时间略高于 SVM. 在面对不同程序时, 基于分类精度与训练时间方面的考虑, SVM 和 XGBoost 这两个子模型可互为补充, 相辅相成, 适用于测试中的路径节点状态分类与预测问题.

5.3.2　模型对比及分析

由上述实验可知, 继续增加样本数量, 子模型精度仍有可提高的空间. 为选取更合适的样本数量, 以及验证本

文构建的链式模型 (即 C-SVMXGBoost) 是否比其他模型在精度与时间方面更有优势, 选取的实验程序与上一节一致, 并设置更多样本数量, 分别为 3 000, 6 000, 10 000, 就模型分类精度进一步展开对比. 每组实验分别重复 5 次, 结果如图 5 所示, 其中 n 表示样本数量.



(a) PG1上各模型分类精度对比 (n=3000) (b) PG1 上各模型分类精度对比 (n=6000) (c) PG1 上各模型分类精度对比 (n=10000)

(d) PG8 上各模型分类精度对比 (n=3000) (e) PG8 上各模型分类精度对比 (n=6000) (f) PG8 上各模型分类精度对比 (n=10000)

图 5 不同规模程序上的各模型分类精度对比

由图 5 可知, 对比不同模型在不同程序和不同样本数量上的精度, 可得到如下几点结论.

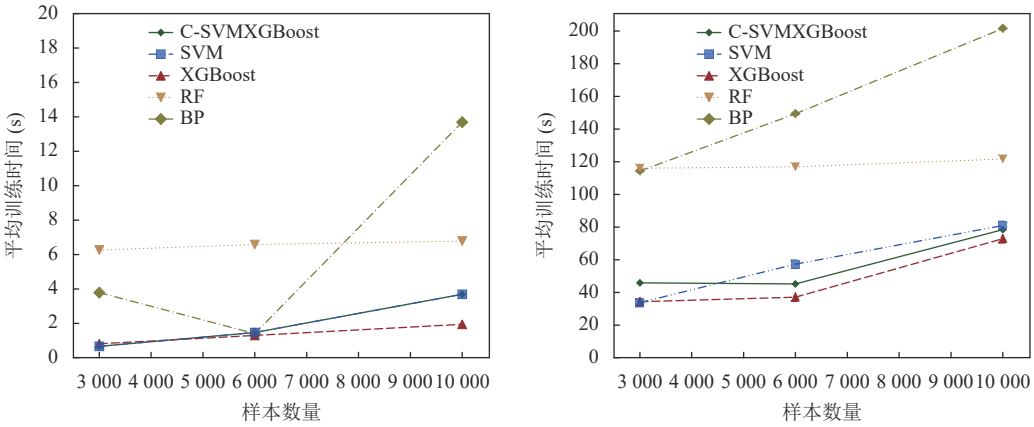
1) 程序规模和复杂度对模型分类精度具有一定的影响. 在较小规模程序 PG1 上, 这几种模型的精度均高于 94%, 而在较大规模程序 PG8 上, 它们的精度有所下降. 其中, 子模型 SVM 和 XGBoost 的分类精度在 85%–95% 范围内; RF 在 75%–85% 范围内; BP 在 70%–78% 范围内. 不论是对较小规模的简单程序 PG1, 还是较大规模的复杂程序 PG8, SVM 模型和 XGBoost 模型精度在不同样本数量的情况下, 均高于 RF 和 BP 模型.

2) 链式模型的融合过程提高了模型的精度. 对比 SVM, XGBoost 和 C-SVMXGBoost 这 3 种模型, 其中对于程序 PG1, 子模型 SVM 模型精度略高于 XGBoost 模型; 对于程序 PG8, 子模型 XGBoost 模型精度值高于 SVM 模型. 在处理不同类型数据上, SVM 模型与 XGBoost 模型各有优势. 链式模型 C-SVMXGBoost 是最优子模型 SVM 和 XGBoost 的融合, 不论是在数值型程序 PG1 还是包含非数值型程序 PG8 上, 它的精度均高于这两个子模型, 因此 C-SVMXGBoost 可更准确地预测路径节点状态.

3) 样本数量对模型精度影响相对较小. 随着样本数量增加, 在不同程序上, 以上几种模型分类精度没有明显提升. 在保证模型训练精度的前提下, 尽可能使用较少样本, 当样本数量为 3 000 时可满足样本需求. 链式模型

C-SVMXGBoost 相对于其他模型, 精度最高, 处于 90%–95% 之间, 但随着样本数量的增加, 其自身提升程度不明显, 这再一次说明该链式模型在小样本测试数据上即可达到较高精度.

利用路径预测模型生成测试用例过程中, 模型精度和模型训练所需时间是影响测试效率的关键因素. 因此, 在上面实验的基础上, 对这几种模型的平均训练时间进行分析, 如图 6 所示. 这里采用折线图进行展示, 其中横坐标表示样本数量 (与图 5 中的样本量对应, 这里只统计样本量为 3 000, 6 000, 10 000 这 3 种情况), 纵坐标表示各模型 5 次实验的平均训练时间.



(a) PG1 (b) PG8

图 6 不同规模程序上的各模型平均训练时间对比

由图 6 可知, 对比不同模型在不同程序和不同样本数量上的平均训练时间, 可得到如下几点结论.

1) 程序规模和复杂度对模型训练时间具有较大影响. 从不同模型在程序 PG1 和 PG8 上的结果可看出, 对于规模较小的简单程序 PG1, 在不同样本数量中, 几种模型的平均训练时间主要在 0–8 s 之间, BP 模型在样本数量为 10 000 时, 平均训练时间最高可达约 15 s; 对于规模较大的稍微复杂的程序 PG8, 在不同样本数量中, 平均训练时间主要集中在 20–80 s 和 100–120 s 之间, BP 模型在样本数量为 10 000 时, 平均训练时间最高可达约 200 s. 由此可见, 规模较大、较复杂程序训练路径预测模型较为耗时, 其原因是这类程序不仅数据复杂, 且路径节点较多, 需训练较多子模型. 此外, 相对于其他几种子模型, SVM 和 XGBoost 模型所需训练时间最短, BP 模型训练时间最长.

2) 链式模型的融合过程需要消耗一定的时间. 我们的 C-SVMXGBoost 模型由两种较优的子模型 SVM 和 XGBoost 融合而成. 对比 SVM, XGBoost 和 C-SVMXGBoost 这 3 种模型, 其中子模型 SVM 在样本量为 3 000 时, 平均训练时间略低于 XGBoost, 随着样本数量的增加, 其训练时间逐渐高于 XGBoost. 在程序 PG8 上, 链式模型 C-SVMXGBoost 略高于这两个模型, 其原因是为提高链式模型整体精度, 则需花费多一点时间寻找最优子模型.

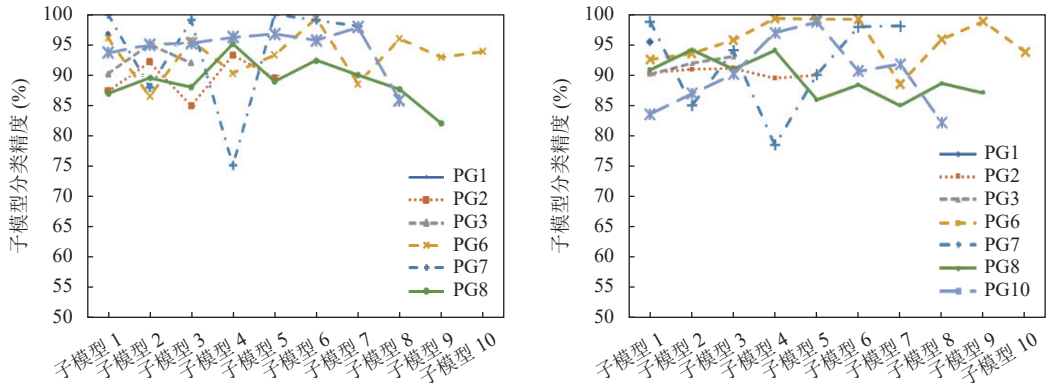
3) 样本数量对模型平均训练时间具有一定影响. 随着样本数量的增加, 不论是基础程序 PG1 还是工业程序 PG8, 模型 SVM, XGBoost, RF 和 C-SVMXGBoost 的平均训练时间增加缓慢, 在程序 PG8 上, BP 模型训练时间急剧增加. 在样本数量为 3 000 时, SVM, XGBoost 和 C-SVMXGBoost 模型时间花费较少, RF 时间相对较多; BP 模型训练时间与样本数量相关性较大, 且耗时最多. 这再一次验证了样本数量对这几种模型的影响.

综合以上几点分析与前面图 5 实验结果可知, 本文构建的链式模型 C-SVMXGBoost 选取了模型精度较高且训练时间较少的子模型进行链式融合, 在模型精度和训练时间上较其他模型而言, 具有较大优势. 为应对不同程序的模型精度与训练时间的需求, 将样本数量设置为定值. 且对于测试数据生成而言, 样本数量不宜过大, 否则消耗过多的样本筛选时间, 根据上面的分析, 将样本量设置为 3 000. 此外, 程序 PG1 和 PG8 为规模差异较大的两个程序, 且它们的输入数据类型不一致, 单一模型很难满足不同程序上路径节点状态预测准确性的要求, 而链式模型能适应不同程序特点以及测试样本分布等情况, 可根据模型精度选取较优子模型, 提高其综合性能.

5.3.3 C-SVMXGBoost 模型评估

精度常用于衡量分类模型的好坏, 计算每个子模型的精度不仅可更清晰地观测模型预测效果, 还可为后续链式模型更新提供参考依据.

这里根据第 3.1 节的公式 (9) 计算子模型精度, 并使用折线图展示部分程序的每个路径节点分别在子模型 SVM 和 XGBoost 上的精度, 如图 7 所示. 其中, 横坐标表示子模型依据路径节点顺序的编号, 纵坐标表示子模型对应的精度. 注意: 由于不同程序的路径节点个数不同, 故部分子模型在某些路径下缺少精度. 此外, 由于部分程序具有较多子模型, 这里不便全部展示, 所以仅给出部分子模型精度.



(a) 不同程序上的 SVM 子模型分类精度　　(b) 不同程序上的 XGBoost 子模型分类精度

图 7　不同程序上的 SVM 或 XGBoost 子模型分类精度

由图 7 可知, 从总体曲线分布情况来看, SVM 模型和 XGBoost 模型对于大部分路径节点状态预测结果较优. 其中, SVM 模型对于程序 PG1, PG2, PG7, PG10 的初始路径节点预测效果较好, 均高于 XGBoost; 对于程序 PG3, PG6, PG8, XGBoost 模型结果较优. 随着路径节点信息不断地加入样本, 这两个模型对于其他节点状态预测效果各有优势. 因此, 在保证模型训练时间较少的基础上, 为解决不同类型程序的路径预测, 融合模型 SVM 与 XGBoost 有利于提高路径预测模型的准确性. 从实验中也可发现, 当被测程序测试数据 (即初始样本) 差异性较小时, 如数据类型单一, 在模型精度达到一定阈值的前提下, 为降低链式模型训练与融合时间, 预训练模型 (即链式模型的初始模型) 可选择 SVM 模型; 对于测试数据差异较大, 如测试数据为非数值, 且长度较短、测试数据输入项较多等, 预训练模型可选择 XGBoost 模型. 还可发现, SVM 和 XGBoost 模型在部分路径节点上存在较大差异, 对它们进行择优与融合可构建更优的链式模型. 此外, 在模型构建时一般无法保证每个路径节点对应的子模型的初始测试样本都相对均衡, 由模型的特点可知, 在测试样本相对不均衡的情况下, XGBoost 模型更有优势.

为进一步验证链式模型分类精度与训练效率, 选取 2 个基础程序 (PG1 和 PG2) 和 4 个工业程序 (PG3, PG6, PG8 和 PG10) 独立执行 5 次, 以所有子模型精度的均值表示链式模型 C-SVMXGBoost 的精度, 结果如表 2 所示.

由表 2 可知, 在若干次实验中, 从链式模型精度来看, 对于不同的程序, 本文构建的链式模型精度较高, 均可保持在 85% 以上, 部分程序模型精度可达 95% 以上. 其中, 程序 PG3 模型精度最低也达 87.73%; 程序 PG6 最高可达 97.85%; 程序 PG10 最高可达 92.60%. 对于较复杂的程序 PG8, 模型精度仍可达 90% 以上.

从链式模型的训练时间来看, 模型训练时间与程序的规模以及分支语句的个数具有较大联系. 对于规模较小且分支语句较少的程序 PG1 和 PG3, 它们的模型训练时间较少, 平均训练时间分别为 0.67 s 和 0.66 s; 规模较大且分支语句较多的程序 PG6 和 PG8, 模型所需平均训练时间较多, 分别为 22.39 s 和 45.83 s. 此外, 子模型精度对链式模型的训练时间也有一定影响. 其中, 相比于 PG1 而言, 规模较小的简单程序 PG2, 其训练时间消耗较多. 由于 PG2 预训练子模型的精度较低, 故需选择另一个预训练模型来构建当前路径的子模型, 这必将花费一定的时间.

综上所述, 对于大部分程序, 链式模型在精度方面具有较好的效果; 在训练时间方面, 部分程序花费时间较多, 但其是否会影响整个测试效率需将构建的链式模型融入到测试用例生成中进一步验证.

表 2 C-SVMXGBoost 模型各评价指标值

程序编号	实验次数	链式模型精度 (%)	训练时间 (s)	平均训练时 (s)
PG1	第1次	96.80	0.65	0.67
	第2次	95.06	0.67	
	第3次	96.13	0.67	
	第4次	95.70	0.71	
	第5次	95.04	0.64	
PG2	第1次	92.60	3.04	3.01
	第2次	90.14	3.01	
	第3次	91.23	3.01	
	第4次	90.04	2.97	
	第5次	91.91	3.03	
PG3	第1次	90.66	0.66	0.66
	第2次	87.73	0.67	
	第3次	92.40	0.65	
	第4次	89.33	0.65	
	第5次	89.73	0.69	
PG6	第1次	97.14	21.21	22.39
	第2次	97.85	24.23	
	第3次	96.90	23.15	
	第4次	97.10	21.16	
	第5次	96.93	22.20	
PG8	第1次	91.31	45.95	45.83
	第2次	91.70	45.76	
	第3次	90.97	46.80	
	第4次	90.13	45.88	
	第5次	91.81	44.78	
PG10	第1次	90.85	7.95	7.62
	第2次	92.53	8.37	
	第3次	91.62	7.13	
	第4次	89.02	7.12	
	第5次	92.60	7.53	

本文方法旨在将链式模型融入于遗传算法中, 生成可覆盖目标路径的测试用例. 为验证链式模型可适用于路径覆盖测试生成以及本文方法的有效性, 这里选取传统遗传法 (即, 按照遗传算法的基本步骤, 通过插桩程序获得测试数据的覆盖路径) 与本文方法进行对比. 结果如表 3 所示. 为更直观地展示在各评价指标下, 本文方法比传统遗传法改进的程度, 对原指标进行扩展, 增加 3 个评价指标. 其中, 进化代数总量之差百分比为传统遗传法的进化代数总量与本文方法的进化代数总量的差值占传统遗传法的比例; 平均进化代数之差百分比为传统遗传法的平均进化代数与本文方法的平均进化代数的差值占传统遗传法的比例; 执行时间之差百分比为传统遗传法的执行时间与本文方法的执行时间的差值占传统遗传法的比例.

由表 3 可知, 与传统遗传法相比, 本文方法在覆盖率、进化代数总量和平均进化代数方面均有非常好的结果, 执行时间方面也具有一定优势.

覆盖率方面, 传统遗传法均未达到 100%, 且大部分被测程序处于 50%–80% 之间, 而本文方法在大部分程序上的覆盖率达到 100%, 在程序 PG8 和 PG10 上的覆盖率也较高, 分别为 95% 和 97%, 覆盖率提升较明显. 进化代数总量方面, 本文方法覆盖目标路径所需进化代数总和远少于传统遗传法. 对于简单程序 PG1 和 PG3, 它们的目

标路径数较少, 传统遗传法的进化代数总量分别达到 4 万代和 6 万代之多; 本文方法为 875 代和 5051 代, 与传统遗传法相比, 至少减少 92.21%. 随着目标路径数的增加, 传统遗传法所需进化代数总量急剧增加, 本文方法则相对缓慢. 在较复杂程序 PG8 中, 传统遗传法的进化代数总量达到最高, 为 615621 代, 本文方法的进化代数总量也达到最高, 为 195410 代, 但仍比传统遗传法的减少了 68.26%. 平均进化代数方面, 在程序 PG1 和 PG2 中, 传统遗传法的分别为 314.47 代和 457.07 代, 本文方法分别为 6.23 代和 8.63 代, 与传统遗传法相比, 分别减少了 98.02% 和 98.11%; 在程序 PG6, PG7, PG8 中, 传统遗传法的分别为 291.28 代、248.74 代、498.05 代, 本文方法分别为 65.10 代、73.41 代、98.69 代, 相比于传统遗传法, 平均进化代数分别减少了 77.65%, 70.49%, 80.18%.

表 3　本文方法与传统遗传法在不同程序上的各指标对比

待测程序	目标路径数	评价指标	对比方法		指标 (%)			C-SVMXGBoost 模型更新次数
			传统遗传法	本文方法	进化代数总量之差百分比	平均进化代数之差百分比	执行时间之差百分比	
PG1	2	覆盖率 (%) 进化代数总量 平均进化代数 执行时间 (s)	77 43252 314.47 53.38	100 875 6.23 57.27	97.98	98.02	–7.29	0
PG2	5	覆盖率 (%) 进化代数总量 平均进化代数 执行时间 (s)	60 75540 457.07 65.02	100 2539 8.63 42.31	96.64	98.11	34.93	0
PG3	2	覆盖率 (%) 进化代数总量 平均进化代数 执行时间 (s)	58 64814 612.43 77.05	100 5051 35.63 64.40	92.21	94.18	16.42	0
PG4	21	覆盖率 (%) 进化代数总量 平均进化代数 执行时间 (s)	73 128470 335.32 62.41	100 26365 43.63 60.49	79.48	86.99	3.08	0
PG5	35	覆盖率 (%) 进化代数总量 平均进化代数 执行时间 (s)	64 215473 425.38 85.10	100 25710 29.54 67.04	88.07	93.06	21.22	0
PG6	57	覆盖率 (%) 进化代数总量 平均进化代数 执行时间 (s)	75 292493 291.28 178.50	100 89534 65.10 163.72	69.39	77.65	8.28	0
PG7	61	覆盖率 (%) 进化代数总量 平均进化代数 执行时间 (s)	82 285126 248.74 301.20	100 10092 73.41 285.87	96.46	70.49	5.09	1
PG8	123	覆盖率 (%) 进化代数总量 平均进化代数 执行时间 (s)	65 615621 498.05 831.28	95 195410 98.69 807.83	68.26	80.18	2.82	1
PG9	12	覆盖率 (%) 进化代数总量 平均进化代数 执行时间 (s)	67 175379 430.67 66.84	100 34621 28.13 50.49	80.26	93.47	24.46	0
PG10	20	覆盖率 (%) 进化代数总量 平均进化代数 执行时间 (s)	71 190217 357.07 129.30	97 50230 70.33 117.11	73.59	80.30	9.43	1

以上 3 种评价指标结果表明, 在覆盖多目标路径过程中, 本文方法不仅可在同一代种群中覆盖更多的目标路径, 而且可在较少的进化代数内覆盖更多的目标路径.

执行时间方面, 本文方法比传统遗传法提升程度较小. 对于程序 PG1, 执行时间之差百分比为−7.29%, 即本文方法执行时间略高于传统遗传法, 其主要原因是该程序规模小、复杂度低, 在传统遗传法执行过程中运行插桩程序本身不需要花费太多时间, 而本文方法则需花费链式模型融合时间以及对满足插桩条件的测试用例进行验证的时间; 对于程序 PG2, PG3 和 PG4, 执行时间上与传统遗传法相比, 分别降低了 34.93%, 16.42%, 3.08%, 即本文方法的执行时间低于传统遗传法, 这是由于这 3 个程序的路径节点数较 PG1 多, 且路径覆盖难度更大. 随着程序规模以及复杂度增加, 对于程序 PG5, PG6, PG9, PG10, 本文方法的执行时间相对于传统遗传法减少相对较多, 分别减少了 21.22%, 8.28%, 24.46%, 9.43%. 对于程序 PG7 和 PG8, 本文方法执行时间相比于传统遗传法节约相对较少, 分别减少了 5.09% 和 2.82%, 其原因是, 在这两个程序的链式模型中, 子模型个数相对较多, 原始样本不足以使得初始模型预测结果准确, 链式模型精度更易受影响; 且在遗传操作中产生一定数量的优秀个体后需更新链式模型, 进而测试时间相对较多; 此外在调用模型过程中也需花费一定时间. 但尽管如此, 本文方法通过减少不必要的插桩时间, 以及利用路径相似性与重用适应度较高的测试用例, 在较少进化代数下生成可覆盖目标路径的用例, 这在一定程度上缓和链式模型本身的融合时间.

可见, 对于不同输入数据类型的程序, 利用 C-SVMXGBoost 链式模型可有效提高测试用例生成效率. 而且, 随目标路径数增多, 相对较难覆盖路径节点数也会增加, 较优测试样本数相对较少, 这样, 链式模型的整体精度必然会受到一定影响, 尽管如此, 本文构建的模型仍十分有效.

5.3.4 基于链式模型的测试效果

由第 5.3.1–5.3.3 节实验可知, 一方面, 单一的分类模型不仅精度较低, 且对精准预测不同类型程序的路径具有一定的局限性; 另一方面, 不同模型间的过多融合不仅使模型选择过程较为复杂, 同时会花费更多融合时间. 因此, 无需进一步做过多模型的消融实验, 但需与各两两融合的链式模型 (它们采用与本文相同的链接方式) 进行对比, 以便充分验证本文融合的链式模型是否较其他链式模型更适用于多路径覆盖的测试用例生成. 不同链式模型信息如表 4 所示.

表 4 不同模型的子模型链式融合信息表

链式融合模型	SVM	XGBoost	RF	BP
C-SVMRF	√	—	√	—
C-SVMBP	√	—	—	√
C-XGBRF	—	√	√	—
C-XGBBP	—	√	—	√
C-RFBP	—	—	√	√
C-SVMXGBoost	√	√	—	—

表 4 中包含 SVM, XGBoost, RF 和 BP 这 4 种模型, 其中 C-SVMRF 为子模型 SVM 和 RF 的链式融合, C-XGBBP 为子模型 XGBoost 和 BP 的链式融合, 其他链式模型也依此方式进行融合.

下面将表 4 中的链式模型融入到遗传算法中, 检验不同模型在多路径测试数据生成中的效果. 这里选取程序规模和目标路径数相差较大的程序 PG1, PG3, PG6, PG7, PG8 和 PG10 进行实验, 结果如表 5 所示. 为更直观地展示本文的链式模型 C-SVMXGBoost 比其他链式模型更优, 这里在原评价指标上扩展“改进程度”这一指标. 其中, 在覆盖率上的最小 (或最大) 改进程度为本文模型与其他链式模型差值的最小值 (或最大值); 其他评价指标上的改进程度计算为: 最小 (或最大) 改进程度为其他链式模型与本文模型在不同评价指标下的差值占其他链式模型比例的最小值 (或最大值).

由表 5 可知, 合理地选择链式融合模型可显著地提升测试用例生成效率.

表 5　本文模型与其他链式模型在不同程序上的各指标对比

待测程序	目标路径数	评价指标	对比链式模型						改进程度 (%)	
			C-SVMRF	C-SVMBP	C-XGBRF	C-XGBBP	C-RFBP	C-SVMXGBoost	最小	最大
PG1	2	覆盖率 (%)	100	100	100	100	100	100	**0**	**0**
		进化代数总量	904	897	922	**895**	**937**	875	**2.23**	**6.62**
		平均进化代数	6.81	**6.63**	6.97	6.85	**8.19**	6.23	**6.03**	**23.91**
		执行时间 (s)	58.43	**58.27**	61.24	60.86	**65.22**	57.27	**1.70**	**12.19**
PG3	2	覆盖率 (%)	100	100	100	100	100	100	**0**	**0**
		进化代数总量	**4990**	5023	5231	5206	**8442**	5051	**−1.22**	**40.17**
		平均进化代数	**34.27**	35.32	37.10	36.78	**65.31**	35.63	**−4.00**	**45.44**
		执行时间 (s)	**62.89**	63.02	65.11	65.07	**74.42**	64.40	**−2.40**	**13.46**
PG6	57	覆盖率 (%)	92	90	95	**95**	**79**	100	**5**	**21**
		进化代数总量	132714	148017	121736	**117969**	**249326**	89534	**24.10**	**64.09**
		平均进化代数	124.63	129.12	94.51	**93.13**	**253.29**	65.10	**30.10**	**74.30**
		执行时间 (s)	**257.00**	270.33	261.54	260.19	**284.02**	163.72	**36.30**	**42.36**
PG8	123	覆盖率 (%)	77	75	**86**	83	**68**	95	**9**	**27**
		进化代数总量	539480	557125	**329419**	371842	**591075**	195410	**40.68**	**66.94**
		平均进化代数	309.74	312.54	**190.53**	225.18	**453.20**	98.69	**48.20**	**78.22**
		执行时间 (s)	862.15	872.32	**845.95**	850.74	**924.82**	807.83	**4.51**	**13.65**
PG10	20	覆盖率 (%)	**85**	82	79	75	**69**	97	**12**	**28**
		进化代数总量	**91472**	115745	136172	145975	**227101**	50230	**45.09**	**77.88**
		平均进化代数	**182.65**	231.09	268.34	291.80	**388.22**	70.33	**61.49**	**81.88**
		执行时间 (s)	**134.08**	141.62	137.48	145.51	**160.26**	117.11	**12.66**	**26.93**

注: 本文链式模型C-SVMXGBoost (用灰底表示)与其他链式模型在各评价指标上的最小改进程度用**粗体**表示, 而最大改进程度用**粗体加下划线**表示, 并在对比链式模型的对应数据上用同样方式进行标注

对于简单程序 PG1 和 PG3, 这 6 种链式模型在覆盖率方面均达到 100%, 但在进化代数总量、平均进化代数和执行时间方面, 链式模型 C-RFBP 相比于其他模型结果较差. 与个别模型相比, 本文模型 C-SVMXGBoost 优势不明显. 例如, 在程序 PG1 上, 本文模型略优于 C-SVMBP 与 C-XGBBP; 在程序 PG3 上, 本文模型仅次于 C-SVMRF 模型, 是由于该程序输入数据维度较高, 在 C-SVMRF 链式模型中仅选择 SVM 子模型即可达到较高精度, 但 SVM 与 RF 在求解因存在难覆盖节点而导致样本不均衡问题上均不及 XGBoost, 故在我们的链式模型 C-SVMXGBoost 中包含了 XGBoost 子模型, 这需花费一定的初始融合时间, 而且由于 PG3 程序本身较为简单, 难覆盖节点较少, XGBoost 在该程序上的作用不明显, 从而导致本文模型在 PG3 程序上比 C-SVMRF 模型稍差.

对于稍复杂程序 PG6, PG8 和 PG10, 链式模型 C-SVMXGBoost 在 4 种评价指标下均优于其他几种模型, 且优势明显; 在程序 PG6 和 PG8 中, 链式模型 C-XGBRF 和 C-XGBBP 的结果优于 C-SVMRF, C-SVMBP 和 C-RFBP; 在程序 PG10 中, 链式模型 C-SVMRF 和 C-SVMBP 的结果优于 C-XGBRF, C-XGBBP 和 C-RFBP. 对比以上几种模型在这些程序上的改进程度, C-SVMXGBoost 的覆盖率最小可提升 5%, 最大可提升 28%; 进化代数总量上, 最小改进程度为 24.10%, 最大改进程度可达 77.88%; 平均进化代数上, 最小改进程度为 30.10%, 最大改进程度可达 81.88%; 执行时间上, 最小改进程度为 4.51%, 最大改进程度可达 42.36%. 得以改进的主要原因是, 无论是针对输入数据类型为数值型程序 (如 PG6), 还是针对非数值型程序 (如 PG8), 在链式融合过程中, 将 SVM 和 XGBoost 分别作为初始预训练模型, 使得链式模型精度较高, 为模型融合节省时间; 而分别将 SVM, XGBoost 与 RF, BP 两两融合的模型不能很好地应对上述程序特点, 对于涉及高维数据处理以及样本不均衡等问题的测试程序, 不能准确地将其路径节点状态分类. 因此, 相比于将 SVM 与 XGBoost 融合的 C-SVMXGBoost 模型, 其他链式模型的精度有所下降, 在遗传变异过程中, 使得路径预测的准确率降低. 此外, 由于融合 RF 和 BP 的链式模型精度较低, 导致需进行插桩验证的次数增多, 同时排除了较多的优秀个体, 从而使得进化代数总量、平均进化代数、执行时间也远多于本文的链式模型 C-SVMXGBoost. 所以, 为适应不同测试程序的特征以及需求, 选择合适的子模型进行链式融合对提升测试效率十分关键.

5.3.5 本文方法可靠性统计分析

为更好地说明本文方法与传统遗传法实验结果的可靠性, 本节借用 SPSS24 (一款统计分析软件), 通过假设检验方法中的方差检验 (F 检验, 显著性水平: α=0.05) 进行显著性分析. 将每个待测程序独立运行 100 次, 记录目标个体首次出现的进化代数, 将传统遗传法与本文方法下获得的该数据作为两组样本来计算 F 统计量. 即, 组间均方与组内均方的比, 该值能验证随机因素对实验结果造成的影响程度. 原假设 H_0 为各组平均数相等, 无显著性差异. 检验结果如表 6 所示.

表 6 检验结果

待测程序	F值	检验结果
PG1	40.75	拒绝H_0
PG2	18.65	拒绝H_0
PG3	12.76	拒绝H_0
PG4	44.21	拒绝H_0
PG5	16.34	拒绝H_0
PG6	24.40	拒绝H_0
PG7	28.33	拒绝H_0
PG8	29.65	拒绝H_0
PG9	42.88	拒绝H_0
PG10	15.74	拒绝H_0

由表 6 可知, 以上几种待测程序的 F 值均远大于界值 3.04. 这从统计学角度可验证: 排除随机因素, 相较于传统遗传法, 本文方法通过融入链式模型与重用优秀测试用例可使得种群个体朝目标路径方向加速演化, 进而提高待测程序的测试效率, 体现出本文方法的有效性.

5.4 与其他经典方法的比较

前文与传统遗传法进行对比, 验证了本文方法的有效性. 为进一步阐明本文将代理模型用于遗传算法在多目标路径测试用例生成方面的优势, 选取一些利用遗传算法或通过构建代理模型生成测试用例的经典方法设计对比实验, 包括关键字流图法[7]、关键点概率法[15]、否定遗传法[14]、神经网络法[8]和 SAEO 法[9]. 这些方法涉及单目标路径或多目标路径的测试用例生成方面, 与本文方法的研究方向一致, 代表性较强. 下面比较本文方法与这些对比方法在程序 PG4, PG6, PG9, PG10, PG11, PG12, PG13 上的测试效果, 对比结果如表 7 所示. 为更直观地展示在各评价指标下, 本文方法比其他对比方法提升的程度, 对原指标进行扩展, 增加“最大覆盖率差值” (即本文方法与各对比方法在待测程序上覆盖率之差的最大值) 和“平均进化代数差值比例” (对比方法与本文方法的平均进化代数之差占对比方法的比例) 这两个指标.

由表 7 可知, 相较于其他几种方法, 本文方法在覆盖率方面较优, 在平均进化代数方面也具有一定优势. 其中, 在程序 PG4 和 PG6 上, 对比关键字流图法, 本文方法与该方法的覆盖率均达到 100%; 平均进化代数差值比例分别为–6.11 和–6.28, 这表明本文方法需要更多迭代次数. 但是, 从覆盖路径数方面, 本文方法可覆盖更多的难覆盖目标路径, 而关键字流图法仅能覆盖单条路径.

在程序 PG9 和 PG10 上, 对比关键点概率法, 本文方法与该方法的覆盖率均为 100%; 平均进化代数差值比例分别为 0.99 和 0.98, 这表明本文方法所需进化代数更少. 对比否定遗传法, 在程序 PG9 上, 本文方法与该方法覆盖率差值最大, 差值为 3; 在这两种程序上的最小平均进化代数差值比例分别为 0.86 和 0.65.

对比神经网络法, 在程序 PG10 和 PG11 上, 均达到最大覆盖率差值, 分别为 13 和 15. 这表明本文方法较神经网络法, 在覆盖率方面提升较大. 对比 SAEO 法, 在程序 PG12 和 PG13 上, 该方法的覆盖率分别为 100% 和 85%, 本文方法的覆盖率分别为 100% 和 94%, 在程序 PG13 上可提升 9%. 这说明, 对比其他两种代理模型, 本文模型

C-SVMXGBoost 的预测结果更准确, 且对不同测试程序均有较强适应性.

从以上几种对比方法的结果可看出, 本文方法在规模较小的程序中优势不太明显, 而在规模较大的程序中优势较为明显. 原因是, 虽然本文模型在不同程序上精度均较高, 但是较小规模程序的分支节点数相对较少, 通过链式模型预测此类程序的覆盖路径, 在遗传过程中较优个体被排除的概率反而会增加, 这使得测试效率不太高; 而较大规模的程序则相反, 较优个体被排除的概率会降低, 链式模型可提高覆盖率和减少进化代数.

表 7　6 种方法在不同程序上的多指标对比

评价指标	对比方法														
	待测程序	关键字流图法[7]	本文方法	待测程序	关键点概率法[15]	本文方法	待测程序	否定遗传法[14]	本文方法	待测程序	神经网络法[8]	本文方法	待测程序	SAEO法[9]	本文方法
目标路径数		1	21		30	12		125	12		30	20		119	20
覆盖率 (%)	PG4	100	100	PG9	100	100	PG9	97	100	PG10	84	97	PG12	100	100
平均进化代数		6.14	43.63		6292	28.13		200	28.13		—	70.33		—	69.04
目标路径数		1	57		20	20		233	20		100	25		227	30
覆盖率 (%)	PG6	100	100	PG10	100	97	PG10	86	97	PG11	85	100	PG13	85	94
平均进化代数		8.94	65.10		5947	70.33		200	70.33		—	82.19		—	125.71
最大覆盖率差值		0			0			13			15			9	
平均进化代数差值比例 最少		–6.28			0.98			0.65			—			—	
平均进化代数差值比例 最多		–6.11			0.99			0.86			—			—	

注: ① 因PG9–PG13为非Python代码程序, 为便于本文方法与其他经典方法比较, 此处选取它们的部分程序统一转换成Python代码进行实验; ② 表中数据均来源于各文献的原始数据, 由于不同对比方法选择实验对象不同, 故存在部分程序缺少实验数据的情况

5.5　链式模型性能与用例生成方法效率分析

上面已对不同链式模型以及测试用例生成方法进行了实验分析, 本节基于这些分析结果从不同角度归纳各模型与用例生成方法的性能.

C-SVMXGBoost 的性能是影响本文方法的关键因素. 为综合评价本文构建的链式模型性能, 更直观地对比不同模型在用例生成方面的优劣, 这里根据第 5.3.4 节表 5 的实验结果对链式模型的性能进行总结, 如表 8 所示.

表 8　不同链式模型的性能对比

链式模型名称	时间消耗	分类精度	用例生成效果
C-SVMRF	△△	△△△	△△△
C-SVMBP	△△△	△△	△△
C-XGBRF	△△△	△△△	△△△
C-XGBBP	△△△	△△	△△
C-RFBP	△△△△	△	△
C-SVMXGBoost	**△**	**△△△△**	**△△△△**

注: 标记“△”表示不同链式模型在不同指标下的性能表现程度, 标记越多则程度越深

由表 8 可知, 在时间方面, 耗时最多的是链式模型 C-RFBP, 其次是 C-SVMBP, C-XGBRF, C-XGBBP, C-SVMRF, 最少的是本文方法构建的 C-SVMXGBoost. 在分类精度和用例生成效果方面, 最高的是 C-SVMXGBoost, 其次是 C-SVMRF, C-XGBRF, C-SVMBP, C-XGBBP, 最低的是 C-RFBP. 根据以上对比结果, 本文方法采用的模型综合性能最好.

本文融合链式模型模拟预测路径, 同时寻找更多的相似目标路径, 实现多路径测试用例生成. 为综合评价本文

方法在测试用例生成方面的效率, 这里结合表 3 (见第 5.3.3 节) 和表 7 的实验结果, 分别从覆盖率、平均进化代数和路径覆盖数这几个方面, 对几种测试用例生成方法展开定性分析, 如表 9 所示.

表 9 各测试用例生成方法的效率对比

对比方法	覆盖率	平均进化代数	路径覆盖数
传统遗传法	◇	◇◇◇◇	◇
关键字流图法[7]	◇◇◇	◇	◇
关键点概率法[15]	◇◇◇	◇◇◇	◇◇◇
否定遗传法[14]	◇◇	◇◇◇	◇◇◇
神经网络法[8]	◇◇	—	◇
SAEO法[9]	◇◇◇	—	◇◇◇
本文方法	**◇◇◇**	**◇◇**	**◇◇◇**

注: 标记“◇”表示不同方法在不同指标下的效率, 标记越多则影响越大

由表 9 可知, 传统遗传法表现最差. 这些方法中, 路径覆盖率最高的有关键字流图法、关键点概率法、SAEO 法和本文方法, 其次是否定遗传法和神经网络法. 平均进化代数效果最优的是关键字流图法, 其次是本文方法, 然后是关键点概率法和否定遗传法. 目标路径被覆盖的数量也是衡量方法优劣的关键因素之一. 在众多方法中, 关键点概率法、否定遗传法、SAEO 法和本文方法均可覆盖多条目标路径, 而关键字流图法以及神经网络法解决的均为单目标路径, 若检测更多路径则需重复实验多次, 这不仅耗费更多时间, 且造成测试数据的浪费.

综上分析, 本文方法保证覆盖率的同时, 在相对较少的进化代数内可覆盖较多的目标路径, 这表明本文方法是有效的, 且测试效率较好.

6 总结与进一步研究

本文提出一种融合 SVM 和 XGBoost 的链式模型 C-SVMXGBoost 的多路径覆盖测试数据生成与重用方法, 该链式模型代替插桩法模拟测试数据覆盖路径, 减少插桩时间的同时, 筛选尽可能多的相似目标路径, 提高测试用例利用率及路径覆盖效率. 相比其他单一模型和融合模型, 本文模型在精度与时间方面具有较大优势, 精度基本保持在 90% 及以上, 执行时间普遍减少, 最多可减少 42.36%. 同时在遗传进化时, 本文方法对符合要求的个体插桩验证, 计算适应度, 并在交叉变异时引入优秀用例. 这不仅提高种群质量, 还充分利用测试数据实现多路径覆盖用例生成. 对比已有经典测试方法, 本文方法有较高用例生成效率, 在覆盖率上提高可达 15%, 平均进化代数降低可达 65%. 未来工作将探讨如何利用机器学习方法优化初始遗传种群质量, 以期进一步提高测试效率.

References:

[1] Bertolino A, Miranda B, Pietrantuono R, Russo S. Adaptive test case allocation, selection and generation using coverage spectrum and operational profile. IEEE Trans. on Software Engineering, 2021, 47(5): 881–898. [doi: 10.1109/TSE.2019.2906187]

[2] Sun CA, Liu BL, Fu A, Liu YQ, Liu H. Path-directed source test case generation and prioritization in metamorphic testing. Journal of Systems and Software, 2022, 183: 111091. [doi: 10.1016/j.jss.2021.111091]

[3] Liao WZ, Xia XY, Jia XJ. Test data generation for multiple paths coverage based on ant colony algorithm. Acta Electronica Sinica, 2020, 48(7): 1330–1342 (in Chinese with English abstract). [doi: 10.3969/j.issn.0372-2112.2020.07.011]

[4] Feng XB, Ding R, Chai BJ, Huo TT. Multi-objective heuristic information optimization algorithm for path coverage-oriented test data generation. In: Proc. of the 3rd Int'l Conf. on Artificial Intelligence and Advanced Manufacture. Manchester: ACM, 2021. 650–654. [doi: 10.1145/3495018.3495135]

[5] Fan SP, Wan L, Yao NM, Zhang Y, Ma BY. Test case sorting method based on key use cases extracted. Acta Electronica Sinica, 2022, 50(1): 149–156 (in Chinese with English abstract). [doi: 10.12263/DZXB.20201284]

[6] Pan F, Gong DW, Tian T, Yao XJ, Li Y. Path similarity-based scheduling sequence sorting for multi-path coverage of parallel programs. SCIENTIA SINICA Informationis, 2021, 51(4): 565–581 (in Chinese with English abstract). [doi: 10.1360/SSI-2019-0113]

[7] Qian ZS, Song T. Reuse of test cases between similar programs based on keyword flow graph. Ruan Jian Xue Bao/Journal of Software, 2021, 32(9): 2691–2712 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5984.htm [doi: 10.13328/j.cnki.jos.005984]

[8] Yao XJ, Gong DW, Li B. Evolutional test data generation for path coverage by integrating neural network. Ruan Jian Xue Bao/Journal of Software, 2016, 27(4): 828–838 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4973.htm [doi: 10.13328/j.cnki.jos.004973]

[9] Gong DW, Sun BC, Yao XJ, Tian T. Test data generation for path coverage of MPI programs using SAEO. ACM Trans. on Software Engineering and Methodology, 2021, 30(2): 17. [doi: 10.1145/3423132]

[10] Qian ZS, Yu QY, Song T, Zhu YM, Zhu J, Zhao C. Test case generation and reuse based on support vector machine regression model. Acta Electronica Sinica, 2021, 49(7): 1386–1391 (in Chinese with English abstract). [doi: 10.12263/DZXB.20200426]

[11] Chen TM, Yang YM, Chen B. Maldetect: An Android malware detection system based on abstraction of Dalvik instructions. Journal of Computer Research and Development, 2016, 53(10): 2299–2306 (in Chinese with English abstract). [doi: 10.7544/issn1000-1239.2016.20160348]

[12] Esteves G, Figueiredo E, Veloso A, Viggiato M, Ziviani N. Understanding machine learning software defect predictions. Automated Software Engineering, 2020, 27(3): 369–392. [doi: 10.1007/s10515-020-00277-4]

[13] Di Nucci D, Panichella A, Zaidman A, De Lucia A. A test case prioritization genetic algorithm guided by the hypervolume indicator. IEEE Trans. on Software Engineering, 2020, 46(6): 674–696. [doi: 10.1109/TSE.2018.2868082]

[14] Xia CY, Zhang Y, Wan L, Song Y, Xiao N, Guo B. Test data generation of path coverage based on negative selection genetic algorithm. Acta Electronica Sinica, 2019, 47(12): 2630–2638 (in Chinese with English abstract). [doi: 10.3969/j.issn.0372-2112.2019.12.024]

[15] Qian ZS, Zhu J, Zhu YM, Yu QY, Li DM, Song J. Multi-path coverage strategy combining key point probability and path similarity. Ruan Jian Xue Bao/Journal of Software, 2022, 33(2): 434–454 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6149.htm [doi: 10.13328/j.cnki.jos.006149]

[16] Qi RZ, Wang ZJ, Huang YH, Li SY. Generating combinatorial test suite with spark based parallel approach. Chinese Journal of Computers, 2018, 41(6): 1284–1299 (in Chinese with English abstract). [doi: 10.11897/SP.J.1016.2018.01284]

[17] Skocelas KG, DeVries B. Test data generation for recurrent neural network implementations. In: Proc. of the 2020 IEEE Int'l Conf. on Electro Information Technology. Chicago: IEEE, 2020. 469–474. [doi: 10.1109/EIT48999.2020.9208306]

[18] Jiang HY, Zong M, Liu XY. Research of software defect prediction model based on ACO-SVM. Chinese Journal of Computers, 2011, 34(6): 1148–1154 (in Chinese with English abstract). [doi: 10.3724/SP.J.1016.2011.01148]

[19] Pradhan D, Wang S, Ali S, Yue T, Liaaen M. CBGA-ES^{+}: A cluster-based genetic algorithm with non-dominated elitist selection for supporting multi-objective test optimization. IEEE Trans. on Software Engineering, 2021, 47(1): 86–107. [doi: 10.1109/TSE.2018.2882176]

[20] Bhati BS, Chugh G, Al-Turjman F, Bhati NS. An improved ensemble based intrusion detection technique using XGBoost. Trans. on Emerging Telecommunications Technologies, 2021, 32(6): e4076. [doi: 10.1002/ett.4076]

[21] Yang HY, Xu J. Android malware detection based on improved random forest. Journal of Communications, 2017, 38(4): 8–16 (in Chinese with English abstract). [doi: 10.11959/j.issn.1000-436x.2017073]

附中文参考文献:

[3] 廖伟志, 夏小云, 贾小军. 基于蚁群算法的多路径覆盖测试数据生成. 电子学报, 2020, 48(7): 1330–1342. [doi: 10.3969/j.issn.0372-2112.2020.07.011]

[5] 范书平, 万里, 姚念民, 张岩, 马宝英. 基于关键用例获取的测试用例排序方法. 电子学报, 2022, 50(1): 149–156. [doi: 10.12263/DZXB.20201284]

[6] 潘峰, 巩敦卫, 田甜, 姚香娟, 李吟. 基于路径相似度的并行程序多路径覆盖调度序列排序. 中国科学: 信息科学, 2021, 51(4): 565–581. [doi: 10.1360/SSI-2019-0113]

[7] 钱忠胜, 宋涛. 面向关键字流图的相似程序间测试用例的重用. 软件学报, 2021, 32(9): 2691–2712. http://www.jos.org.cn/1000-9825/5984.htm [doi: 10.13328/j.cnki.jos.005984]

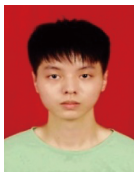
[8] 姚香娟, 巩敦卫, 李彬. 融入神经网络的路径覆盖测试数据进化生成. 软件学报, 2016, 27(4): 828–838. http://www.jos.org.cn/1000-9825/4973.htm [doi: 10.13328/j.cnki.jos.004973]

[10] 钱忠胜, 俞情媛, 宋涛, 朱懿敏, 祝洁, 赵畅. 基于支持向量机回归模型的测试用例生成与重用. 电子学报, 2021, 49(7): 1386–1391. [doi: 10.12263/DZXB.20200426]

[11] 陈铁明, 杨益敏, 陈波. Maldetect: 基于Dalvik指令抽象的Android恶意代码检测系统. 计算机研究与发展, 2016, 53(10): 2299–2306.

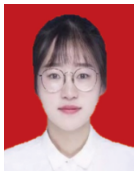
[doi: 10.7544/issn1000-1239.2016.20160348]
[14] 夏春艳, 张岩, 万里, 宋妍, 肖楠, 郭冰. 基于否定选择遗传算法的路径覆盖测试数据生成. 电子学报, 2019, 47(12): 2630–2638. [doi: 10.3969/j.issn.0372-2112.2019.12.024]
[15] 钱忠胜, 祝洁, 朱懿敏, 俞情媛, 李端明, 宋佳. 结合关键点概率与路径相似度的多路径覆盖策略. 软件学报, 2022, 33(2): 434–454. http://www.jos.org.cn/1000-9825/6149.htm [doi: 10.13328/j.cnki.jos.006149]
[16] 戚荣志, 王志坚, 黄宜华, 李水艳. 基于Spark的并行化组合测试用例集生成方法. 计算机学报, 2018, 41(6): 1284–1299. [doi: 10.11897/SP.J.1016.2018.01284]
[18] 姜慧研, 宗茂, 刘相莹. 基于ACO-SVM的软件缺陷预测模型的研究. 计算机学报, 2011, 34(6): 1148–1154. [doi: 10.3724/SP.J.1016.2011.01148]
[21] 杨宏宇, 徐晋. 基于改进随机森林算法的Android恶意软件检测. 通信学报, 2017, 38(4): 8–16. [doi: 10.11959/j.issn.1000-436x.2017073]

钱忠胜(1977－), 男, 博士, 教授, 博士生导师, CCF 专业会员, 主要研究领域为软件工程, 机器学习, 智能化软件.



姚昌森(1998－), 男, 硕士生, 主要研究领域为软件工程, 机器学习.

俞情媛(1997－), 女, 博士生, CCF 学生会员, 主要研究领域为软件工程, 机器学习.



秦朗悦(1998－), 女, 硕士生, 主要研究领域为软件工程, 机器学习.

张丁(1994－), 女, 硕士生, 主要研究领域为软件工程, 机器学习.

成轶伟(1996－), 男, 硕士, 主要研究领域为软件工程, 机器学习.