软件学报 ISSN 1000-9825, CODEN RUXUEW
[doi: 10.13328/j.cnki.jos.007472] [CSTR: 32375.14.jos.007472]

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

结合 K-means 与 SVR 的多路径覆盖测试用例约简与生成*

钱忠胜, 俞情媛, 范赋宇, 许克文, 陈 超

(江西财经大学 计算机与人工智能学院, 江西 南昌 330013)

通信作者: 钱忠胜, E-mail: changesme@163.com

摘 要: 结合机器学习相关技术的启发式测试用例生成方法可显著提高测试效率. 已有研究关注于利用部分测试用例构建高效的代理模型, 忽略了初始种群质量以及代理模型对多路径测试效率的影响. 由此, 提出一种结合 K-means 与 SVR (support vector machine regression, 支持向量机回归) 的测试用例约简与生成方法. 通过 K-means 将随机生成的用例聚为若干簇, 保留与簇中心距离在一定阈值内的用例, 生成这些用例的路径覆盖矩阵. 利用该矩阵评估测试用例的路径覆盖潜能以及路径的难易覆盖程度, 并基于这两者对测试用例进行排序, 分别从不同簇中选取若干用例构成测试用例约简集, 将其作为初始遗传种群. 这不仅增强初始种群的多样性, 降低其冗余性, 还有助于减少多路径覆盖的测试用例进化次数. 同时, 将聚类前的用例及其适应度作为样本训练适应于多路径覆盖的 SVR 适应度预测模型, 并使用遗传进化生成的新用例更新模型, 进一步提高模型精度, 可减少执行插桩程序带来的大量时间消耗. 这样, 种群质量与测试效率均得以提升. 实验表明, 在 15 个程序上, 所提方法在覆盖率、平均进化代数等指标上均有较好改善. 其中在覆盖率上, 与 3 类基准方法相比, 最少可提高 7%, 最多可达 49%; 与 5 种具有竞争性的方法相比, 可提高约 10%, 最多可达 25%. 所提方法对融合机器学习的多路径测试研究提供了方法指导.

关键词: 测试用例; K-means 聚类; 支持向量机回归模型; 多路径覆盖; 遗传算法

中图法分类号: TP311

中文引用格式: 钱忠胜, 俞情媛, 范赋宇, 许克文, 陈超. 结合K-means与SVR的多路径覆盖测试用例约简与生成. 软件学报. http://www.jos.org.cn/1000-9825/7472.htm

英文引用格式: Qian ZS, Yu QY, Fan FY, Xu KW, Chen C. Test Case Reduction and Generation of Multi-path Coverage Based on K-means and SVR. Ruan Jian Xue Bao/Journal of Software (in Chinese). http://www.jos.org.cn/1000-9825/7472.htm

Test Case Reduction and Generation of Multi-path Coverage Based on K-means and SVR

QIAN Zhong-Sheng, YU Qing-Yuan, FAN Fu-Yu, XU Ke-Wen, CHEN Chao

(School of Computing and Artificial Intelligence, Jiangxi University of Finance and Economics, Nanchang 330013, China)

Abstract: The heuristic test case generation method that combines machine learning-related technologies can significantly improve the test efficiency. Existing studies focus on building efficient surrogate models with partial test cases, but ignore the influence of both the initial population quality and surrogate models on the multi-path testing efficiency. Therefore, this study proposes a test case reduction and generation method combining K-means and support vector machine regression (SVR). The randomly generated test cases are clustered into several clusters by adopting K-means, and only the test cases that are within a particular distance away from the cluster center are retained, with the path coverage matrix for these test cases constructed. This matrix is employed to evaluate the path coverage potential of test cases and the coverage difficulty of paths. Additionally, based on these two conditions, the test cases are ranked, and several test cases are selected from different clusters to construct the test case reduction set, which is taken as the initial genetic population. This not only increases the diversity of the initial population and reduces its redundancy, but also helps to reduce the iteration number for multi-path

* 基金项目: 国家自然科学基金 (62262025); 赣鄱俊才支持计划主要学科学术和技术带头人培养项目 (20243BCE51024); 江西省自然科学基金重点项目 (20224ACB202012)
收稿时间: 2024-10-03; 修改时间: 2025-03-25, 2025-05-06; 采用时间: 2025-05-26; jos 在线出版时间: 2026-01-21

coverage test cases. Meanwhile, the test cases before clustering and their fitness are employed as the samples to train the SVR fitness prediction model designed for multi-path coverage, and then the new test cases generated by genetic evolution are utilized to update the model, thus improving the model accuracy and reducing the time consumed due to the instrumentation program execution. In this way, both population quality and test efficiency can be improved. The experimental results show that on fifteen programs, the proposed method has better improvements in terms of indicators such as the coverage rate and average evolutionary generation. Specifically, in terms of the coverage rate, the proposed method demonstrates an improvement of at least 7% and up to 49% compared to three types of baseline methods, and shows the enhancement of approximately 10% to a maximum of 25% compared to five competitive methods. The proposed method provides guidance for the research on multi-path testing that combines machine learning.

Key words: test case; K-means clustering; support vector machine regression (SVR) model; multi-path coverage; genetic algorithm

1 引 言

近来, 基于搜索的启发式测试用例生成研究十分广泛[1−3]. 在众多进化算法中, 遗传算法作为一种全局搜索方法, 在软件测试中取得了丰硕成果[4,5], 合理地应用该算法可高效地筛选与生成目标用例.

传统进化生成算法效率较低[6], 通常利用随机生成的测试用例作为初始种群, 并通过演化生成目标测试用例. 此外, 在选择较优种群个体时, 每个个体均需执行程序以评估它们的优劣, 这对于规模较大、存在数量较多且较难覆盖路径的程序而言, 必将会花费大量测试时间[7].

不少学者通过改进遗传算法提高用例生成效率. 就已有研究来看, 改进种群遗传进化方向, 是提高用例生成效率的关键. 为更好地引导个体朝向目标用例进化, 有研究者设计适应度函数来评估个体覆盖情况, 以尽早筛选出可覆盖目标路径的用例进行演化[5]. 利用已覆盖的分支路径信息也可引导和改善种群个体进化方向[8]. 相比之下, K-means 方法作为一种高效的相似性挖掘策略, 将其应用于测试领域可挖掘用例间的关系[9−11], 有助于提高优秀个体被选中参与进化生成的概率, 加快目标用例的生成. 这些研究虽可改进种群进化方向, 但每次进化后的个体均需执行程序, 而且由于部分路径较难覆盖, 则种群个体需不断地演化与执行程序, 这带来了较高的执行时间成本. 为减少执行程序带来的时间消耗, 利用机器学习构建代理模型是一种有效途径[12]. 近来, 不少学者利用机器学习相关技术构建代理模型以减少测试时间[13,14], 包括神经网络、SVM 等. 但他们往往只针对单路径的个体适应度预测, 若生成多目标路径覆盖的用例则需构建多个代理模型, 这十分耗时且低效. 此外, 由于训练数据的限制, 他们使用的初始代理模型难以用于部分难覆盖路径, 模型更新也复杂且耗时. 因此, 构建轻量级的适应度预测模型有助于提升测试效率.

然而, 现有研究均忽略了初始种群对测试用例进化生成的影响, 未从源头缓解进化生成缓慢的问题. 已有研究中采用的随机初始种群往往质量较低且冗余, 这制约了用例的进化生成, 且降低了测试稳定性. 另外, 初始种群中若相似用例过多, 则难以引入其他类型个体基因, 即使通过改进种群进化方向也很难快速优化个体基因, 还可能陷入局部最优. 而且, 代理模型作为减少执行程序时间的一种有效工具, 难以改善种群进化方向. 此外, 这些基于适应度预测模型的用例生成方法只针对单目标路径[7,13,14], 且它们依赖于初始代理模型. 相较于单路径的代理模型, 构建可适用于多目标路径的代理模型在用例生成方面具有更高的适用性与效率. 在工业领域中, 如目标路径冗余度较高的信息验证与管理系统、安全需求严格的航海系统等, 它们功能复杂、模块较多, 通过增强初始种群质量与多样性, 以及构建代理模型有助于提升它们的测试效率.

由此可见, 一方面, 初始种群质量影响进化生成目标用例的进程. 提高初始种群的质量及多样性, 可引导个体更好地进化生成, 减少迭代次数; 另一方面, 种群个体的评估效率也是影响测试的一个关键因素. 构建高效的多路径个体适应度代理模型可有效提高种群个体的评估效率. 在众多机器学习方法中, K-means 策略可高效地将测试用例进行聚类, 有利于挖掘与分析用例间的关系, 提高种群质量与多样性; SVR 模型可快速且精确地构建测试用例与其对应适应度间的关系, 有助于增强个体评估能力, 提升多目标路径覆盖的测试用例生成效率.

基于此, 本文提出一种结合 K-means 方法与 SVR 模型的多路径测试用例生成策略, 主要解决以下几个问题.

1) 挖掘用例间的潜在相似度, 约简冗余数据. 聚集的用例具有覆盖相似路径的潜能, 利用 K-means 聚类方法挖掘测试用例间的联系有助于筛选可覆盖多目标路径的用例. 另外, 随机生成的用例一般冗余性高, 尽可能增加与目标用例距离相近的用例, 可提高优秀用例的选中概率, 进而加快生成可覆盖多目标路径的测试用例.

2) 提高个体适应度计算效率, 减少时间消耗. 种群个体适应度计算是进化生成目标用例的关键, 通常情况下, 此过程较为耗时. 利用 SVR 构建高效的多路径个体适应度预测模型可有效降低多路径用例生成的时间成本. 而且, SVR 的轻量级特性也为模型的快速更新提供支撑.

3) 融合多角度的测试用例生成方法, 提高测试效率. 相较于仅从单一角度考虑的测试用例生成方法, 多角度分析对测试效率的提升比较明显, 不过需兼顾各角度间的互补性与互斥性. K-means 方法与 SVR 预测模型可相辅相成, 两者的有效结合能提高用例生成效率.

2 相关工作

进化算法有助于生成测试用例、最小化测试用例套件, 已被广泛应用于测试中. 目前, 已有较多研究者已从不同角度优化进化算法, 有效地提高了测试效率.

钱忠胜等人[15]根据路径覆盖的关键点概率设计适应度函数引导个体进化, 生成可覆盖多路径的测试用例. Di Nucci 等人[16]基于超体积的遗传算法提升用例选择与排序的效率, 但对于高冗余测试嵌套系统, 该方法性能有所下降. Amarif 等人[17]将遗传算法引入 JUnit 测试框架中, 以期在单元测试阶段实现自动化测试. Ji 等人[18]将粒子群优化理论引入遗传算法中, 减少进化生成中因随机性带来的影响. Wang 等人[19]利用量子退火进化算法解决经典软件上的测试用例最小化问题. 此外, 测试用例的执行路径以及已覆盖路径是指导用例生成的关键信息. Wang 等人[1]根据两种不同路径结构的函数设计多任务路径覆盖模型, 并提出一种求解该模型的多因子优化框架. Guo 等人[3]提出一种基于路径结构矩阵和主折叠搜索的用例自动生成策略.

可见, 进化算法, 尤其是遗传算法, 作为一种优化策略, 在进化生成测试数据中展现出较强的适应性与优越性. 将路径覆盖等信息融入其中进行优化, 可有效提高用例生成质量. 但上述研究在个体评估时均需花费较多时间, 如应对功能复杂、规模较大的工业系统等. 而机器学习相关技术有助于挖掘用例间的关系, 减少种群进化次数及提高个体评估效率, 进而缩减测试时间.

聚类策略是一种可用于特征相似性挖掘的机器学习方法, 可很好地挖掘测试用例间的联系. Chen 等人[9]利用 K-means 设计频率变换和改进的小波变换方法, 更好地识别用例间的相似性, 缓解随机测试效率低下的问题, 但未考虑聚类簇的个数对测试效率的影响. Pei 等人[10]通过 K-means 方法将测试结果与距离相结合, 动态调整测试轮廓, 增加优秀个体被选中的概率, 但必须建立在完备的测试用例套件上. Pradhan 等人[20]将聚类思想用于遗传算法中, 降低父解中的随机性, 进而实现多目标用例生成. 但该方法忽略不同主题对算法的影响. Chen 等人[21]使用 K-medoids 算法对用例集进行更细粒度的分类, 并通过簇内贪婪排序提高早期测试用例的多样性.

另外, 构建合适的个体适应度预测模型也可显著提高测试效率. 钱忠胜等人[7]针对单目标路径构建代理模型, 并以此来代替传统插桩方式获取遗传进化中每个个体的适应度, 但若生成可覆盖多目标路径的用例则需构建若干代理模型. 他们还从测试路径角度[22]构建路径预测模型以减少基于覆盖路径的个体适应度计算时间. Gong 等人[12]根据多模态特性将程序样本聚为若干簇, 每个簇训练一个代理模型, 在进化过程中, 使用代理模型估计个体适应度, 这使得每一次评估个体均需对若干代理模型进行筛选, 此过程较为耗时. Gao 等人[14]构建全局和局部两个代理模型, 分层获取个体适应度值, 以减少适应度获取的计算开销. Skocelas 等人[23]提出一种利用 RANN 的测试用例生成方法, 降低测试的复杂性, 提高资源受限的嵌入式系统适用性.

然而, 机器学习方法有助于筛选簇中优秀用例, 降低冗余性. 此外, 利用这些技术构建代理模型可有效降低进化生成中带来的较多适应度获取时间. 但现有研究均忽略了种群质量对测试的影响. 此外, 他们构建的代理模型难以适用于多目标路径的用例生成. 因此, 改善初始种群质量, 构建可应用于多路径覆盖的代理模型, 有助于减少模型构建次数, 提升多路径测试生成效率.

3 基础理论

为便于理解, 以三角形判定程序为例 (如算法 1 所示), 简述 K-means 测试用例约简及 SVR 模型的基本理论.

算法 1. 三角形判定算法.

输入: 三角形的三条边, 分别为 a, b, c;
输出: 三角形类型判定.

```
Begin
1.   Def triangle(a, b, c):
2.     If a>0 and b>0 and c>0:
3.       instrument(p(1));
4.       If a+b>c and b+c>a and a+c>b:
5.         instrument(p(2));
6.         If a==b and b==c:
7.           instrument(p(3));
8.           return (“这是等边三角形”)
9.         Elif a==b or b==c or c==a:
10.          instrument(p(4));
11.          return (“这是等腰三角形”)
12.        Else:
13.          return (“这是不规则三角形”)
14.        End If
15.      Elif a+b==c or b+c==a or a+c==b:
16.        instrument(p(5));
17.        return (“这是个退化三角形”)
18.      Else:
19.        return (“这不是个三角形”)
20.      End If
21.    Else:
22.      return (“请输入大于 0 的数字”)
23.    End If
End
```

3.1 基于 K-means 聚类的测试用例约简

在测试中应尽可能地减少易覆盖路径的测试用例, 而将难覆盖路径的测试用例保留下来. 相比于其他聚类方法, K-means 聚类策略具有低复杂性、高效性等优势. 基于 K-means 聚类的测试用例约简方法通过挖掘用例间的相似性, 不仅可减少随机生成中的冗余数据, 促进目标用例的生成, 而且可增强用例的多样性, 实现多目标路径覆盖的用例生成.

对于随机生成的一组用例 $T=\{t_1,t_2,\ldots,t_n\}$, 计算它们与 k 个测试簇心 (C_1, C_2,…, C_k) 的欧氏距离, 并将它们分别聚合到最近簇中心所在的测试用例簇中. 距离计算如公式 (1) 所示:

$$D(t_i,C_j)=\sqrt{\sum_{a=1}^{m}(t_i^{(a)}-C_j^{(a)})^2} \tag{1}$$

其中, $t_i\,(1\leqslant i\leqslant n)$ 为第 i 个测试用例, $t_i^{(a)}(1\leqslant a\leqslant m)$ 为该测试用例的第 a 个输入分量; $C_j\,(1\leqslant j\leqslant k)$ 为第 j 个测试聚类中心, $C_j^{(a)}$ 为该中心的第 a 个输入分量.

为优化测试用例的聚类结果, 通过 *SSE* 指标衡量簇内测试用例到簇中心的均方误差. 当均方误差达到最小时, 聚类终止, 如公式 (2) 所示:

$$SSE=\sum_{j=1}^{k}\sum_{i=1}^{n}D(t_i,C_j)^2 \tag{2}$$

例 1: 假定存在一组三角形判定程序的测试用例 T={t_1=(1, 2, 3), t_2= (1, 2, 5), t_3=(4, 3, 5), t_4=(7, 9, 7), t_5=(7, 9, 9), t_6=(20, 1, 25)}, 根据公式 (1) 和 (2) 将它们聚为 3 个簇{t_1, t_2, t_3}、{t_4, t_5}与{t_6}, 它们的聚类中心分别为 C_1=(2, 3, 4.33), C_2=(7, 9, 8) 以及 C_3=(20, 1, 25).

聚类后的测试用例簇 *Clus* 质量会受噪声用例的干扰, 而噪声用例因与其他用例距离较远, 往往难以覆盖难覆盖的目标路径, 不利于测试用例的约简与选择. 而且, 将其引入初始种群制约了用例的进化与生成. 轮廓法[10]通过计算用例簇内的紧密程度以及簇间的分离程度, 可有效评估测试用例聚类质量. 因此, 我们利用轮廓法对聚类后的测试用例簇 *Clus* 中每一个用例 t_i 进行质量评估, 剔除噪声用例. 测试用例质量评估如公式 (3) 所示:

$$\begin{cases}Q_{t_i}=\dfrac{valueAvg_{t_i,Clus_{\text{clst}}}-valueAvg_{t_i,Clus_j}}{\max\{valueAvg_{t_i,Clus_j},valueAvg_{t_i,Clus_{\text{clst}}}\}}\\ \text{s.t. } valueAvg_{t_i,Clus_j}=avg_{t_i,Clus_j}(D(t_i,t_{i,\text{other}})),\ valueAvg_{t_i,Clus_{\text{clst}}}=avg_{t_i,Clus_{\text{clst}}}(D(t_i,t_{i,Clus_{\text{clst}}}))\end{cases} \tag{3}$$

其中, $valueAvg_{t_i,Clus_j}$ 为用例 t_i 与其所在簇 $Clus_j$ 中所有其他用例 $t_{i,\text{other}}$ 间的平均距离; $valueAvg_{t_i,Clus_{\text{clst}}}$ 为 t_i 到最近的其他簇内所有用例的平均距离; Q_{t_i} 为 t_i 的质量, 若其值在 (0, 1] 内, 则该个体在合适的簇中, 将其保留, 否则视为噪声, 将其剔除.

例 2: 根据公式 (3) 可得 $valueAvg_{t_1,Clus_1}=2.87$, $valueAvg_{t_1,Clus_{\text{clst}}}=10.52$, 故 $Q_{t_1}=0.73$. 依此方式, 其他 5 个测试用例的 Q 值分别为 0.74、0.53、0.77、0.79、0.00. 因 $Q_{t_6}=0.00$, 故将其视为噪声点并剔除. 剩余测试用例簇为{t_1, t_2, t_3}与{t_4, t_5}. 由 t_2=(1, 2, 5) 和 t_6=(20, 1, 25) 两个测试用例可知, 它们具有相同的测试功能, 即判定“这不是个三角形”. 但是, 相比于 t_6, 测试用例 t_2 通过将输入数据“1”进行一定轮次的进化即可生成能覆盖其他目标路径的用例; 而 t_6 的输入数据相差较远, 通过进化则难以生成. 此外, t_2 和 t_6 的覆盖路径往往较易覆盖, 故无需保留较多此类用例. 因此, 将 t_6 剔除, 可提高测试用例约简集质量, 优化遗传进化效率; 也可在一定程度上减少迭代次数, 缩减测试时间.

利用保留下来的测试用例执行程序, 生成路径覆盖矩阵 $\boldsymbol{P}_{\text{cov}}$, 如公式 (4) 所示:

$$\boldsymbol{P}_{\text{cov}}=\begin{bmatrix}p_1^{(1)} & p_1^{(2)} & \cdots & p_1^{(s)}\\ p_2^{(1)} & p_2^{(2)} & \cdots & p_2^{(s)}\\ \cdots & \cdots & \ddots & \cdots\\ p_r^{(1)} & p_r^{(2)} & \cdots & p_r^{(s)}\end{bmatrix} \tag{4}$$

矩阵的行表示一条完整路径 p_u=($p_u^{(1)}$, $p_u^{(2)}$,..., $p_u^{(l)}$,..., $p_u^{(s)}$), 其中, $p_u^{(l)}$ 表示第 u 条路径的第 l 个节点的状态 ($1\leqslant u\leqslant r$, $1\leqslant l\leqslant s$), 即用例经过或未经过 (记为 1 或 0) 某一节点.

例 3: 将剩余用例执行算法 1 的程序, 其中 t_1 的覆盖路径 p_1=(1, 0, 0, 0, 1), 其他用例的覆盖路径分别为: p_2=(1, 0, 0, 0, 0), p_3=(1, 1, 0, 0, 0), p_4=(1, 1, 0, 1, 0), p_5=(1, 1, 0, 1, 0). 因此, 可得如下路径覆盖矩阵 $\boldsymbol{P}_{\text{cov}}$.

$$\boldsymbol{P}_{\text{cov}}=\begin{bmatrix}1 & 0 & 0 & 0 & 1\\ 1 & 0 & 0 & 0 & 0\\ 1 & 1 & 0 & 0 & 0\\ 1 & 1 & 0 & 1 & 0\\ 1 & 1 & 0 & 1 & 0\end{bmatrix}.$$

已覆盖路径与目标路径间的相似性可体现出测试用例覆盖目标路径的能力以及路径的难易覆盖程度. 通过衡量用例能力及路径难易覆盖程度, 有助于测试用例的约简, 筛选出较强潜在覆盖能力、可覆盖难覆盖路径的测试

用例. 路径相似度计算如公式 (5), 多路径相似矩阵如公式 (6) 所示:

$$sim(p_u, p_v^*) = \frac{1}{s} \times \sum_{l=1}^{s} SN_l, \text{ s.t. } \begin{cases} SN_l = 1, \text{ if } p_u^{(l)} = p_v^{*(l)} \\ SN_l = 0, \text{ if } p_u^{(l)} \neq p_v^{*(l)} \end{cases} \tag{5}$$

$$\boldsymbol{R} = \boldsymbol{P}_{\text{cov}} \times \boldsymbol{P}_{\text{tar}} = \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_r \end{bmatrix} \times [p_1^*, p_2^*, \ldots, p_c^*] = \begin{bmatrix} sim_{11} & sim_{12} & sim_{13} & \cdots & sim_{1c} \\ sim_{21} & sim_{22} & sim_{23} & \cdots & sim_{2c} \\ \cdots & \cdots & \cdots & \ddots & \cdots \\ sim_{r1} & sim_{r2} & sim_{r3} & \cdots & sim_{rc} \end{bmatrix} \tag{6}$$

其中, SN_l 记录用例覆盖路径 p_u 与目标路径 p_v^* ($1 \leqslant v \leqslant c$) 第 l 个路径节点状态是否相同, 相同则将其记为 1, 否则为 0; $sim(p_u, p_v^*)$ 则表示路径 p_u 与 p_v^* 的相似度, 简写为 sim_{uv}.

例 4: 假定多目标路径为 p_1^* =(1, 1, 1, 0, 1), p_2^* =(1, 1, 0, 0, 1), 则多目标路径矩阵 $\boldsymbol{P}_{\text{cov}} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{bmatrix}$. 根据公式 (5) 和 (6) 可知, 测试用例 t_1 的覆盖路径 p_1 与目标路径 p_1^* 的相似度 $sim(p_1, p_1^*) = 0.4$, $sim(p_1, p_2^*) = 0.8$; 同理, $sim(p_2, p_1^*) = 0.6$, $sim(p_2, p_2^*) = 0.6$, $sim(p_3, p_1^*) = 0.8$, $sim(p_3, p_2^*) = 0.8$, $sim(p_4, p_1^*) = 0.6$, $sim(p_4, p_2^*) = 0.6$, $sim(p_5, p_1^*) = 0.6$, $sim(p_5, p_2^*) = 0.6$. 因此, 可得如下多路径相似矩阵 $\boldsymbol{R}$:

$$\boldsymbol{R} = \begin{bmatrix} 0.4 & 0.8 \\ 0.6 & 0.6 \\ 0.8 & 0.8 \\ 0.6 & 0.6 \\ 0.6 & 0.6 \end{bmatrix}.$$

多路径相似矩阵 $\boldsymbol{R}$ 的行表示某一用例的覆盖路径与目标路径的相似度, 若相似度较大, 且与多个目标路径均相似, 则表明该测试用例具备覆盖更多目标路径的潜能. 因此, 我们根据测试用例与目标路径的相似程度来衡量测试用例的覆盖潜能, 约简并保留高潜能的用例. 测试用例 t_i 的潜能 PT_{t_i} 为达到相似度阈值路径数与总目标路径数的比值, 如公式 (7) 所示:

$$PT_{t_i} = \frac{Num_{t_i}}{Total_{p^*}} \times 100\% \tag{7}$$

其中, Num_{t_i} 表示测试用例 t_i 的覆盖路径与目标路径的相似度大于阈值的个数, $Total_{p^*}$ 为总目标路径数.

例 5: 假定相似度阈值为 0.6, 因目标路径为 p_1^* 和 p_2^*, 则 $Total_{p^*} = 2$, 利用公式 (7) 计算 t_1 的潜能为 $PT_{t_1} = 1/2 = 0.5$, t_2 的潜能 $PT_{t_2} = 2/2 = 1$, t_3 的潜能 $PT_{t_3} = 2/2 = 1$, t_4 的潜能 $PT_{t_4} = 2/2 = 1$, t_5 的潜能 $PT_{t_5} = 2/2 = 1$.

此外, 随机生成的用例往往难以覆盖难覆盖的目标路径. 因此, 覆盖难覆盖路径的相似用例有助于进化生成难覆盖目标路径的测试用例. 多路径相似矩阵 $\boldsymbol{R}$ 的列可反映目标路径被用例覆盖的程度. 若达到相似度阈值的测试用例较多, 则表明该路径越容易被覆盖, 反之亦然. 因此, 通过衡量路径的难易覆盖程度, 可在约简测试用例时尽可能地保留更多的与覆盖难覆盖目标路径相似的用例. 目标路径 p_v^* 被覆盖的难易程度 $DP_{p_v^*}$ 则为达到相似度阈值的测试用例数与总测试用例数的比值, 如公式 (8) 所示:

$$DP_{p_v^*} = \frac{Num_{p_v^*}}{Total_t} \tag{8}$$

其中, $Num_{p_v^*}$ 为与第 v 个目标路径相似的覆盖路径对应的用例数, $Total_t$ 为用例总和. 若 $DP_{p_v^*}$ 值越大, 则该目标路径越易被覆盖, 反之则越难被覆盖.

例 6: 根据公式 (8) 计算可得, 目标路径 p_1^* 与 p_2^* 的覆盖难易程度分别为 $DP_{p_1^*} = 4/5 = 0.8$, $DP_{p_2^*} = 5/5 = 1$. 故生成覆盖 p_1^* 的测试用例难度高于 p_2^*, 选择用例时则更多考虑可覆盖难覆盖路径 p_1^* 的测试用例.

通常, 同一簇中的用例相似度较高, 它们的覆盖路径存在一定的相似性, 且一般覆盖难覆盖路径的概率较低. 为筛选出具有多样性的较优用例组, 使它们尽早地参与遗传演化以覆盖目标路径, 需在每个簇中约简测试用例, 保留潜能较大的测试用例, 同时还需侧重考虑与难覆盖路径相似的路径对应的用例.

基于此, 依据用例潜能对不同簇的测试用例排序, 并对目标路径的难易覆盖程度排序. 用例簇中的测试用例数占总测试用例数的比例越高, 表明此类用例更易生成. 因此, 依据用例的占比选择少部分此类用例, 而在约简

集中保留更多难生成的用例. 不同簇中用例数量的选取方式如公式 (9) 所示, 测试用例约简集表达形式如公式 (10) 所示:

$$TCNum_j = \left[\left(1 - \frac{TC_j}{KMNum}\right)/k\right] \times Num_{\text{tar}},\ 1 \leqslant j \leqslant k \tag{9}$$

$$\begin{cases} TC_{\text{Red}} = \{tcr_1, tcr_2, \ldots, tcr_k, tcr_{DP}\} \\ \text{s.t. } tcr_j = (t_1, t_2, \ldots, t_{TCNum_j}),\ 1 \leqslant j \leqslant k,\ tcr_{DP} = (t_1, t_2, \ldots, t_d),\ d = Num_{\text{tar}} - \sum_{j=1}^{k} Num_j \end{cases} \tag{10}$$

在公式 (9) 中, TC_j 为第 j 个簇中的用例数, $KMNum$ 为经 K-means 聚类后保留下来测试用例的数量, Num_{tar} 为预设的约简集中用例总数, $TCNum_j$ 为第 j 个簇中保留的约简用例的数量. 公式 (10) 中, TC_{Red} 为测试用例约简集, 其包含 k 个用例簇 tcr_1, tcr_2,…, tcr_k (选取每个簇中的前 $TCNum$ 个用例) 与依据路径难易覆盖程度选取的用例组 tcr_{DP} (选取与覆盖难覆盖路径相似的前 d 个用例).

例 7: 基于前述例子可知 $KMNum$=5, 为便于描述, 假设约简集的用例总数 Num_{tar} 为 4. 根据公式 (9) 可知, 需从第 1 个簇中选取 1 个用例, 从第 2 个簇中选取 2 个用例, 第 3 个簇中的用例为噪声数据, 已被删除. 因此, 当前 k 为 2, 根据公式 (10), 仅从前两个簇中选择测试用例作为测试用例约简集. 根据用例潜能择优原则, 选择第 1 个簇中 t_2 (随机从 t_2、t_3 中选择), 第 2 个簇中的 t_4、t_5 为约简集中的测试用例. 另外, 根据目标路径的难易覆盖程度, 从这些用例中选出一个与难覆盖路径 p_1^* 更相似的测试用例 t_3 加入约简集中. 故最终测试用例约简集 TC_{Red}={t_2, t_4, t_5, t_3}.

3.2 基于 SVR 模型的适应度值预测

适应度函数作为个体择优的评判方式, 在启发式测试用例生成中起到至关重要的作用. 因用例的覆盖路径与目标路径越相似, 且与之相似的目标路径越多, 该用例则可能具备更高的目标路径覆盖概率 (即, 用例潜能越大), 故我们将公式 (7) 作为个体适应度评估函数.

在个体优劣评判过程中, 尤其是针对规模较大或目标路径较多的工业程序, 直接利用启发式算法生成用例需更多的进化次数, 进而需多次地执行程序以评估用例是否覆盖目标路径, 这必然会增加一定测试成本. 而 SVR 是一种基于惩罚学习的轻量级回归模型, 通过学习 n 个输入用例 x 及其对应的适应度 y 间的回归关系 $f(x)$, 构建精确的 SVR 适应度预测模型, 以利用该模型代替执行程序获取多路径个体适应度, 有助于缩减多次执行程序的测试时间. 此外, 充分且多样的测试数据可提高 SVR 模型精度, 故我们采用比约简集数据量更多的初始随机数据, 并剔除通过公式 (3) 评估的噪声数据, 进行 SVR 训练. 可用于多路径覆盖的 SVR 个体适应度预测模型如公式 (11) 所示:

$$y = f(x) = \sum_{i=1}^{n} wx_i + b \tag{11}$$

其中, w 表示个体适应度预测模型的权重, 引入偏置 b 提高个体适应度预测模型的泛化能力.

例 8: 选取初始测试数据 T={t_1=(1, 2, 3), t_2= (1, 2, 5), t_3=(4, 3, 5), t_4=(7, 9, 7), t_5=(7, 9, 9)} (来源于例 1), 每个用例的 3 个输入分量为模型的输入特征. 通过式 (7) 计算它们的适应度值, 分别为 y_1=0.5, y_2=1, y_3=1, y_4=1, y_5=1. 故由测试用例及其对应适应度值构成的 SVR 适应度预测模型的训练数据形如 (x_i, y_i)=((1, 2, 3), 0.5). 设模型根据公式 (11) 训练学习到的初始参数值为 w=(0.02, 0.05, 0.03), b=0.4. 假定在演化过程中生成新个体 t_7=(3, 4, 1), 利用初始模型可得其适应度预测值为 y_7=0.69. 若不采用 SVR 模型预测, 则需再次执行算法 1 中的程序获取 t_7 的适应度值, 得到 0.5. 可见, 若不采用 SVR 模型则必然会花费一定程序执行时间才可得到当前用例的适应度值. 若面向规模较大或目标路径数较多的程序, 测试用例所消耗的多次执行时间往往会远高于 SVR 模型构建与预测时间. 这里, 当前 t_7 的预测值与真实值间的误差为 0.19, 为使得预测值更加接近真实值, 便于 SVR 模型能替代程序执行获取个体适应度, 用于演化生成目标用例, 则需对模型进一步优化.

值得注意的是, 在训练数据集中仍难免存在噪声测试用例. 为使部分噪声用例对模型的影响降低到最小, 引入松弛因子 ξ^+ 与 ξ^- (表示超出和未超出 ε 惩罚区间的情况), 以及参数 C 优化个体适应度预测模型, 使得对适应度的预测更准确, 优化后的目标函数如公式 (12) 所示:

$$\begin{cases} MinR(w,\xi^+,\xi^-)=\dfrac{1}{2}\|w\|^2+C\sum_{i=1}^{n}(\xi^++\xi^-) \\ \text{s.t. } f(x)-y^i\leqslant\varepsilon+\xi_i^-,\ y^i-f(x)\geqslant\varepsilon+\xi_i^+ \text{且 } \xi^+\geqslant 0,\ \xi^-\geqslant 0 \end{cases} \tag{12}$$

例 9: 设 ε 惩罚区间为 (−0.2, 0.2), 该区间允许存在一定的小误差, 有助于提高模型的泛化能力. 以 t_3=(4, 3, 5) 为例, 其预测值 y_3' =0.78, 与真实值 (y_3=1) 间的误差为 0.22, 超过了惩罚范围, 超越值为 0.02. 为优化该预测偏差, 引入常用的松弛因子 ξ^+ =0.1 对模型进行调整, 使其更接近真实值. 松弛因子可有效缓解噪声用例对模型训练的干扰, 提升个体适应度预测的准确性. 设通过公式 (12) 优化后的 SVR 模型对 t_3 的适应度预测值 y_3'' =0.85. 相比于 y_3', 调整后的适应度预测值 y_3'' 更接近真实值 y_3.

直接优化目标函数复杂度高, 会制约适应度预测模型的参数优化. 因此, 引入 Lagrange 因子 α–i (记为 α_i^-) 和 α+i (记为 α_i^+), 将优化目标转化为二次规划问题, 以更高效地学习 SVR 适应度预测模型参数. 利用 Lagrange 函数将约束条件融入目标适应度预测函数, 得到 Lagrange 因子与 w 的关系. 优化目标函数转化为如公式 (13) 所示:

$$y=f(x,\alpha_i^+,\alpha_i^-)=\sum_{i=1}^{n}(\alpha_i^+-\alpha_i^-)(x_i\cdot x)+b \tag{13}$$

例 10: 为进一步提高模型的拟合精度并加速参数求解过程, 通过引入 Lagrange 优化后的模型参数值为 w'= (0.011, −0.039, 0.1), b'=0.47. 此时 t_3 的适应度预测值为 y_3''' =0.897. 相比于 y_3'' 和 y_3', 其与真实值间的误差最小, 从而进一步提升适应度预测结果的准确性.

此外, 在测试用例的输入特征与其适应度间往往存在非线性关系, 导致样本在特征空间中线性不可分, 影响 SVR 模型预测的准确性. 为此, 利用核方法将测试用例转换到高维特征域中, 进一步优化模型. 然而, 高维映射会增加计算开销, 故可引入径向基核函数 K 简化在同一特征空间中内积<x_i, x> 的计算, 有效提高适应度预测模型精度以及降低模型计算的复杂度, 最终目标函数如公式 (14) 所示:

$$y=f(x,\alpha_i^+,\alpha_i^-)=\sum_{i=1}^{n}(\alpha_i^+-\alpha_i^-)K(x_i,x)+b \tag{14}$$

例 11: 假定 t_3 是一个线性不可分的用例样本, 直接计算 t_3 与其他用例间的内积无法体现非线性关系, 故需通过径向基核函数 K 计算该用例与其他用例间的相似性, 以替代高维内积运算, 解决线性不可分的问题, 降低计算复杂度. 即, $K(t_3, t_1)$=exp(−7), $K(t_3, t_2)$=exp(−5), $K(t_3, t_4)$=exp(−24.5), $K(t_3, t_5)$=exp(−30.5), 这些核函数值反映了用例 t_3 与其他用例映射到高维空间后的相似性强弱, 即它们在高维空间的差异性. 将这些结果代入公式 (14), 可提升 SVR 模型对线性不可分用例的拟合能力, 进而优化模型.

上述步骤通过模型训练阶段的参数优化提高了 SVR 模型的预测能力, 但由于用例的多样性及非线性特征分布, 适应度预测值与真实值间仍可能存在偏差. 为衡量 SVR 模型预测每一次个体适应度的准确性, 利用广泛使用的平方损失函数计算预测值与真实值间的偏差, 达到更佳预测效果, 同时增强其泛化能力, 如公式 (15) 所示:

$$L(Y,f(X))=(Y-f(X))^2 \tag{15}$$

其中, $f(X)$ 与 Y 分别为模型获取的个体适应度预测值与程序执行获得的精确值.

例 12: 根据例 10 和例 11 学习得到的模型参数, 可得这 5 个用例的适应度预测值为 $f(t_1)$=0.703, $f(t_2)$=0.903, $f(t_3)$=0.897, $f(t_4)$=0.896, $f(t_5)$=1.096, 它们的损失分别为 $L(y_1, f(t_1))$=0.0412, $L(y_2, f(t_2))$=0.0094, $L(y_3, f(t_3))$=0.0106, $L(y_4, f(t_4))$=0.0108, $L(y_5, f(t_5))$=0.0092, 总损失为 L_{total}=0.0813. 利用梯度下降以寻找最小化总损失, 优化后的参数为 w''=(0.0013, −0.0039, 0.1), b''=0.47. 在该参数下的这 5 个测试用例总损失 L_{total}=0.0803. 此时, 测试用例 t_7 的预测值 y_7' =0.5583, 与真实值 0.5 间的误差为 0.058 3. 相比于初始 SVR 模型的预测结果 y_7=0.69 (见例 8), y_7' 更接近真实值, 误差较小. 可见, 通过上述一系列 SVR 模型优化策略能有效提高模型精度, 使误差控制在可接受范围内, 进而在获

取测试用例适应度值时利用 SVR 预测模型替代程序执行, 减少用例多次执行程序带来的显著时间消耗, 提高多目标路径的测试用例生成效率.

我们针对多目标路径覆盖的测试用例生成, 前期利用 K-means 生成的测试用例约简集作为遗传初始种群, 提高初始种群质量并使得用例具备多样性, 后期通过 SVR 模型在种群进化中预测个体适应度, 减少因种群迭代产生个体需多次执行插桩程序所带来的时间消耗.

4 结合 K-means 与 SVR 的测试用例约简与生成

融入机器学习的启发式用例生成方法在测试生成方面效果显著. 本文先利用 K-means 对不同簇中的测试用例进行约简, 提高种群质量; 再利用 SVR 模型预测种群中个体适应度, 减少多路径个体适应度求解时间. K-means 与 SVR 模型相结合的测试用例约简与生成框架如图 1 所示.

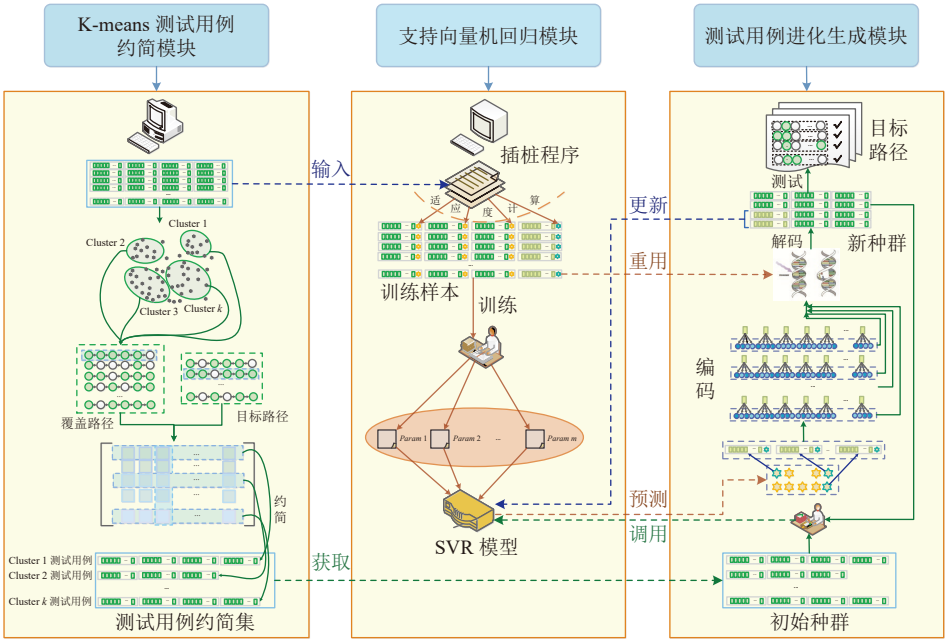

图 1 结合 K-means 与 SVR 模型的测试用例约简与生成框架

主要包括 3 个模块, 分别为 K-means 测试用例集约简模块、支持向量机回归模块和测试用例进化生成模块. ① K-means 测试用例集约简模块主要利用 K-means 聚类策略及测试用例对目标路径的覆盖能力, 以簇为单位对聚类后的测试用例进行约简, 选取精英用例, 减少冗余, 提高初始种群质量; ② 支持向量机回归模块主要是将初始测试用例输入插桩程序计算适应度值, 利用测试用例及对应的适应度值训练可模拟覆盖多路径的个体适应度 SVR 模型, 减少插桩时间; ③ 测试用例进化生成模块主要利用约简集和 SVR 模型启发生成目标测试用例. 在生成用例时, 从用例约简模块中获取测试用例约简集作为初始遗传种群, 提高种群个体质量; 对生成的个体调用构建的 SVR 模型预测其适应度, 提高用例生成效率. 在种群进化中引入训练样本中部分优秀个体基因进行重用, 并利用进化种群中生成的优秀个体更新 SVR 模型, 进一步提高测试效率.

4.1 利用 K-means 聚类约简测试用例

一方面, 距离相近的个体划分到一个簇中, 可覆盖相同或相似目标路径; 另一方面, 初始种群中的个体质量在一定程度上影响遗传进化的方向与效率. 本节采用 K-means 聚类获取约简集, 并将其作为遗传初始种群, 这可避免随机种群普遍质量较低的问题, 同时减少数据的冗余, 并让初始种群具备多样性. 具体过程如算法 2 所示.

算法 2. 利用 K-means 生成测试用例约简集.

输入: 随机测试用例集 *TCRand*, 聚类数 *k*, 目标路径集 *PTLst*;
输出: 测试用例约简集 *TCRedLst.*

Begin
1. *clusterCen*, *clusterTC* ←*cluKM*(*TCRand*, *k*);
// 利用公式 (1), (2) 获取 K-means 聚类后的簇中心及用例
2. **For** *cen* **in** *range*(*clusterCen*):
3. **For** *tc* **in** *range*(*clusterTC*):
4. *dist* ← *calcDist*(*tc*, *cen*); // 利用公式 (3) 计算每个簇中用例与簇中心的距离
5. **If** *dist* > 0:
6. *clusLst* ← *Save*(*tc*); // 保存距离大于 0 的测试用例
7. **Else**:
8. *Delete*(*tc*); // 删除距离小于 0 的测试用例
9. **End If**
10. **End For**
11. **End For**
12. *TCRedLst* ← ∅; // 初始化测试用例约简集
13. *path* ← *Instru*(*clusLst*); // 执行插桩程序获取覆盖路径
14. *pathCovMat* ← *genMatrix*(*path*); // 生成如公式 (4) 的路径覆盖矩阵
15. *pathSimMat* ← *calcSim*(*pathCovMat*, *PTLst*); // 利用公式 (5), (6) 获取路径相似度矩阵
16. *TCPoten* ← *calcPoten*(*pathSimMat*); // 利用公式 (7) 计算用例潜能
17. *pathDegree* ← *calcDegree*(*pathSimMat*); // 利用公式 (8) 计算路径难易覆盖度
18. *TCRedLst* ← *Reduce*(*TCPoten*, *pathDegree*); // 利用公式 (9), (10) 筛选并生成用例约简集
19. **Output** *TCRedLst*
End

首先, 在第 1 行, 利用 K-means 将随机生成的测试数据聚为 *k* 个簇, 获取聚类中心以及簇中测试用例. 其次, 在第 2–11 行, 利用公式 (3) 计算每个簇中用例与簇中心的距离, 保留距离大于 0 的用例, 将小于 0 的剔除 (噪声值). 通常情况下, 若聚类效果较优, 噪声值往往较少. 然后, 在第 12–17 行, 计算测试用例潜能以及路径难易覆盖程度. 最后, 在第 18 行, 利用公式 (9) 和 (10) 对不同簇中的测试用例进行约简, 构成用例约简集.

4.2 利用 SVR 模型预测适应度值

随着程序规模及其复杂度的增大, 在遗传进化中个体适应度计算时间显著增加, 尤其是面向多目标路径的测试用例生成. 为减少因多次执行插桩程序获取适应度带来的高昂时间成本, 我们构建适用于多路径覆盖的个体适应度预测模型. 具体过程如算法 3 所示.

算法 3. 利用 SVR 模型预测个体适应度值.

输入: 测试样本 *TCSamp*, 优秀个体集 *EPLst*, 模型更新阈值 *UT*;
输出: 个体适应度预测值 *fitValue.*

Begin
1. *TCTrain*, *TCTest* ← *Divide*(*TCSamp*); // 划分训练集和预测集

2. *SVRModel* ←*constructFitPredSVR*(); // 构建 SVR 适应度预测模型
3. *fitValue* ← *SVRModel.predFit*(*TCTest*); // 利用 SVR 模型预测个体适应度值
4. **Output** *fitValue*
// 子函数 *constructFitPredSVR*(), 构建 SVR 适应度预测模型
5. **Def** *constructFitPredSVR*():
6. *SVRModel* ← *trainModel*(*TCTrain*); // 利用公式 (11)–(15) 训练 SVR 回归模型
7. **If** *len*(*EPLst*) ⩾ *UT*: // 优秀个体数达到阈值
8. *updateModel*(*SVRModel*);
9. **End If**
10. **Return** *SVRModel*
11. **End Def**
End

在第 1 行, 把初始测试用例输入插桩程序, 计算每个用例的精确适应度, 将用例及其对应的适应度作为训练样本, 并将它们划分为训练集与预测集, 以构建 SVR 模型. 在第 2 行, 调用函数 *constructFitPredSVR*(), 根据公式 (11)–(13) 求解公式 (14) 的目标函数, 并利用公式 (15) 的损失函数不断优化 SVR 模型, 直到损失值最小, 此时 SVR 模型构建完成. 另外, 由于初始种群质量对 SVR 模型精度具有一定影响, 故需利用演化生成中产生的一定数量较优个体更新 SVR 模型, 进一步提高模型精度. 在第 3 行, 将测试用例输入训练好的 SVR 模型, 并在第 4 行输出对应的适应度预测值.

4.3 进化生成测试用例

这里借助遗传算法生成目标路径的测试用例, 并将第 4.1 节的测试用例约简集作为遗传初始种群, 将第 4.2 节的 SVR 模型用于预测个体适应度, 以生成覆盖多目标路径的测试用例. 具体过程如算法 4 所示.

算法 4. 结合 K-means 方法与 SVR 模型生成测试用例.

输入: 测试用例约简集 *TCRedLst*, 外部优秀个体集 *EEPLst*, 插桩阈值 *IT*, 目标路径集 *PTLst*, 目标适应度 *fitTarg*, 种群大小 *popSize*, 最大进化代数 *ET*, 交叉概率 *CP*, 变异概率 *MP*;
输出: 目标测试用例集 *TCTargLst*.

Begin
1. *TCTargLst* ← [ ]; // 初始化目标测试用例
2. *initPop* ← *TCRedLst*; // 将算法 2 的约简集作为遗传初始种群
3. *SVRModel* ←*constructFitPredSVR*(); // 利用算法 3 的 *constructFitPredSVR*() 函数, 构建 SVR 模型
4. *trueValue*, *TCTargLst* ← *getTCTarg*(*initPop*, *predValue*); // 获取个体精确适应度与目标用例集
5. *evotime* = 0;
6. **For** *evotime* **in** *range*(*ET*):
7. **If** *PTLst* <>∅:
8. *popLst*1 ← *Roulette*(*popSize*, *trueValue*); // 轮盘赌法选择个体
9. *popLst*2 ← *Crossover*(*popLst*1, *EEPLst*, *CP*); // 引入优秀个体交叉并保留
10. *newPopLst* ← *Mutate*(*popLst*2, *MP*); // 变异操作
11. *newTC* ← *convertToDecimal*(*newPopLst*); // 转换成十进制
12. *predValue* ← *SVRModel.predFit*(*newTC*); // 根据算法 3 预测个体适应度
13. *trueValue*, *TCTargLst* ← *getTCTarg*(*newTC*, *predValue*);

```
        // 获取个体精确适应度与目标用例集
14.   Else:
15.       Break;
16.   End If
17.   If len(TCTargLst) ⩾ UT: // 优秀测试用例数达到阈值
18.       SVRModel ← constructFitPredSVR();
        // 利用算法 3 的 constructFitPredSVR() 函数, 更新 SVR 模型
19.   End If
20.   evotime += 1
21. End For
22. Output TCTargLst;
  // 子函数 getTCTarg(), 删除目标路径集 PTLst 中已覆盖路径, 保存目标用例 TCTargLst
23. Def getTCTarg(TC, tempValue): // 获取目标测试用例
24.   If tempValue ⩾ IT: // 达到插桩验证的阈值
25.       trueValue, path ← Instru(TC); // 将当前用例输入插桩程序获取精确值及覆盖路径
26.       If path ∈ PTLst:
27.           PTLst.drop(path);
28.           TCTargLst.append(TC); // 保存可覆盖目标路径的测试用例
29.       End If
30.   End If
31.   Return trueValue, TCTargLst
32. End Def
End
```

在第 1 行, 初始化目标测试用例; 在第 2 行, 从算法 2 中获取约简集, 将其作为遗传初始种群. 在第 3 行, 利用算法 3 的 *constructFitPredSVR*() 函数训练 SVR 模型, 并在第 4 行, 调用 *getTCTarg*() 函数, 判定目标路径是否全覆盖, 并保存目标用例. 在第 5–21 行, 若目标路径集不为空, 表明仍有目标路径未被覆盖, 则需进行遗传进化, 在交叉过程中引入适应度较高的个体基因进行用例重用; 为避免陷入局部最优, 某一个体被引入次数超过给定次数则将其删除. 生成新一代种群后, 将其转换为十进制, 调用 SVR 模型以预测适应度. 再次调用 *getTCTarg*() 函数, 若当前用例覆盖了目标路径, 则将该路径从目标路径集中剔除; 优秀用例集中的用例数达到一定阈值, 则调用算法 3 的 *constructFitPredSVR*() 函数更新 SVR 模型, 使得在后续用例进化生成中个体适应度预测更加精确; 当目标路径集中的所有路径均被覆盖或达到最大进化代数, 进化操作终止. 在第 22 行, 输出目标测试用例集. 为便于理解, 下面接着前述若干示例介绍结合 K-means 与 SVR 的多路径测试用例生成过程.

例 13: 由例 7 可知, 初始种群为 $initPop=\{t_2, t_4, t_5, t_3\}$, 而由例 12 可得最优多路径个体适应度预测 SVR 模型. 先将这 4 个测试用例执行三角形判定程序, 获取它们的真实适应度值以及覆盖路径. 由例 4 可知, 这些用例并未覆盖目标路径, 故需根据它们的适应度值, 通过轮盘赌法选择优秀用例, 进行遗传进化. 适应度决定了它们被选中的概率. 假定选中个体$\{t_3, t_4\}$作为交叉变异的父代. 接着, 将这两个用例转换为二进制编码, 并进行交叉与变异, 形成新的两个子代. 之后, 将子代转换为十进制表示为 t'_3 =(4, 3, 6), t'_4 =(7, 7, 7), 获得第 2 代种群为$\{t_3, t_4, t'_3, t'_4\}$. 将当前种群中的个体输入 SVR 模型, 预测适应度值, 它们的适应度分别为: 0.897、0.896、1.005、0.998. 相比于 t_3 和 t_4, 用例 t'_3 和 t'_4 的适应度预测值更接近于 1, 故选择这两个用例执行程序, 验证其真实值及覆盖路径, 而 t_3 和 t_4 则不执行程序. 由此可知, t'_4 覆盖了例 4 中的一条目标路径 [1, 1, 1, 0, 0], 并将该路径移除. 由于目标路径集中仍存在一条未覆盖目标路径 [1, 1, 0, 0, 1]. 故需继续执行上述遗传进化操作, 直至目标路径被全覆盖或达到迭代阈值, 则完成用例生成.

由于例 13 的路径数较少, SVR 模型无需进行更新. 但在用例数量及目标路径数较多时, 则需根据路径覆盖情况动态更新多路径个体适应度预测模型, 以提高 SVR 模型预测效率. 下面通过实验验证结合 K-means 与 SVR 模型方法的有效性.

5 实验设计与分析

为检验基于 K-means 的测试用例约简法以及 SVR 模型代替插桩法预测个体适应度对用例生成的有效性, 本节通过大量仿真实验进行对比.

5.1 问题研究

本文提出的测试用例生成方法能否提高测试效率, 一方面取决于采用 K-means 聚类方式筛选的约简集的质量, 另一方面取决于 SVR 模型预测多路径个体适应度的准确性. 实验中围绕以下几个问题探讨本文方法在测试用例生成方面的优势.

问题 1: 测试用例约简集能否改善初始种群质量, 推进遗传进化过程, 提高多目标路径测试用例生成效率?

针对此问题, 分别从约简集中的测试用例数、聚类簇数, 以及利用约简集生成测试用例效率这几方面进行验证. 通过初始种群的不同选取方式设计对比实验, 选取方式包括随机生成测试用例与 K-means 聚类约简测试用例这两种. 在对比方法中, 它们仅初始种群来源不同, 其他操作均相同. 实验设计与分析见第 5.4.1 节.

问题 2: (a) SVR 模型预测个体适应度的准确性如何? (b) 利用 SVR 模型能否提高面向多路径的测试效率?

针对问题 (a), 为验证 SVR 模型在不同复杂度程序上的准确性, 随机抽取 1 个功能较单一程序与功能较复杂程序, 利用一定数量的样本训练 SVR 回归模型, 通过已训练模型获取预测样本的适应度, 将预测样本的预测值与其精确值作对比. 针对问题 (b), 通过个体适应度的不同获取方式设计对比实验, 包括适应度函数计算方式 (即执行插桩程序) 与 SVR 模型预测方式. 在对比方法中, 它们仅获取个体适应度的方式不一样, 其他操作均相同. 实验设计与分析见第 5.4.2 节.

问题 3: 结合 K-means 与 SVR 模型的测试用例约简与生成方法 (即本文方法) 是否可提高测试效率? 其效率可提高到何种程度?

针对此问题, 在前两个问题的基础上设计消融实验, 将基于 K-means 的测试用例约简集与 SVR 模型预测的适应度均用于遗传进化中, 与仅采用约简集等方法做对比, 以验证本文将两者相结合的方法在测试方面的效率以及提高程度. 实验设计与分析见第 5.4.3 节.

此外, 为进一步验证本文方法的优势, 在第 5.4.4 节还与其他典型的测试用例生成方法对比, 比较这些对比方法在不同程序上的测试效率.

5.2 评价标准

为检验基于 K-means 的约简集以及通过 SVR 模型计算种群个体适应度的方法对测试用例生成效率的影响, 主要使用如下几个指标进行评价.

1) 覆盖率: 执行若干次实验, 在最大进化代数内目标路径均被覆盖的次数总和占总执行次数的比例.

2) 进化代数总量: 在若干次实验中, 覆盖任一条目标路径所需进化代数的总和.

3) 平均进化代数: 若干次实验进化代数总和的平均值. 具体地, 在执行若干次实验中, 若在最大进化代数内, 待测程序的目标路径均被覆盖, 则记录此时的进化代数, 若未被全覆盖, 则记录最大进化代数.

4) 执行时间: 若干次实验中测试用例生成的时间消耗总和.

本文实验均执行 100 次对结果进行综合统计, 以更合理地说明所提方法的有效性. 为更全面地呈现各方法间的差异性, 我们在覆盖率、进化代数、执行时间这几个方面的基础上设置额外的对比指标 (如覆盖率差值), 在第 5.4.3 与 5.4.4 节实验的对应部分给出了相关阐述.

5.3 程序信息与实验参数

本文选取 2 个基础程序和 13 个工业程序进行实验, 它们已被广泛运用于测试分析中, 具体信息如表 1 所示.

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	目标路径数	程序来源
基础程序	PG1	冒泡排序	8	1	2	https://blog.csdn.net/s1156605343/article/details/106177863
	PG2	三角形判定	15	1	5	https://www.jb51.net/article/184534.htm
工业程序	PG3	LuhnCheck	24	1	2	https://blog.csdn.net/weixin_33716557/article/details/91539545
	PG4	计算器	196	5	21	https://www.cnblogs.com/quemengqio/p/7799468.html
	PG5	Snake	163	11	35	https://gitee.com/explore
	PG6	Gobang	470	23	57	
	PG7	Account	1 670	36	61	
	PG8	Nav Acc Data PP	3 990	155	123	
	PG9	Sed	9 341	77	12	https://sir.csc.ncsu.edu/php
	PG10	Flex	14 405	162	20	
	PG11	DepSolver	8 988	227	20	https://github.com/chef-boneyard/depsolver
	PG12	ClustalW	23 265	468	30	https://github.com/coldfunction/CUDA-clustalW
	PG13	AllTrue32	7	1	33	文献[14]
	PG14	Transmit	30	1	2	https://github.com/Cloudslab/iFogSim
	PG15	Send	47	1	9	

在表 1 中, PG1 和 PG2 为基础程序, 它们的代码行数、函数个数, 以及目标路径均较少. PG3–PG15 为工业程序, 它们的代码行数在 7–23 265 范围内, 函数个数在 1–468 范围内, 目标路径数在 2–123 范围内. 由于被测程序的语言各异, 我们选择程序 PG1 和 PG10 用于消融实验, 以检测本文方法各模块的有效性; 程序 PG11–PG15 作为与其他测试生成方法对比的实验对象, 以验证本文方法的优势.

本实验在 Windows 10 平台下完成, 采用 Python 语言与 jdk1.8, 并使用 PyCharm 开发环境. 为使多样性种群以及 SVR 适应度预测模型更好地适用于遗传进化策略, 同时使基于用例潜能的个体选择策略以更大概率生成覆盖多目标路径的测试用例, 以及尽可能地覆盖多目标路径, 搜索更广泛的空间, 设置多点交叉概率为 0.9, 变异概率为 0.1, 进化代数阈值为 1 000 等. 此外, 基准方法的参数设置基本相同, 除了初始种群规模因测试方法、程序等不同而有所区别, 其他参数均相同.

5.4 实验对比与分析

这里先介绍本文实验中涉及的各类不同方法的特点, 具体信息如表 2 所示.

表 2 不同测试用例生成方法介绍

方法类别	方法名	方法简述
基准方法	插桩法	随机生成初始种群, 通过执行插桩程序计算个体适应度, 并以一定概率对个体实施交叉变异等操作, 生成测试用例
	约简法	利用第3.1节提出的测试用例约简集作为遗传初始种群, 其他过程与插桩法一致
	支持向量机回归法(SVR法)	利用第3.2节构建的SVR模型预测遗传进化中种群个体的适应度, 并对较优个体进行插桩验证, 其他过程与插桩法一致
对比方法	关键点概率[15]	将可覆盖关键点的测试用例优先参与遗传进化, 启发生成目标测试用例
	DSGEO[13]	利用代理模型评估个体适应度并指导生成测试用例
	HSADE[14]	结合全局与局部代理模型在目标子空间中进行精确的适应度估计, 指导生成测试用例
	DNTSA[3]	基于路径结构矩阵和mainfold-inspired搜索自动生成测试用例
	MfO-PC[1]	基于MfO-PC框架与种群的搜索算法生成测试用例
本文方法	KM-SVR法	将K-means用例约简集作为初始种群, 利用SVR模型预测个体适应度, 并在遗传进化中引入优秀个体基因, 其他过程均与插桩法一致

5.4.1 用例约简实验

针对问题 1, 将利用 K-means 聚类方法约简后的用例作为遗传初始个体实施进化生成过程, 其中用例数量与

质量均对测试效率具有一定影响. 下面分别从约简集中的用例数量、聚类簇数, 以及约简集对测试效率影响等方面, 验证约简集在测试用例生成中的有效性.

1) 约简集中用例数量对测试生成效率的影响

覆盖率是评价测试方法有效性的核心指标, 可直接体现方法的有效性. 为验证约简集中用例数量的作用, 选取不同数量的用例以观察它们在不同程序上对覆盖率的影响, 如图 2 所示. 其中横坐标为测试用例数量, 纵坐标为目标路径覆盖率.

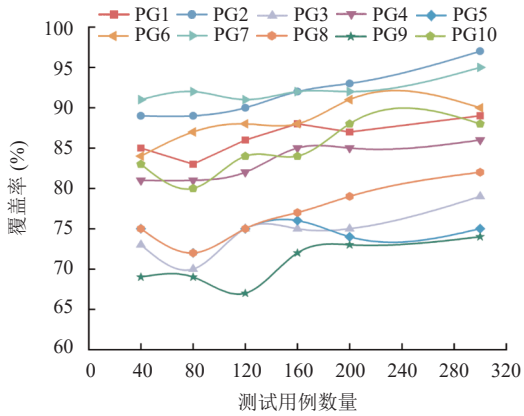

图 2　在不同程序上测试用例数量对覆盖率的影响

由图 2 可知, 随着用例数增加, 每个程序的覆盖率大致呈上升趋势. 其中, 大部分程序在用例数为 40、80、120、160、200 时覆盖率提升相对缓慢; 在数量为 300 时, PG2、PG3、PG7、PG8 提升较为明显, 程序 PG1、PG4、PG5、PG9、PG10 提升较少. 由此可见, 约简集中的用例数量在不同程序上对覆盖率影响不同.

另外, 初始测试用例数量越多, 在个体进化生成中所需时间也越多, 因此, 选择初始种群个体数量时不仅需考虑覆盖率, 还需兼顾执行时间. 下面采用与图 2 同样规模的测试用例, 观察它们在不同程序上对执行时间的影响. 说明: 由于程序规模、复杂度等不同, 它们在执行时间上存在较大差异, 采用表格形式表达更清晰, 实验结果如表 3 所示.

表 3　在不同程序上测试用例数量对执行时间的影响 (s)

待测程序	40	80	120	160	200	300
PG1	50.27	53.58	51.04	**48.39**	51.33	52.91
PG2	65.02	65.53	67.10	67.59	**69.44**	73.25
PG3	77.05	83.16	79.32	78.77	**79.46**	88.20
PG4	62.41	62.38	65.73	**68.33**	69.51	68.72
PG5	85.10	92.74	85.82	**86.03**	89.25	86.55
PG6	178.51	185.27	185.06	186.34	**183.75**	187.42
PG7	301.20	312.86	317.65	318.77	**325.12**	351.37
PG8	831.28	845.51	838.03	841.25	**846.50**	867.92
PG9	66.84	67.85	71.39	**67.92**	68.13	70.05
PG10	129.30	143.06	131.32	135.88	**138.03**	138.47

各程序在所选较为合适的测试用例数量下的执行时间用粗体表示.

由表 3 可知, 在一定范围内增加种群个体数并不会给测试带来较大额外时间消耗, 这是因为相较于随机生成的种群, 经约简后其质量得到提高. 部分程序随种群数量的增加, 执行时间略有减少, 例如在程序 PG1 和 PG3 上, 种群数量为 160 时, 执行时间均少于种群数量为 120 时, 其主要原因是: 增加的种群个体质量有所提升, 在进化过程中可更早地覆盖目标路径, 减少迭代次数, 进而减少交叉变异的时间消耗. 此外, 结合图 2, 相较于时间, 用例数量对覆盖率影响更为明显. 因此, 在选择个体时应优先考虑用例数对目标路径覆盖率的影响, 再考虑执行时间.

具体地, 结合图 2 和表 3 可知, PG1、PG4、PG5、PG6、PG9、PG10 这 6 个程序在用例数量 300 以内时, 覆盖率呈波动变化. 其中, 程序 PG1、PG5 在用例数量为 160 时覆盖率最佳, 且相较于数量为 200 和 300 时时间消耗最少; 程序 PG4、PG9 的覆盖率在用例数量为 160 时与在 200 和 300 时相当, 但执行时间更少; 程序 PG6 在用例数量为 200 时覆盖率最佳, 且相较于 160 与 300 时所需执行时间最少; 程序 PG10 在各用例数量情况下所需执行时间相差不大, 但在 200 时覆盖率最佳. 在执行时间方面, 程序 PG1 在用例数量为 160 时, 所需执行时间最少; PG4、PG5、PG9 在用例数量为 160 时相较于数量为 200 和 300 时时间消耗最少; PG6 在用例数量为 200 相较于 160 与 300 时, 所需执行时间最少; 程序 PG10 在各用例数量情况下所需执行时间相差不大, 但在 200 时覆盖率最佳.

而程序 PG2、PG3、PG7、PG8 这 4 个程序在用例数量为 200 以内时, 覆盖率增加较为平缓, 在 200 时达到最佳; 在 200–300 间覆盖率有所增加. 在执行时间方面, 用例数量为 200 以内时, 增加缓慢, 且在 200 时所消耗时间与其他组相当; 用例数量为 300 时, 它们的执行时间均呈显著性增加.

结合以上分析, 基于优先考虑覆盖率的同时, 权衡测试的执行时间消耗, 分别将程序 PG1–PG10 的初始种群数设置为 160、200、200、160、160、200、200、200、160、200.

2) 约简集中聚类簇数对测试效率的影响

本文针对不同簇构建测试用例约简集, 因此聚类簇数对测试效率也有一定影响. 这里, 考虑到不同程序的目标路径存在差异 (见表 1), 以及执行程序的时间消耗, 将目标路径数超过 50 的程序 (PG6–PG8) 的随机种子数设为 800, 其余均设为 400, 并采用常用的手肘法 (见公式 (2)), 针对种群大小选择不同程序的聚类簇数, 结果如图 3 所示.

由图 3 可知, 聚类簇数的选择与程序规模, 以及目标路径数息息相关. 其中, 规模较小且目标路径数较少的程序 PG1–PG5, 最优聚类簇数为 3; 规模较大且目标路径数较多的程序 PG6–PG8, 最优聚类簇数分别为 4、5、5. 相较于程序 PG6–PG8, 程序 PG9、PG10 虽然程序规模较大, 但目标路径数较少, 因此, 它们最优聚类簇数小于 5. 由此可见, 针对代码行数或被测目标路径数较多的测试对象, 可适当增加用例数以及聚类簇数.

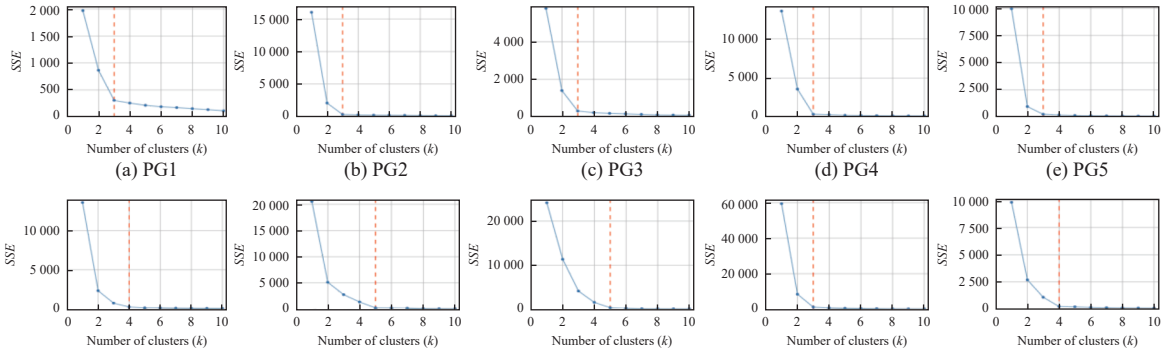

图 3 不同程序的聚类簇数选取

此外, 聚类簇数在一定程度上反映了种群的多样性. 为验证多样性种群对多路径用例生成的影响, 设置不同聚类簇数 (根据图 3, 可设置聚类簇的个数上限为 5), 根据测试用例约简法获取每个簇中的用例, 对比它们在不同程序上的用例生成效率, 结果如表 4 所示.

由表 4 可知, 在大部分程序上, 随着聚类簇数的增加, 测试效率均得到一定改善.

(1) 在程序 PG1–PG6、PG9、PG10 上, 当聚类簇数达到设定的最优簇数 (见图 3) 时, 测试效率普遍最佳. 其原因是: 当聚类簇数较少时, 测试用例种类较为单一, 遗传进化时易出现局部最优收敛的问题, 较难生成可覆盖多目标路径的用例. 当聚类簇数持续增加时, 覆盖率并未有明显提高; 进化代数总量和平均进化代数有时呈略微增长的趋势; 执行时间也随之增加. 例如, 对于程序 PG4, 在聚类簇数为 3 (最优值) 时, 覆盖率达 85%, 进化代数总量为 95 163, 平均进化代数为 237.90, 执行时间为 69.51 s; 当聚类簇数增加到 5 时, 覆盖率仅提高 1%, 执行时间则需多消耗

5.12 s. 其主要原因是, 当聚类簇数超过最优值时, 不同聚类簇中的用例存在一定的耦合, 导致存在较多相似用例, 破坏了最佳测试簇的多样性, 降低了在遗传进化时选中优秀种群基因的可能性. 此外, 前期也需消耗更多的时间以选取测试用例, 这必会导致测试时间增加.

表 4　不同聚类簇数对用例生成效果的对比

待测程序	评价指标	1	2	3	4	5
PG1	覆盖率 (%)	65	82	**88**	88	90
	进化代数总量	58741	42873	**36034**	35784	33684
	平均进化代数	419.21	245.43	**193.08**	185.71	172.12
	执行时间 (s)	56.74	52.12	**48.39**	47.69	50.70
PG2	覆盖率 (%)	57	76	**93**	92	93
	进化代数总量	87487	48632	**45368**	46735	47867
	平均进化代数	507.24	366.47	**121.10**	128.34	131.89
	执行时间 (s)	74.30	63.88	**69.44**	70.12	65.73
PG3	覆盖率 (%)	52	68	**75**	76	78
	进化代数总量	69712	57745	**47032**	49962	50175
	平均进化代数	636.43	545.37	**349.43**	356.78	351.99
	执行时间 (s)	87.78	84.34	**79.46**	81.51	83.46
PG4	覆盖率 (%)	69	78	**85**	85	86
	进化代数总量	153649	107156	**95163**	94413	100414
	平均进化代数	398.63	306.84	**237.90**	227.67	246.32
	执行时间 (s)	79.35	75.65	**69.51**	68.72	64.39
PG5	覆盖率 (%)	60	69	**76**	74	75
	进化代数总量	246713	185936	**182885**	218710	202473
	平均进化代数	461.37	406.75	**375.55**	397.64	385.22
	执行时间 (s)	97.52	93.75	**89.25**	91.35	92.56
PG6	覆盖率 (%)	71	80	86	**91**	91
	进化代数总量	340684	248346	185422	**164720**	165153
	平均进化代数	303.46	254.28	217.75	**134.62**	153.47
	执行时间 (s)	194.04	188.38	186.91	**183.75**	183.83
PG7	覆盖率 (%)	76	83	87	90	**93**
	进化代数总量	368547	328735	285341	268745	**226839**
	平均进化代数	326.41	300.24	197.63	178.08	**166.28**
	执行时间 (s)	358.94	246.83	335.66	330.39	**325.12**
PG8	覆盖率 (%)	61	69	72	75	**79**
	进化代数总量	668326	577481	547435	533261	**479412**
	平均进化代数	525.88	437.32	367.36	353.52	**294.37**
	执行时间 (s)	897.36	876.90	863.66	854.36	**843.50**
PG9	覆盖率 (%)	63	68	**72**	72	73
	进化代数总量	238723	188736	**169005**	129745	102431
	平均进化代数	477.52	402.37	**385.90**	387.21	352.74
	执行时间 (s)	69.37	66.07	**68.92**	66.74	67.39
PG10	覆盖率 (%)	70	76	81	**84**	85
	进化代数总量	238417	199857	177549	**136331**	156874
	平均进化代数	382.22	316.30	279.36	**255.62**	279.77
	执行时间 (s)	145.95	140.87	137.68	**135.88**	138.84

注: 粗体表示在最优聚类簇数下的测试效率

(2) 在 PG7、PG8 上, 随着聚类簇数增加, 当达到最优聚类簇数 5 时测试效率最佳. 相对于其他程序, 这两种程序的目标路径数较多, 则所需用例的种类也较多. 虽在前期需花费选取合适聚类簇的时间, 但构建满足覆盖路径的初始用例约简集可减少后期种群进化时间. 例如, 在 PG8 (航海系统) 上, 当聚类簇数为 3 时, 平均进化代数为 367.36, 当聚类簇数为 5 时, 平均进化代数为 294.37, 显著减少了约 73 代.

综上可知, 当目标路径较多时, 需选择更多的聚类簇作为用例约简集的筛选子集, 提高个体的多样性, 从而提升生成覆盖多目标路径的测试效率.

3) 基于约简集的多路径覆盖测试用例生成

为充分发挥约简集在测试中的优势, 可根据上述实验针对不同待测程序设置合适的用例数量以及聚类簇数. 约简集中的用例数设置在 160–200 内较为合理, 若目标路径数较多或存在较难覆盖的路径, 则此时选取 200 个用例数, 提高约简集中可覆盖相似目标路径的用例被选中的概率. 此外, 聚类簇数对测试效率也有一定影响, 在大部分待测程序上, 其值普遍设置为 3 较为合适, 对于规模较大且目标路径数相对较多的程序, 例如 PG7 和 PG8, 可将聚类簇数设置为 5. 具体设置如表 5 所示.

表 5　不同程序的初始种群参数设置

待测程序	初始种群大小 (约简集的用例数)	聚类簇数 (k)	待测程序	初始种群大小 (约简集的用例数)	聚类簇数 (k)
PG1	160	3	PG6	200	4
PG2	200	3	PG7	200	5
PG3	200	3	PG8	200	5
PG4	160	3	PG9	160	3
PG5	160	3	PG10	200	4

接下来, 为验证测试用例的选取策略 (约简法) 对测试效率的影响, 将约简法与插桩法对比 (注意, 约简法中初始种群参数见表 5), 结果如表 6 所示.

表 6　插桩法与测试用例约简法的结果对比

待测程序	评价指标	插桩法	约简法	待测程序	评价指标	插桩法	约简法
PG1	覆盖率 (%)	77	88	PG6	覆盖率 (%)	75	91
	进化代数总量	43 252	36 034		进化代数总量	292 493	164 720
	平均进化代数	314.47	193.08		平均进化代数	291.28	134.62
	执行时间 (s)	53.38	48.39		执行时间 (s)	178.50	183.75
PG2	覆盖率 (%)	60	93	PG7	覆盖率 (%)	82	93
	进化代数总量	75 540	45 368		进化代数总量	285 126	226 839
	平均进化代数	457.07	121.10		平均进化代数	248.74	166.28
	执行时间 (s)	65.02	69.44		执行时间 (s)	301.20	325.12
PG3	覆盖率 (%)	58	75	PG8	覆盖率 (%)	65	79
	进化代数总量	64 814	47 032		进化代数总量	615 621	479 412
	平均进化代数	612.43	349.43		平均进化代数	498.05	294.37
	执行时间 (s)	77.05	79.46		执行时间 (s)	831.28	843.50
PG4	覆盖率 (%)	73	85	PG9	覆盖率 (%)	67	72
	进化代数总量	128 470	95 163		进化代数总量	175 379	169 005
	平均进化代数	335.32	237.90		平均进化代数	430.67	385.90
	执行时间 (s)	62.41	69.51		执行时间 (s)	66.84	68.92
PG5	覆盖率 (%)	64	76	PG10	覆盖率 (%)	71	84
	进化代数总量	215 473	182 885		进化代数总量	190 217	136 331
	平均进化代数	425.38	375.55		平均进化代数	357.07	255.62
	执行时间 (s)	85.10	89.25		执行时间 (s)	129.30	135.88

由表 6 可知, 对比约简法与插桩法, 可得到如下几点结论.

(1) 覆盖率、进化代数方面. 在覆盖率上, 约简法较插桩法提升明显. 例如, 在目标路径数较多的程序 PG5 (游戏程序) 上, 覆盖率可提高 9%; 在 PG7 (功能较为复杂的网站登录系统) 上提高了 11%; 在 PG8 (程序规模较大且目标路径数较多的航海系统) 上提升显著, 达 14%; 在程序 PG9 (非交互式文本处理工具) 上提高相对较少, 但也有 5% 的提高. 在进化代数上, 约简法均有所减少, 这表明在同一进化代数内, 约简集中的种群个体信息较为丰富, 可覆盖更多目标路径.

(2) 执行时间方面. 约简法在执行时间上一般高于插桩法, 但不明显. 例如, 在路径高度冗余程序 PG3 (银行卡号校验系统)、PG9 上的时间高出约 2 s, 因在约简测试用例时需花费一定时间, 另外约简集的用例数较大, 在遗传进化中也必然会消耗更多时间.

综上分析, 在多测试用例覆盖相同或相似路径而产生高冗余路径的程序或目标路径数较多的应用上, 约简集可改善初始种群质量, 使得在相对较少的进化代数内生成目标用例, 但稍许耗时. 接下来验证本文方法所结合的 SVR 模型在测试时间方面的优势.

5.4.2 适应度预测实验

针对问题 2(a), 随机选取一个功能较单一程序与功能较复杂程序, 将 SVR 预测的用例适应度值与插桩法计算的精确值对比, 分析 SVR 模型的性能, 如图 4 所示. 其中, 横坐标为用例编号, 纵坐标为用例的适应度差值百分比 (即, SVR 模型获取的预测值与传统插桩方式得到的精确值之差占精确值的百分比).

由图 4 可知, 本文方法模拟预测的适应度值与执行插桩程序获取的精确值之差的百分比分布在较小的范围内, 但不排除极少数预测值与精确值差值较大的个体 (在图 4 中已用三角形标记). 功能较单一程序的差值百分比集中分布在 [−10%, 9%], 功能较复杂程序的集中分布在 [−5%, 6%].

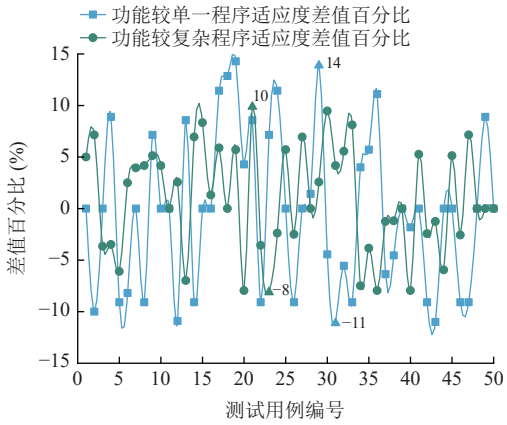

图 4 适应度值之差百分比结果

从适应度值之差百分比曲线可知, 本文的 SVR 模型预测结果即使与精确值存在一定差异, 但差值会在较小的范围内. 我们将适应度在一定范围内的个体输入程序并计算其准确的适应度值, 让误差降低到最小. 由此可知: SVR 模型模拟测试用例适应度的效果良好, 适合利用训练模型代替插桩法进行适应度的获取.

针对问题 2(b), 将 SVR 模型应用于测试用例的进化生成中, 并与插桩法对比分析, 验证 SVR 模型对测试效率的影响. 结果如表 7 所示.

根据表 7, 可得如下几点结论.

(1) 覆盖率、进化代数等方面. 插桩法的覆盖率普遍高于 SVR 法, 但相差较小. 在程序 PG8 上, 两种方法差值达到最大, 但差值也仅为 9%; 在程序 PG2 上差值最小, 仅为 1%. 在进化代数方面, 插桩法略少于 SVR 法. 插桩法在覆盖率和进化代数方面略优于 SVR 法的主要原因是, 插桩法得到的适应度值是准确值, 而 SVR 模型预测的适应度值与准确值存在一定偏差, 这导致在利用 SVR 法选择优秀个体进化生成目标用例时的准确性有所降低, 故在

这几个指标上 SVR 法略逊于插桩法.

(2) 执行时间方面. 随着程序规模的扩大, 插桩法在时间方面的消耗呈现显著上升趋势, 而 SVR 法表现出相对平缓的增长, 且 SVR 法在大部分程序上优于插桩法. 在规模较小的程序 PG1 和 PG2 上, 插桩法的执行时间少于 SVR 法, 分别减少 3.96 s 和 0.36 s. 但随着程序规模及其复杂度的增加, SVR 法在执行时间方面越显优势. 例如, 在稍大程序 PG3、PG4 与 PG5 上, SVR 法比插桩法分别节约了 8.90 s、9.49 s 与 12.51 s; 在规模最大的程序 PG8 上更是节约了 152.04 s. 可见, SVR 法对于规模相对较大的程序更能体现执行时间方面的优势.

表 7 插桩法、SVR 法与本文方法对程序测试的结果对比

待测程序	评价指标	插桩法	SVR法	本文方法	待测程序	评价指标	插桩法	SVR法	本文方法
PG1	覆盖率 (%)	77	73	100	PG6	覆盖率 (%)	75	70	100
	进化代数总量	43252	46017	636		进化代数总量	292493	310422	74652
	平均进化代数	314.47	384.59	5.48		平均进化代数	291.28	365.83	52.26
	执行时间 (s)	53.38	57.34	57.85		执行时间 (s)	178.50	152.28	136.81
PG2	覆盖率 (%)	60	59	100	PG7	覆盖率 (%)	82	79	100
	进化代数总量	75540	76914	2308		进化代数总量	285126	295038	86326
	平均进化代数	457.07	473.41	6.86		平均进化代数	248.74	264.87	61.13
	执行时间 (s)	65.02	65.38	66.90		执行时间 (s)	301.20	248.05	263.01
PG3	覆盖率 (%)	58	51	100	PG8	覆盖率 (%)	65	56	98
	进化代数总量	64814	69837	4392		进化代数总量	615621	657962	186892
	平均进化代数	612.43	683.60	27.31		平均进化代数	498.05	530.07	85.13
	执行时间 (s)	77.05	68.15	63.17		执行时间 (s)	831.28	679.24	692.53
PG4	覆盖率 (%)	73	65	100	PG9	覆盖率 (%)	67	62	100
	进化代数总量	128470	138039	21095		进化代数总量	175379	188325	28904
	平均进化代数	335.32	427.22	35.56		平均进化代数	430.67	473.84	23.88
	执行时间 (s)	62.41	52.92	47.92		执行时间 (s)	66.84	59.72	61.24
PG5	覆盖率 (%)	64	63	100	PG10	覆盖率 (%)	71	67	97
	进化代数总量	215473	212562	20358		进化代数总量	190217	218083	43652
	平均进化代数	425.38	437.21	25.14		平均进化代数	357.07	410.32	59.98
	执行时间 (s)	85.10	72.59	65.62		执行时间 (s)	129.30	112.73	127.82

根据第 5.4.1 节与第 5.4.2 节实验可知, 约简法在覆盖率等方面具有优势, 但在执行时间方面欠佳; SVR 预测法在执行时间这一指标上表现较优, 而在多路径的覆盖率上效果欠佳. 这两种方法分别在种群质量与执行时间方面各具优势, 且互为补充. 第 5.4.3 节对这两种方法结合的可行性展开实验分析.

5.4.3 消融实验

为验证结合 K-means 用例约简法与 SVR 模型预测法的有效性, 及其对测试效率方面的综合影响, 下面设计消融实验分析.

针对问题 3 中结合 K-means 聚类与 SVR 模型对测试效率的影响, 将本文方法与插桩法对比, 由第 5.4.2 节的表 7 可得如下几点结论.

(1) 覆盖率、进化代数方面. 在简单程序 PG2 与路径冗余较高程序 PG3 上, 本文方法相较于插桩法的覆盖率提升较为明显, 分别提高 40% 和 42%; 对于规模较大程序 PG7 以及安全需求严格且目标路径数较多的安全型关键系统 PG8, 本文方法仍可有较大提升, 分别提高 18% 和 33%. 进化代数方面, 本文方法也有明显改善.

(2) 执行时间方面. 除较小程序 PG1 和 PG2 外, 本文方法在其他程序上均少于插桩法, 尤其是在较大规模程序 PG8 上, 可节约更多的执行时间, 减少达 138.75 s. 由此可见, 本文方法在执行时间方面也具有优势.

综上所述, 通过 K-means 聚类获取约简集, 并利用 SVR 模型评估个体, 可从种群质量与测试时间方面提高测试效率.

针对问题 3 中本文方法对测试效率的提升程度, 结合表 6、表 7 的实验结果, 对比插桩法、约简法、SVR 法

以及本文方法在不同评价指标上的数值差异, 以百分比形式表示, 如表 8 所示. 注意: 为说明结合 K-means 与 SVR 的有效性, 这里选取插桩法、约简法以及 SVR 法作为基准方法与对比方法. 另外, 覆盖率差值是指其他方法的覆盖率与基准方法的差值; 进化代数总量比 (或平均进化代数比/执行时间比) 是指基准方法的进化代数总量 (或平均进化代数/执行时间) 与其他方法的进化代数总量 (或平均进化代数/执行时间) 差值占基准方法的比例.

表 8　不同方法的效率对比 (%)

待测程序	评价指标	与插桩法对比			与约简法对比		与SVR法对比
		约简法	SVR法	本文方法	SVR法	本文方法	本文方法
PG1	覆盖率差值	11	−4	**23**	−15	**12**	**27**
	进化代数总量比	16.69	−6.39	**98.76**	−27.70	**98.51**	**98.84**
	平均进化代数比	38.60	−22.30	**98.26**	−99.19	**97.16**	**98.58**
	执行时间比	9.35	−7.42	**−8.37**	−18.50	**−19.55**	**−0.89**
PG2	覆盖率差值	33	−1	**40**	−34	**7**	**41**
	进化代数总量比	39.94	−1.82	**96.94**	−69.53	**94.91**	**97.00**
	平均进化代数比	73.51	−3.57	**98.50**	−290.92	**94.34**	**98.55**
	执行时间比	−6.80	−0.55	**−2.89**	5.85	**3.66**	**−2.32**
PG3	覆盖率差值	17	−7	**42**	−24	**25**	**49**
	进化代数总量比	27.44	−7.75	**93.22**	−48.49	**90.66**	**93.71**
	平均进化代数比	42.94	−11.62	**95.54**	−95.63	**92.18**	**96.00**
	执行时间比	−3.13	11.55	**18.01**	14.23	**20.50**	**7.31**
PG4	覆盖率差值	12	−8	**27**	−20	**15**	**35**
	进化代数总量比	25.93	−7.45	**83.58**	−45.06	**77.83**	**84.72**
	平均进化代数比	29.05	−27.41	**89.40**	−79.58	**85.05**	**91.68**
	执行时间比	−11.38	15.21	**23.22**	23.87	**31.06**	**9.45**
PG5	覆盖率差值	12	−1	**36**	−13	**24**	**37**
	进化代数总量比	15.12	1.35	**90.55**	−16.23	**88.87**	**90.42**
	平均进化代数比	11.71	−2.78	**94.09**	−16.42	**93.31**	**94.25**
	执行时间比	−4.88	14.70	**22.89**	18.67	**26.48**	**9.60**
PG6	覆盖率差值	16	−5	**25**	−21	**9**	**30**
	进化代数总量比	43.68	−6.13	**74.48**	−88.45	**54.68**	**75.95**
	平均进化代数比	53.78	−25.59	**82.06**	−171.75	**61.18**	**85.71**
	执行时间比	−2.94	14.69	**23.36**	17.13	**25.55**	**10.16**
PG7	覆盖率差值	11	−3	**18**	−14	**7**	**21**
	进化代数总量比	20.44	−3.48	**69.72**	−30.06	**61.94**	**70.74**
	平均进化代数比	33.15	−6.48	**75.42**	−59.29	**63.24**	**76.92**
	执行时间比	−7.94	17.65	**12.68**	23.71	**19.10**	**−6.03**
PG8	覆盖率差值	14	−9	**33**	−23	**19**	**42**
	进化代数总量比	22.13	−6.88	**69.64**	−37.24	**61.02**	**71.60**
	平均进化代数比	40.90	−6.43	**82.91**	−80.07	**71.08**	**83.94**
	执行时间比	−1.47	18.29	**16.69**	19.47	**17.90**	**−1.96**
PG9	覆盖率差值	5	−5	**33**	−10	**28**	**38**
	进化代数总量比	3.63	−7.38	**83.52**	−11.43	**82.90**	**84.65**
	平均进化代数比	10.40	−10.02	**94.46**	−22.79	**93.81**	**94.96**
	执行时间比	−3.11	10.65	**8.38**	13.35	**11.14**	**−2.55**
PG10	覆盖率差值	13	−4	**26**	−17	**13**	**30**
	进化代数总量比	28.33	−14.65	**77.05**	−59.97	**67.98**	**79.98**
	平均进化代数比	28.41	−14.91	**83.20**	−60.52	**76.54**	**85.38**
	执行时间比	−5.09	12.82	**1.14**	17.04	**5.93**	**−13.39**

本文方法的实验结果用粗体表示.

根据表 8, 可得如下几点结论.

(1) 与插桩法相比, 在覆盖率方面, 约简法与本文方法均有提高, SVR 法略有降低. 其中, 约简法最少提升 5%, 最多提升达 33%; 本文方法最少提升 18%, 最多提升 42%; SVR 法最多降低 9%. 在进化代数总量、平均进化代数方面, 约简法与本文方法均有减少, 这说明采用约简集可尽早地生成目标测试用例; SVR 法有所增加, 这是由于 SVR 模型预测个体适应度存在偏差, 并未对初始种群进行择优筛选. 执行时间方面, SVR 法与本文方法具有明显优势, 而约简法耗时略多.

(2) 与约简法相比, SVR 法的优势体现在执行时间方面, 最少减少 5.85%, 最多可达 23.87%, 而本文方法在各评价指标下均最优 (较小程序 PG1 除外).

(3) 与 SVR 法相比, 本文方法可在较少的进化代数内即可生成目标测试用例. 执行时间方面, 本文方法在部分被测程序上略高于 SVR 法, 相差不大, 主要是因为 SVR 法仅是本文方法的一部分, 而本文方法无法避免初始测试用例约简集筛选以及 SVR 模型训练的时间, 从而导致测试所需时间稍有提高.

从以上消融实验的结果可看出, 仅采用 K-means 聚类约简法与仅采用 SVR 法启发生成测试用例, 它们只从某些方面改善测试生成进程, 而本文方法将两者结合, 对于规模较大、路径冗余性高或数量较多等的工业系统, 以及安全关键型系统, 可在覆盖率、进化代数、执行时间方面综合提高测试效率.

5.4.4 对比实验

为进一步评估问题 3 中结合 K-means 与 SVR 的测试用例生成效率, 下面将本文方法与其他相关典型方法展开对比, 包括关键点概率[15]、DSGEO[13]、HSADE[14]、DNTSA[3]和 MfO-PC[1]. 这些方法均聚焦于多路径用例生成, 研究目标与本文相近, 因而可作为代表性的对比方法.

接下来, 将本文方法与上述经典的测试用例生成方法进行对比. 特别地, 为更清晰地呈现本文方法相较于这些对比方法的优越性, 我们延伸第 5.2 节中提及的“覆盖率”这一指标, 设计“覆盖率差值” (本文方法与对比方法的覆盖率之差) 指标. 各方法在不同程序上的多指标实验结果如表 9 所示.

表 9 6 种方法在不同程序上的多指标对比

评价指标	PG9		PG11		PG2		PG4		PG14	
	关键点概率[15]	本文方法	DSGEO[13]	本文方法	HSADE[14]	本文方法	DNTSA[3]	本文方法	MfO-PC[1]	本文方法
目标路径数	30	12	398	20	4	5	15	20	2	2
覆盖率 (%)	100	100	95	100	100	100	100	100	100	100
平均进化代数	6292	23.88	—	63.76		23.88	—	91.68	—	68.12
覆盖率差值 (%)	0		5		0		0		0	

评价指标	PG10		PG12		PG13		PG15		PG15	
	关键点概率[15]	本文方法	DSGEO[13]	本文方法	HSADE[14]	本文方法	DNTSA[3]	本文方法	MfO-PC[1]	本文方法
目标路径数	20	20	2537	30	33	33	9	9	9	9
覆盖率 (%)	100	97	70	95	100	100	66.67	78	54.5	78
平均进化代数	5947	59.98	—	112.38	—	157.45	—	379.65	—	379.65
覆盖率差值(%)	−3		25		0		11.33		23.5	

注: ① 因PG9–PG15为非Python代码程序, 为便于本文方法与其他经典方法进行比较, 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, 由于不同的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

由表 9 可知, 相较于其他几种方法, 本文方法总体上占优, 分析如下.

(1) 对比关键点概率、DNTSA 以及 MfO-PC 这 3 种基于搜索的测试用例生成方法, 本文方法的覆盖率在大部分程序上有显著提升. 例如, 在程序 PG15 (资源管理系统) 上最少提升 11.33%, 最多提升 23.5%. 在程序 PG10 (词法分析器) 上, 本文方法虽稍逊于关键点概率法, 但在平均进化代数上, 仅需 59.98 代, 远少于关键点概率法的 5947 代. 可见, 本文方法优势比较明显, 这是因为在初始种群阶段经过种群多样性的筛选, 保证携带优秀基因尽早地被选中.

(2) 对比 DSGEO 和 HSADE 这两种采用代理模型生成测试用例的方法, 本文方法在覆盖率上表现较好. 相较于 DSGEO, 所提方法的覆盖率有所提高, 例如在 PG11 上提升 5%, 在 PG12 上提升 25%. 在 PG2 和 PG13 上, 我们的覆盖率与 HSADE 方法均达到 100%. 可见, 相较于代理模型的方法, 我们前期对测试用例进行聚类与排序, 不仅有助于改善初始种群的质量, 还具备种群多样性, 提高了多目标路径的覆盖率, 且提出的 SVR 适应度预测模型预测误差较小, 降低了优秀用例的适应度预测结果偏差较大的风险.

从以上几种对比方法的结果可看出, 本文方法融合了 K-means 聚类及 SVR 模型具有一定优势. 一方面, 通过 K-means 聚类降低初始用例的冗余性, 同时使其具备多样化, 提高优秀用例尽早被选中的概率; 另一方面, SVR 模型在保证多路径个体适应度预测结果准确性的同时, 大幅减少执行待测程序获取个体适应度的时间.

为进一步阐明本文采用的 K-means 聚类策略与 SVR 模型在时间复杂度上的优势, 我们选取较新的利用代理模型生成测试用例的方法 DSGEO[13]和 HSADE[14]进行对比分析, 如表 10 所示.

表 10　模型时间复杂度对比

对比项	DSGEO[13]	HSADE[14]		本文方法	
	RBFN模型	GPR模型	RBFN模型	K-means策略	SVR模型
单模型	—	$O(N^2D+N^3)$	$O(KND)+O(N^2D)+O(N^3)$	$O(KND)$	$O(N^2)$
多模型	$M\times(O(KND)+O(N^2D)+O(N^3))$	—		—	
多目标路径	$Num_{tar}\times M\times O_{RBFN}(\cdot)$	$Num_{tar}\times((O_{GPR}(\cdot)+O_{RBFN}(\cdot))$		$<<O_{K\text{-}means}(\cdot)+Num_{tar}\times O_{SVR}(\cdot)$	
整体复杂度	$O(N^3)$	$O(N^3)$		$O(N^2)$	

由表 10 可知, 相较于 DSGEO 和 HSADE 这两种方法, 我们采用的 K-means 策略与 SVR 模型更具轻量级, 时间复杂度较低.

(1) 就模型而言, DSGEO 方法采用的径向基函数神经网络模型 (RBFN) 的时间复杂度为 $O(KND+N^2D+N^3)$, 分别为利用 K-means 选择神经元中心向量的时间复杂度 $O(KND)$, 计算核函数的时间复杂度 $O(N^2D)$, 以及求解模型权重的时间复杂度 $O(N^3)$, 其中 K 为聚类中心数, N 为训练样本数 (即参与训练的测试用例数), D 为测试用例的维度. HSADE 方法结合了高斯过程回归模型 (GPR) 与 RBFN 模型, 其中 GPR 的时间复杂度为训练用例构成的协方差矩阵以及求解该逆矩阵的时间复杂度之和 $O(N^2D+N^3)$. 本文模型的时间复杂度为 K-means 聚类的复杂度 $O(KND)$ 与 SVR 求解优化目标函数的时间复杂度 $O(N^2)$ 之和. 与 DSGEO 和 HSADE 方法相比, 我们方法采用的模型时间复杂度较低. 由此可知, 各方法使用模型的时间复杂度关系为 HSADE＞DSGEO＞本文方法.

(2) 本文方法与 HSADE 方法均采用单模型进行个体适应度预测, 而 DSGEO 方法采用多模型集成的代理模型预测个体适应度, 时间复杂度更高, 为 $M\times(O(KND+N^2D+N^3))$, 其中 M 为模型的个数. 另外, 本文构建的模型可应用于多目标路径生成中, 模型的时间复杂度远小于 $O_{K\text{-}means}(\cdot)+Num_{tar}\times O_{SVR}(\cdot)$, 其中 Num_{tar} 为目标路径数; 而 DSGEO 方法与 HSADE 方法的代理模型仅针对单目标路径进行个体适应度预测, 若用于多目标路径覆盖中, 它们的模型时间复杂度分别为 $Num_{tar}\times M\times O_{RBFN}(\cdot)$ 和 $Num_{tar}\times(O_{GPR}(\cdot)+O_{RBFN}(\cdot))$, 这显著高于本文方法. 故本文模型在多路径覆盖用例生成中时间复杂度更低. 此外, 本文方法采用的 SVR 模型在小样本数据集上的准确性优于 GPR 与 RBFN 两类模型, 泛化性更高.

综上分析, 在多路径覆盖的测试用例生成中, 相比于其他代理模型, 利用 K-means 聚类策略与 SVR 模型在时间复杂度方面更占优.

5.5　回答问题

通过分析实验结果, 对第 5.1 节中提出的几个问题做如下回答.

问题 1: 测试用例约简集能否改善初始种群质量, 推进遗传进化过程, 提高多目标路径测试用例生成效率?

由第 5.4.1 节图 2 可知, 在被测的 10 个程序上, 约简集影响目标路径的覆盖率, 将其融入遗传算法中可使得大部分待测程序的覆盖率达 75% 以上. 结合图 2、图 3 和表 3、表 4 可知, 合适的聚类簇数可在提高覆盖率的同时, 尽量减少测试时间消耗. 此外, 由表 6 可知, 与插桩法相比, 约简法在大部分指标上具有明显改善. 例如, 虽然在

PG2、PG3、PG6 的执行时间上稍多于插桩法, 但在覆盖率上至少可提升 12%. 综合来看, 约简集可提高对目标用例生成效率.

问题 2: (a) SVR 模型模拟个体适应度的准确性如何? (b) 利用 SVR 模型能否提高测试效率?

由第 5.4.2 节图 4 可知, SVR 模型模拟用例适应度的效果良好, 在功能较单一程序上适应度差值百分比在 [−11%, 14%] 范围内, 功能较复杂程序上适应度差值百分比在 [−8%, 10%] 范围内. 此外, 由表 7 可知, 在路径覆盖率及平均进化代数方面, 利用 SVR 模型预测个体适应度值的 SVR 法与插桩法均相差较小. 在时间方面, 随着程序规模及其复杂度增加, SVR 法的执行时间显著少于插桩法, 在规模较大程序上的时间减少可达 18.29%. 由于 SVR 模型模拟个体适应度的时间与测试用例本身密切相关, 不会随着程序规模增加而有太大变化, 故 SVR 法随程序规模的增加, 其时间优势越明显.

问题 3: 结合 K-means 与 SVR 模型的测试用例约简与生成方法 (本文方法) 是否可提高测试效率? 其效率可提高到何种程度?

由第 5.4.3 节表 8 可知, 仅采用 K-means 的约简法在覆盖率、进化代数方面效果较优, 但在执行时间方面略微耗时, 需多花费约 1.47%–11.38%. 仅采用 SVR 模型 (SVR 法) 在覆盖率、进化代数方面效果欠佳, 但在执行时间方面优势明显, 可减少约 10.65%–18.29%. 在规模较大、目标路径数量较多的工业系统, 或安全需求较高的安全关键型系统上, 本文方法将两者结合, 覆盖率可提高约 9%–49%, 进化代数总量降低约 61.02%–98.84%, 平均进化代数降低约 75.42%–98.58%, 执行时间减少约 1.14%–23.36%. 由第 5.4.4 节表 9 也可看出, 相较于其他测试方法, 本文方法在多路径覆盖测试用例生成方面有所改进, 其中在覆盖率上可提高约 10%, 最多可达 25%. 表 10 也清晰地表明, 相较于其他方法采用的模型, 本文构建的模型在时间复杂度上占优. 由此可见, 结合 K-means 与 SVR 模型可有效提高测试用例生成效率.

6 总结与下一步工作

本文提出一种结合 K-means 聚类与 SVR 模型的测试用例约简与生成策略, 提高测试效率, 主要贡献如下.

1) 利用 K-means 生成测试用例约简集的方法可减少冗余数据的适应度计算次数. 通过 K-means 方法对初始测试用例集聚类, 根据个体潜能以及目标路径覆盖难易程度获取约简集, 并以此作为遗传进化的初始种群. 该方法可减少相似测试用例, 提高初始种群质量, 让优秀个体尽早地参与进化生成, 提高测试效率.

2) 通过 SVR 模型预测个体适应度的方法可有效减少测试执行时间. 训练可预测个体适应度的 SVR 模型, 并根据其预测结果的准确性更新模型. 利用该模型评估遗传进化中种群个体的适应度, 并对部分个体进行插桩验证. 该模型在保证预测结果相对准确的前提下, 尽可能减少执行插桩程序的次数, 这可节约较多测试时间.

3) 结合 K-means 聚类与 SVR 模型可从覆盖率、进化代数、执行时间等方面共同提高用例生成效率. 基于 K-means 约简方法在覆盖率、进化代数方面效果较好, 在执行时间方面略微耗时; 而 SVR 多路径个体适应度预测模型可明显缩减测试时间, 这恰好弥补了前期构建约简集带来的部分时间消耗. 两种方法相辅相成, 共同提高测试效率. 所提方法能有效地适用于程序规模较大、目标路径较难覆盖, 以及路径数量较多的工业程序测试场景. 在测试资源受限的环境中 (如信息验证与管理系统), 或安全需求较高而对测试时间限制不太严格的系统 (如航海系统) 等, 结合 K-means 约简集与 SVR 适应度预测模型能够在较少的进化代数与合理的执行时间内, 提升测试效率.

下一步工作中, 拟构建测试用例与目标路径关系的聚类模型, 并在种群进化中自适应调整聚类簇, 提高用例约简集的质量以及优秀个体在演化中的优先选择概率. 此外, 考虑将自适应策略与具备动态调整机制的强化学习模型相结合, 学习测试用例与其适应度间的关系, 以适应动态变化的测试环境, 进一步拓宽约简集与适应度预测模型的测试应用场景.

References

[1] Wang XP, Hu ZB, Shi LY, Cai GC, Su QH. Multi-task modeling and multifactorial optimization for path coverage problem of automated test case generation. Applied Soft Computing, 2024, 154: 111407. [doi: 10.1016/j.asoc.2024.111407]

[2] Perera A, Aleti A, Turhan B, Böhme M. An experimental assessment of using theoretical defect predictors to guide search-based software testing. IEEE Trans. on Software Engineering, 2023, 49(1): 131–146. [doi: 10.1109/TSE.2022.3147008]

[3] Guo CJ, Tang Y, Li XB, Ding PH. Automated test case generation based on path structure matrix and mainfold-inspired search. In: Proc. of the 27th Int'l Conf. on Computer Supported Cooperative Work in Design. Tianjin: IEEE, 2024. 1845–1851. [doi: 10.1109/CSCWD61410.2024.10580654]

[4] Wang WW, Li YC, Zhao RL, Li Z. Parallel test case generation based on front and back end of Web applications with genetic algorithm. Ruan Jian Xue Bao/Journal of Software, 2020, 31(5): 1314–1331 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5955.htm [doi: 10.13328/j.cnki.jos.005955]

[5] 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, 2022. 650–654. [doi: 10.1145/3495018.3495135]

[6] Li JY, Zhan ZH. Expensive multi-objective evolutionary algorithm with multi-objective data generation. Chinese Journal of Computers, 2023, 46(5): 896–908 (in Chinese with English abstract). [doi: 10.11897/SP.J.1016.2023.00896]

[7] 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]

[8] 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]

[9] Chen JF, Chen HB, Guo YC, Zhou MM, Huang RB, Mao CY. A novel test case generation approach for adaptive random testing of object-oriented software using K-means clustering technique. IEEE Trans. on Emerging Topics in Computational Intelligence, 2022, 6(4): 969–981. [doi: 10.1109/TETCI.2021.3122511]

[10] Pei HY, Yin BB, Xie M, Cai KY. Dynamic random testing with test case clustering and distance-based parameter adjustment. Information and Software Technology, 2021, 131: 106470, [doi: 10.1016/j.infsof.2020.106470]

[11] Cui ZQ, Zhang JM, Zheng LW, Chen X. A survey of research on coverage-guided greybox fuzzing. Chinese Journal of Computers, 2024, 47(7): 1665–1696 (in Chinese with English abstract). [doi: 10.11897/SP.J.1016.2024.01665]

[12] 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 (TOSEM), 2021, 30(2): 17. [doi: 10.1145/3423132]

[13] Sun BC, Gong DW, Yao XJ. Integrating DSGEO into test case generation for path coverage of MPI programs. Information and Software Technology, 2023, 153: 107068. [doi: 10.1016/j.infsof.2022.107068]

[14] Gao L, Bai SY, Liu MX, Li F. Automated test case generation for path coverage using hierarchical surrogate-assisted differential evolution. Applied Soft Computing, 2024, 158: 111586. [doi: 10.1016/j.asoc.2024.111586]

[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] 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]

[17] Amarif M, Alfitouri A, Allag Z. An automatic generation of test cases in JUnit testing framework using genetic algorithm. In: Proc. of the 11th Int'l Conf. on Systems and Control. Sousse: IEEE, 2023. 312–317. [doi: 10.1109/ICSC58660.2023.10449836]

[18] Ji SH, Zhu SQ, Zhang PC, Dong H, Yu JN. Test-case generation for data flow testing of smart contracts based on improved genetic algorithm. IEEE Trans. on Reliability, 2023, 72(1): 358–371. [doi: 10.1109/TR.2022.3173025]

[19] Wang XY, Muqeet A, Yue T, Ali S, Arcaini P. Test case minimization with quantum annealers. ACM Trans. on Software Engineering and Methodology, 2025, 34(1): 5. [doi: 10.1145/3680467]

[20] Pradhan D, Wang S, Ali S, Yue T, Liaaen M, 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]

[21] Chen JF, Gu YC, Cai SH, Chen HB, Chen JY. A novel test case prioritization approach for black-box testing based on K-medoids clustering. Journal of Software: Evolution and Process, 2024, 36(4): e2565. [doi: 10.1002/smr.2565]

[22] 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, 2024, 35(6): 2795–2820 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6905.htm [doi: 10.13328/j.cnki.jos.006905]

[23] 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 (EIT). Chicago: IEEE, 2020. 469–474. [doi: 10.1109/EIT48999.2020.9208306]

附中文参考文献

[4] 王微微, 李奕超, 赵瑞莲, 李征. Web 应用前后端融合的遗传算法并行化测试用例生成. 软件学报, 2020, 31(5): 1314–1331. http://www.jos.org.cn/1000-9825/5955.htm [doi: 10.13328/j.cnki.jos.005955]

[6] 黎建宇, 詹志辉. 基于多目标数据生成的昂贵多目标进化算法. 计算机学报, 2023, 46(5): 896–908. [doi: 10.11897/SP.J.1016.2023.00896]

[7] 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]

[8] 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]

[11] 崔展齐, 张家铭, 郑丽伟, 陈翔. 覆盖率制导的灰盒模糊测试研究综述. 计算机学报, 2024, 47(7): 1665–1696. [doi: 10.11897/SP.J.1016.2024.01665]

[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]

[22] 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, 2024, 35(6): 2795–2820. http://www.jos.org.cn/1000-9825/6905.htm [doi: 10.13328/j.cnki.jos.006905]

作者简介

钱忠胜, 博士, 教授, 博士生导师, CCF 高级会员, 主要研究领域为软件工程, 智能化软件, 机器学习.

俞情媛, 博士生, 主要研究领域为软件工程, 智能化软件, 机器学习.

范赋宇, 硕士生, CCF 学生会员, 主要研究领域为软件工程, 机器学习.

许克文, 硕士生, 主要研究领域为软件工程, 机器学习.

陈超, 硕士生, 主要研究领域为软件工程, 机器学习.