软件学报 ISSN 1000-9825, CODEN RUXUEW
2025,36(5):2026−2042 [doi: 10.13328/j.cnki.jos.007204] [CSTR: 32375.14.jos.007204]

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

代理辅助多任务进化优化引导的 MPI 程序路径覆盖测试用例生成*

孙百才[1,3], 巩敦卫[2], 姚香娟[3]

[1](中国矿业大学 信息与控制工程学院, 江苏 徐州 221116)

[2](青岛科技大学 信息科学技术学院, 山东 青岛 266061)

[3](中国矿业大学 数学学院, 江苏 徐州 221116)

通信作者: 巩敦卫, E-mail: dwgong@vip.163.com

摘　要: 基于进化优化的消息传递接口 (message-passing interface, MPI) 程序路径覆盖测试中, 进化个体适应值的评价需要反复执行 MPI 程序, 而程序的重复执行往往需要高昂的计算成本. 鉴于此, 提出一种代理辅助多任务进化优化引导的 MPI 程序路径覆盖测试用例生成方法, 该方法能够显著约减 MPI 程序的实际执行次数, 进而提高测试效率. 首先, 面向 MPI 程序目标路径内每条目标子路径, 训练相应的代理模型; 然后, 基于对应每条目标子路径的代理模型, 估计相应测试用例生成优化任务中进化个体的适应值, 并形成候选测试用例集; 最后, 基于候选测试用例集及其面向每条目标子路径的真实适应值, 更新对应每条目标子路径的代理模型. 将所提方法应用于 7 个基准 MPI 程序的基本路径覆盖测试中, 并与其他若干先进方法比较. 实验结果表明, 所提方法能够在确保测试用例生成高有效性的前提下, 显著提高测试效率.

关键词: 路径覆盖测试用例生成; 代理辅助多任务进化优化; 候选测试用例集

中图法分类号: TP311

中文引用格式: 孙百才, 巩敦卫, 姚香娟. 代理辅助多任务进化优化引导的MPI程序路径覆盖测试用例生成. 软件学报, 2025, 36(5): 2026–2042. http://www.jos.org.cn/1000-9825/7204.htm

英文引用格式: Sun BC, Gong DW, Yao XJ. Test Case Generation for Path Coverage of MPI Program Guided by Surrogate-assisted Multi-task Evolutionary Optimization. Ruan Jian Xue Bao/Journal of Software, 2025, 36(5): 2026–2042 (in Chinese). http://www.jos.org.cn/1000-9825/7204.htm

Test Case Generation for Path Coverage of MPI Program Guided by Surrogate-assisted Multi-task Evolutionary Optimization

SUN Bai-Cai[1,3], GONG Dun-Wei[2], YAO Xiang-Juan[3]

[1](School of Information and Control Engineering, China University of Mining and Technology, Xuzhou 221116, China)

[2](School of Information Science and Technology, Qingdao University of Science and Technology, Qingdao 266061, China)

[3](School of Mathematics, China University of Mining and Technology, Xuzhou 221116, China)

Abstract: During the path coverage testing of a message passing interface (MPI) program based on evolutionary optimization, the fitness of evolutionary individuals needs to be evaluated by repeatedly executing the MPI program. However, repeated execution of an MPI program often requires high computational costs. Therefore, this study proposes an approach to generate test cases for path coverage of MPI programs guided by surrogate-assisted multi-task evolutionary optimization, which significantly reduces the actual execution times of MPI programs, thereby improving testing efficiency. Firstly, surrogate models are trained for each target sub-path in the target path of an

* 基金项目: 国家自然科学基金 (62302502); 中国博士后科学基金 (2022M713368); 江苏省卓越博士后计划 (2022ZB528); 中央高校基本科研业务费专项资金 (2023QN1073)
收稿时间: 2023-11-09; 修改时间: 2024-02-23; 采用时间: 2024-04-08; jos 在线出版时间: 2024-08-21
CNKI 网络首发时间: 2024-08-22

MPI program. Then, the fitness of evolutionary individuals is estimated using the surrogate model corresponding to each target sub-path, and a candidate set of test cases is formed. Finally, all surrogate models are updated based on the candidate set and the actual fitness for each target sub-path. The proposed approach is applied to the basis path coverage testing of seven benchmark MPI programs and compared with several state-of-the-art approaches. The experimental results show that the proposed approach significantly improves testing efficiency while ensuring high effectiveness in generating test cases.

Key words: test case generation for path coverage; surrogate-assisted multi-task evolutionary optimization; candidate set of test case

MPI 程序是一类由若干进程构成的并行程序, 被广泛应用于高效解决经济安全和社会发展中很多复杂的问题[1,2]. 测试一个 MPI 程序时, 采用的测试准则非常重要, 除了能够引导测试用例的生成外, 还能评价测试用例的充分性. 目前为止, 已经提出多种测试准则, 其中, 基本路径覆盖是一种同时满足分支与语句覆盖准则的测试方法, 已经成为一种常用的结构覆盖准则[3].

已知的是, MPI 程序的一条目标路径通常由每一进程内一条目标子路径组合而成. 鉴于此, 可独立分析 MPI 程序目标路径内每条目标子路径的测试用例生成问题. 考虑到路径覆盖测试用例生成问题能够被转换为最优化问题[4], 因此, 这里将每条目标子路径的测试用例生成问题转换为最优化问题, 并建立相应的优化模型. 鉴于多任务进化优化已经广泛用于并行解决若干最优化问题[5−7], 因此, 本文将面向 MPI 程序目标路径内每条目标子路径的优化模型置于一个优化任务中, 并在基于进化算法并行求解每一优化模型的过程中, 形成候选测试用例集, 用于覆盖 MPI 程序目标路径.

基于进化算法并行求解每一优化任务中优化模型的过程中, 需反复执行被测 MPI 程序, 以便评价该优化任务中进化个体适应值. 然而, 重复执行被测 MPI 程序通常需要高昂的计算成本, 这导致每一优化模型的求解变成一个昂贵优化问题. 为降低昂贵优化问题的求解成本, Wang 等人[8]提出了一种具有区域划分的代理辅助差分进化算法. 此后, Si 等人[9]研究了基于代理辅助进化算法的大规模昂贵单/多目标优化问题求解, 并表明在解决昂贵优化问题方面, 代理辅助进化算法具有明显优势. 此外, Jin 等人[10]指出进化优化可能受益于代理模型引入的估计误差, 该误差能够使得种群多样性更好和不容易陷入局部最优. 鉴于此, 本文将在目标路径内每条目标子路径的优化任务中训练一个代理模型, 用于代替被测 MPI 程序的实际执行, 以及估计进化个体面向该目标子路径的适应值, 从而降低 MPI 程序的实际执行成本.

综上, 本文提出一种代理辅助多任务进化优化引导的 MPI 程序路径覆盖测试用例生成方法. 所提方法中, 首先面向目标路径内每条目标子路径, 训练相应的代理模型; 然后, 在每条目标子路径的测试用例生成优化任务中, 基于相应的代理模型, 估计进化个体的适应值, 并选择最优进化个体, 用于形成候选测试用例集; 最后, 继续在每条目标子路径对应的优化任务中, 基于候选测试用例集及其面向每条目标子路径的真实适应值, 更新相应的代理模型.

所提方法的优势之一是基于每一优化任务从局部方面寻找覆盖目标路径内每条目标子路径的最优进化个体, 并基于最优进化个体, 形成候选测试用例集, 进而从全局方面验证某一候选测试用例是否覆盖了目标路径. 显然, 以上优势恰好符合局部寻优加速全局最优的原理. 此外, 所提方法另一优势为进一步利用代理模型来降低每一优化任务中进化个体适应值的评价成本. 结合上述分析, 所提方法将有利于高效生成 MPI 程序目标路径的测试用例.

本文第 1 节是相关工作. 所提方法在第 2 节给出, 包括: 总体框架、代理模型训练、候选测试用例集形成, 以及代理模型更新. 第 3 节将提出方法应用于若干 MPI 程序路径覆盖测试中, 并与多种方法对比, 给出实验数据与分析结果. 实验有效性的威胁在第 4 节讨论. 最后, 第 5 节总结全文, 并指出需要进一步研究的问题.

1 相关工作

1.1 基础知识

与本文相同, 文献 [4] 也研究了 MPI 程序路径覆盖测试. 因此, 文献 [4] 中涉及如下基础知识同样适用于本文: MPI 程序及其测试输入的表示、节点、子路径相似度等. 为更好地理解相关基础知识, 我们给出如图 1 所示的一个 MPI 程序示例代码.

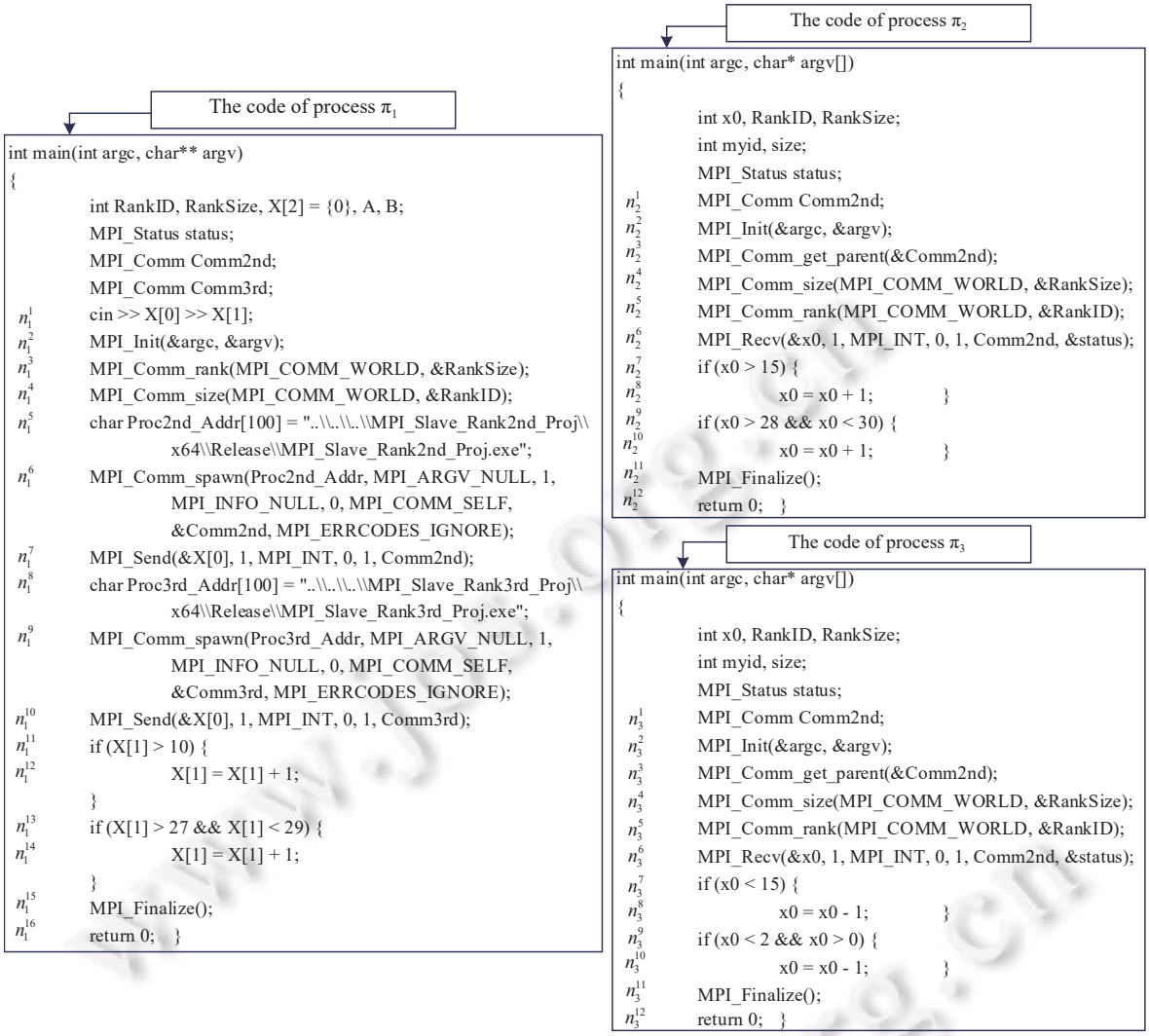

图 1 MPI 示例程序

MPI 程序的测试输入可以表示为 $X=\{x_i|i=1,2,\ldots,M\}$, 其中, x_i 是 X 中第 i 个输入变量. 一个 MPI 程序可以表示为 $\Pi=\{\pi_j|j=1,2,\ldots,N\}$, 其中, π_j 表示 Π 中第 j 个进程, N 表示进程数.

● 节点. 进程 π_j 中基本执行的单元称为节点, 通常对应于一或若干顺序执行的语句、一个循环或判断条件语句、一个通信原语. 这里, 我们将一个位于 π_j 内的节点表示为 n_j^l, 其中, l 表示 π_j 内第 l 个节点.

● 路径. 以测试输入 X 执行一个 MPI 程序 Π 时, X 遍历进程 π_j 中的节点组成一条子路径, 记为 $p_j(X)$. $p_j(X)$ 中包含的节点数, 称为 $p_j(X)$ 的子路径长度, 记为 $|p_j(X)|$. 此时, X 执行 Π 中的一条完整路径由每一进程内遍历子路径组成, 记为 $P(X)=\{p_j(X)|j=1,2,\ldots,N\}$. 此外, 一条目标路径被表示为 $P^*=\{p_j^*|j=1,2,\ldots,N\}$, 其中, p_j^* 为 P^* 中第 j 条目标子路径. 文献 [4] 中, 通过公式 (1) 计算目标子路径 p_j^* 与遍历子路径 $p_j(X)$ 的相似度. 注意, 这里也将 $Sim(p_j^*,p_j(X))$ 称为 X 面向 p_j^* 的覆盖程度.

$$Sim(p_j^*,p_j(X))=\frac{|p_j^*\cap p_j(X)|}{\max\{|p_j^*|,|p_j(X)|\}} \tag{1}$$

其中, $|p_j^*\cap p_j(X)|$ 是从第 1 个节点开始, p_j^* 和 $p_j(X)$ 连续相同节点的数量. $\max\{|p_j^*|,|p_j(X)|\}$ 为 p_j^* 和 $p_j(X)$ 长度的最

大值.

由公式 (1) 可知, $Sim(p_j^*, p_j(X))$ 越大, X 遍历的子路径 $p_j(X)$ 越接近目标子路径 p_j^*. 当 $Sim(p_j^*, p_j(X)) = 1$ 时, $p_j(X)$ 即为 p_j^*. 此时, X 即为覆盖 p_j^* 的测试用例. 这样一来, 可转换进程 π_j 内目标子路径 p_j^* 的测试用例生成问题为最大优化问题, 并建立如公式 (2) 所示的优化模型.

$$OptMod_j = \max_{X \in D} F_j(X) = Sim(p_j^*, p_j(X)) \tag{2}$$

其中, X 为一个测试输入, 也称为一个进化个体; $F_j(X)$ 为适应值函数, 其具体的数值表示 X 覆盖 p_j^* 的适应值, 即覆盖程度. 结合多任务进化优化的工作原理, 这里能够将面向 p_j^* 的测试用例生成优化模型 $OptMod_j$ 置于第 j 个优化任务 $task_j$ 中, 并基于本文所提策略和进化算法求解 $OptMod_j$, 其中, $j = 1,2,\ldots,N$.

1.2 基于进化优化的路径覆盖测试

Ma 等人[11]提出了一个面向 C 语言程序的测试用例生成框架, 该框架基于禁忌搜索算法搜索隐藏故障的可行路径, 从而约减需生成的有效测试用例数量. 鉴于软件测试成本通常占软件开发总成本的 40% 以上, Khari 等人[12]分别使用爬山算法、粒子群优化、萤火虫算法, 以及布谷鸟搜索算法等 6 种进化优化方法, 生成覆盖路径的测试用例, 并通过对比实验数据, 找到求解路径覆盖测试问题的最优算法. 基于相似路径通常由相似测试用例来覆盖的原则, Cai 等人[13]设计了一款新式局部搜索策略, 并提出了一种基于改进分散搜索策略的进化优化算法, 用于生成覆盖路径的测试用例. 此后, Semujju 等人[14]提出了一种基于反馈导向机制的路径覆盖测试用例生成方法, 该方法的优势在于能够自动删除不可行路径, 以及筛选出可行路径.

针对 MPI 程序的不确定性特征, Tian 等人[15]通过变异不确定性原语的通信参数, 寻找合适的通信序列, 并使用遗传算法生成了覆盖路径的测试用例. 随后, Gong 等人[16]提出了一种用于 MPI 程序路径覆盖测试的反馈导向遗传算法, 该算法通过改进遗传算子提升了测试用例生成效率. 针对不确定性带来的 MPI 程序测试难度大的问题, 我们提出为每一路径选择一个优越的通信序列, 并基于协同进化遗传算法生成期望的测试用例, 实现以同一测试用例执行具有不确定性的 MPI 程序时, 该测试用例会遍历相同的程序片段[17]. 针对 MPI 程序测试效率低的问题, Du 等人[18]基于实际执行代码行数与条件谓词数等评价指标, 选择最容易覆盖的路径作为目标路径, 并使用遗传算法生成期望的测试用例. 随后, 我们提出了分布式代理模型驱动的 MPI 程序路径覆盖测试用例进化生成方法, 进一步减少了测试所需成本[19].

以上成果表明, 研究 MPI 程序路径覆盖测试用例生成问题及其进化求解方法, 是有效生成测试用例的重要途径. 然而, 基于现有工作生成覆盖 MPI 程序目标路径的测试用例生成时, 所需的测试成本仍居高不下. 鉴于此, 本文将 MPI 程序目标路径内每条目标子路径的测试用例生成问题优化模型置于一个优化任务中, 并在每一优化任务中, 基于进化算法来求解相应的优化模型, 以及利用代理模型来降低被测 MPI 程序的实际执行成本, 从而进一步提升 MPI 程序路径覆盖测试效率.

1.3 代理辅助进化优化

代理辅助进化优化的一般原理如下: 使用进化算法求解昂贵优化问题的过程中, 基于收集的历史数据训练一或多个代理模型, 用于估计进化个体的适应值, 从而减少进化个体适应值的评估次数和成本[20]. 当前, 用于进化个体适应值估计的代理模型多种多样, 主要分为单一代理模型和多代理模型两类[21].

针对单一代理模型, Li 等人[22]提出了一种径向基函数辅助进化优化框架, 降低了昂贵优化问题的求解成本. 为解决昂贵高维优化问题, Zhan 等人[23]提出了一种 Kriging 辅助差分进化算法, 该算法应用于若干测试问题上, 验证了 Kriging 模型的优越性. 为解决高约束昂贵优化问题, Jiao 等人[24]提出了一种高斯过程辅助进化算法, 该算法能够提高优化问题求解速度和约束处理能力. 关于多代理模型, Wang 等人[25]提出了一种数据驱动进化优化算法, 该算法采用集成代理模型代替昂贵优化问题的实际求解, 降低了问题的求解成本. 针对高维决策变量导致的代理模型预测准确度不高的问题, Lin 等人[26]提出了一种用于处理昂贵多目标优化问题的集成代理框架, 显著提高了模型的预测准确度. 使用进化算法求解 MPI 程序路径覆盖测试用例生成优化问题的过程中, 我们开发了集成代理

模型, 该模型用于代替 MPI 程序的实际执行, 从而降低了进化个体的实际评价成本[4]. 此外, 我们聚类一个样本集为若干簇, 并基于每一簇内样本训练一个代理模型, 以用于估计靠近簇中心进化个体的适应值, 实现减少 MPI 程序的实际执行次数[27].

本文在基于进化算法并行求解每一优化任务中优化模型的过程中, 使用代理模型代替 MPI 程序的实际执行, 从而进一步降低测试成本. 鉴于此, 上述研究工作能为本文代理辅助的训练和应用提供重要的理论和方法指导.

2 所提方法

2.1 总体框架

为提升测试用例生成效率, 本文提出一种代理辅助多任务进化优化引导的 MPI 程序路径覆盖测试用例生成方法. 所提方法中, 首先, 面向目标路径内每条目标子路径, 训练一个代理模型, 用于代替被测 MPI 程序的实际执行; 然后, 基于每条目标子路径对应的代理模型, 估计相应测试用例生成优化任务中进化个体的适应值, 以此选择最优进化个体和形成候选测试用例集; 最后, 基于候选测试用例集及其面向每条目标子路径的真实适应值, 更新对应每条目标子路径的代理模型, 并进化种群, 直至生成覆盖目标路径的测试用例. 所提方法的总体框架如算法 1.

算法 1. 所提方法总体框架.

输入: $\{P^*_{np}|np=1,2,\ldots,NP\}$;
输出: $TCSet$.

1. **FOR** $P^*_{np}=P^*_1\rightarrow P^*_{NP}$ **DO**
2. $SurrModelSet\leftarrow Surr\_Model\_Set\_Train(P^*_{np})$
3. 设置 P^*_{np} 的覆盖标志 $CovFlag=0$
4. **FOR** $task_j=task_1\rightarrow task_N$ && $CovFlag==0$ **DO**
5. 初始化种群 $EvoPop_j$
6. **FOR** $iter=1\rightarrow MaxGen$ && $CovFlag==0$ **DO**
7. $EstFit_j\leftarrow SurrModelSet(EvoPop_j)$
8. $BestInd_j\leftarrow Best\_Ind\_Select(EstFit_j)$
9. $\{CovFlag,TC,CTSet,RealFit_j\}\leftarrow CovFlag\_CTSet\_Etc(BestInd_j)$
10. $SurrMod_j\leftarrow Mod\_Upd(iter,CTSet,RealFit_j)$
11. 进化种群 $EvoPop_j$
12. **END FOR**
13. **END FOR**
14. 添加 TC 至 $TCSet$
15. **END FOR**
16. **RETURN** $TCSet$

该算法的输入为 MPI 程序的基本目标路径集, 即 $\{P^*_{np}|np=1,2,\ldots,NP\}$, 输出为覆盖该基本目标路径集的测试用例集合, 即 $TCSet$. 算法 1 首先以 P^*_{np} 为输入, 运行函数 $Surr\_Model\_Set\_Train(P^*_{np})$, 训练对应 P^*_{np} 内目标子路径的代理模型集合 $SurrModelSet$ (行 1 和行 2); 然后, 针对第 j 个优化任务 $task_j$, 基于 $SurrModelSet(EvoPop_j)$ 估计种群 $EvoPop_j$ 内进化个体的适应值 $EstFit_j$, 并运用函数 $Best\_Ind\_Select(EstFit_j)$ 选择最优进化个体 $BestInd_j$, 以及基于函数 $CovFlag\_CTSet\_Etc(BestInd_j)$ 得到候选测试用例集 $CTSet$ 及其面向目标子路径 p^*_j 的真实适应值 $RealFit_j$ 等 (行 3–9); 最后, 基于 $iter$、$CTSet$ 和 $RealFit_j$, 运行函数 $Mod\_Upd(iter,CTSet,RealFit_j)$, 以便更新对应目标子路径 p^*_j 的代理模型 $SurrModel_j$, 并进化种群, 直至生成覆盖每一目标路径的测试用例 (行 10–16). 关于

$Surr\_Model\_Set\_Train(P^*_{np})$, $SurrModelSet(EvoPop_j)$, $Best\_Ind\_Select(EstFit_j)$, $CovFlag\_CTSet\_Etc(BestInd_j)$, $Mod\_Upd(iter, CTSet, RealFit_j)$ 的具体实施细节, 将分别在第 2.2–2.4 节中给出.

2.2 代理模型训练

基于进化优化的 MPI 程序路径覆盖测试用例生成过程中, 进化个体适应值的评价需要重复执行被测 MPI 程序, 而被测程序的反复执行需要高昂的计算成本. 鉴于此, 训练合理的代理模型, 用于代替 MPI 程序的实际执行, 将能够显著降低 MPI 程序的执行成本.

训练代理模型之前, 需采样一定数量具有代表性的测试输入. 考虑到拉丁超立方采样已经成功应用于 MPI 程序测试输入采样中, 且证明采样的测试输入集能够满足均匀分布[17], 因此, 这里同样基于拉丁超立方采样, 获取具有代表性的测试输入集, 其实施细节如下: 首先, 平均划分测试输入 X 中每维变量 x_i ($i=1,2,\ldots,M$) 的取值范围为 $InSize$ 个区间, 记为 $\{R_i^k | k=1,2,\ldots,InSize\}$; 然后, 在变量 x_i 对应的每一区间 R_i^k ($k=1,2,\ldots,InSize$) 内, 采样一个变量值, 并把这 $InSize$ 个变量值作为一个集合, 记为 $InputSet_i$; 最后, 依次在每一 $InputSet_i$ ($i=1,2,\ldots,M$) 中, 不放回随机提取一个变量值, 用以组合成一个测试输入, 直至 $InputSet_i$ 为空. 至此, 把采样的 $InSize$ 个测试输入作为一个集合, 记为 $InputSet=\{x_i^k | i=1,2,\ldots,M; k=1,2,\ldots,InSize\}$, 如公式 (3) 所示.

$$InputSet=\begin{bmatrix} x_1^1 & x_2^1 & \cdots & x_M^1 \\ x_1^2 & x_2^2 & \cdots & x_M^2 \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{InSize} & x_2^{InSize} & \cdots & x_M^{InSize} \end{bmatrix} \tag{3}$$

采样具有代表性测试输入集合后, 需基于每一测试输入执行被测 MPI 程序, 并计算出该测试输入对目标路径 P^*_{np} 内每一目标子路径的覆盖程度, 具体操作细节如下: 首先, 依次提取公式 (3) 中每一行变量值 $InputSet^k$ ($k=1,2,\ldots,InSize$) $=\{x_i^k | i=1,2,\ldots,M\}$, 即一个测试输入; 然后, 基于 $InputSet^k$ 执行一个 MPI 程序, 以此获取该测试输入实际遍历该 MPI 程序内每一进程 π_j ($j=1,2,\ldots,N$) 的子路径, 表示为 p_j ($InputSet^k$); 最后, 基于公式 (1), 计算 p_j ($InputSet^k$) 与相应目标子路径 p_j^* 的相似度, 即 $InputSet^k$ 对于 p_j^* 的覆盖程度, 记为 Sim_j^k. 相应地, 基于所有测试输入执行 MPI 程序后, 将计算的面向目标路径 P^*_{np} 内每一目标子路径的覆盖程度矩阵表示为 $CovMat=\{Sim_j^k | j=1,2,\ldots,N; k=1,2,\ldots,InSize\}$, 如公式 (4) 所示.

$$CovMat=\begin{bmatrix} Sim_1^1 & Sim_2^1 & \cdots & Sim_N^1 \\ Sim_1^2 & Sim_2^2 & \cdots & Sim_N^2 \\ \vdots & \vdots & \ddots & \vdots \\ Sim_1^{InSize} & Sim_2^{InSize} & \cdots & Sim_N^{InSize} \end{bmatrix} \tag{4}$$

获取覆盖程度矩阵 $CovMat$ 后, 需基于测试输入集合 $InputSet$ 内每一行变量值与矩阵 $CovMat$ 内每一列覆盖程度, 形成对应目标路径 P^*_{np} 内每一目标子路径的训练样本集, 其具体实施步骤如下: 首先, 提取 $InputSet$ 内第 k 行变量值 $InputSet^k=\{x_i^k | i=1,2,\ldots,M\}$, 作为第 k 个训练样本的输入; 然后, 抽取 $CovMat$ 内第 k 行和第 j 列覆盖程度, 作为第 k 个训练样本面向第 j 条目标子路径 p_j^* 的输出, 即 Sim_j^k; 接着, 针对 P^*_{np} 内 p_j^*, 形成第 k 个训练样本, 记为 $Sample_j^k=(InputSet^k, Sim_j^k)$; 最后, 重复上述步骤, 直至形成对应 p_j^* 的 $InSize$ 个训练样本组成的集合, 记为 $SampleSet_j$ ($j=1,2,\ldots,N$) $=\{Sample_j^k | k=1,2,\ldots,InSize\}$.

基于形成的样本集, 训练代理模型之前, 需选定合适的代理模型类型. 考虑到克里金模型 (Kriging model) 已经广泛应用于估计进化个体的适应值[23,28], 因此, 这里采用克里金模型作为代理模型类型. 随后, 这里将基于形成的 $SampleSet_j$, 训练对应目标路径 P^*_{np} 内每一目标子路径 p_j^* 的代理模型, 记为 $SurrModel_j$. 合并所有代理模型, 得到对应 P^*_{np} 内所有目标子路径的代理模型集合, 记为 $SurrModelSet=\{SurrModel_j | j=1,2,\ldots,N\}$.

算法 2 给出了训练代理模型的伪代码, 即 $Surr\_Model\_Set\_Train(P^*_{np})$. 算法 2 的输入为第 np 条目标路径 P^*_{np}, 输出为对应 P^*_{np} 内所有目标子路径的代理模型集合 $SurrModelSet$. 算法 2 首先采样测试输入集合和计算覆盖程度

矩阵 (行 1 和行 2); 然后, 形成面向每条目标子路径的样本集, 并用于训练相应的代理模型 (行 3–6); 最后, 合并和返回所有的代理模型 (行 7 和行 8).

算法 2. $Surr\_Model\_Set\_Train(P^*_{np})$.

输入: P^*_{np};
输出: $SurrModelSet$.

1. 采样具有代表性的测试输入集合 $InputSet$
2. 基于 $InputSet$ 计算覆盖程度矩阵 $CovMat$
3. **FOR** $p^*_j = p^*_1 \to p^*_N$ **DO**
4. 形成对应 p^*_j 的训练样本集 $SampleSet_j$
5. 基于 $SampleSet_j$, 训练对应 p^*_j 的代理模型 $SurrModel_j$
6. **END FOR**
7. 合并 $SurrModel_j$ 至 $SurrModelSet$
8. **RETURN** $SurrModelSet$

2.3 候选测试用例形成

已知的是, 本文的目标是生成覆盖目标路径 P^*_{np} $(np = 1,2,\ldots,N)$ 的测试用例, 而面向 P^*_{np} 内每条目标子路径 p^*_j $(j = 1,2,\ldots,N)$ 的优化任务 $task_j$ 是为了生成覆盖 p^*_j 的测试用例. 显然, 覆盖 p^*_j 的测试用例并无法保证能够覆盖 P^*_{np}. 鉴于此, 本节需要从面向 P^*_{np} 内 p^*_j 的测试用例生成优化任务中选择最优进化个体, 并形成候选测试用例集, 用于覆盖目标路径 P^*_{np}.

在测试用例生成优化任务 $task_j$ 中选择最优进化个体之前, 需要基于第 2.2 节中训练好的代理模型来估计 $task_j$ 内种群中每一进化个体的适应值, 即运行函数 $SurrModelSet(EvoPop_j)$, 其函数实施细节如算法 3 所示: 一方面, 在 $task_j$ 中调用代理模型集合 $SurrModelSet$ 内第 j 个代理模型 $SurrModel_j$ (行 1); 另一方面, 以 $task_j$ 内种群 $EvoPop_j$ 中每一进化个体为输入, 运行代理模型 $SurrModel_j$, 估计出所有进化个体的适应值集合, 记为 $EstFit_j$ (行 2–6). 注意, 这里定义 $EvoPop_j = \{X^l_j | l = 1,2,\ldots,PopSize\}$, 其中, X^l_j 表示 $EvoPop_j$ 中第 l 个进化个体, $PopSize$ 表示种群内进化个体数. 相应地, 所有进化个体的适应值集合 $EstFit_j$ 也表示为 $EstFit_j = \{\bar{F}_j(X^l_j) | l = 1,2,\ldots,PopSize\}$, 其中, $\bar{F}_j(X^l_j)$ 表示 X^l_j 的估计适应值.

算法 3. $SurrModelSet(EvoPop_j)$.

输入: $SurrModelSet$, $EvoPop_j = \{X^l_j | l = 1,2,\ldots,PopSize\}$;
输出: $EstFit_j$.

1. 调用 $SurrModelSet$ 内第 j 个代理模型 $SurrModel_j$
2. **FOR** $X^l_j = X^1_j \to X^{PopSize}_j$ **DO**
3. 基于 $SurrModel_j$ 估计进化个体 X^l_j 的适应值 $\bar{F}_j(X^l_j)$
4. 将 X^l_j 的估计适应值 $\bar{F}_j(X^l_j)$ 保存至 $EstFit_j$
5. **END FOR**
6. **RETURN** $EstFit_j$

得到 $EstFit_j$ 后, 这里给出最优进化个体选择策略, 对应函数 $Best\_Ind\_Select(EstFit_j)$, 其函数实施步骤如算法 4: 一方面, 基于种群 $EvoPop_j$ 中每一进化个体对应的估计适应值, 提取最大估计适应值对应的索引号 (行 1–6); 另一方面, 基于提取的索引号, 选择相应的进化个体, 以作为最优进化个体, 记为 $BestInd_j = \{x^j_i | i = 1,2,\ldots,M\}$ (行 7 和行 8).

算法 4. $Best\_Ind\_Select(EstFit_j)$.

输入: $EstFit_j$;

输出: $BestInd_j$.

1. 设置 $MaxEstFit = 0$ 和 $EstIndex = Null$
2. **FOR** $\bar{F}_j(X_j^l) = \bar{F}_j(X_j^1) \to \bar{F}_j(X_j^{PopSize})$ **DO**
3. **IF** $MaxEstFit < \bar{F}_j(X_j^l)$ **THEN**
4. $MaxEstFit = \bar{F}_j(X_j^l)$, 且 $EstIndex = l$
5. **END IF**
6. **END FOR**
7. 基于 $EstIndex$, 选择最优进化个体 $BestInd_j$
8. **RETURN** $BestInd_j$

通过上述操作, 这里能够得到每一优化任务 $task_j$ 中当前种群内的最优进化个体 $BestInd_j$. 虽然, $BestInd_j$ 在覆盖 P_{np}^* 内目标子路径 p_j^* 时具有极大优势, 但是, 从覆盖整条 MPI 程序目标路径 P_{np}^* 的角度来讲, $BestInd_j$ 并不一定具备优势. 鉴于此, 这里将基于每一优化任务 $task_j$ 中的 $BestInd_j$, 形成一个候选测试用例集, 记为 $CTSet$, 用于覆盖 P_{np}^*.

算法 5. $CovFlag\_CTSet\_Etc(BestInd_j)$.

输入: $BestInd_j$;

输出: $CovFlag$, TC, $CTSet$, $RealFit_j$.

1. **FOR** $task_j = task_1 \to task_N$ **DO**
2. 添加 $BestInd_j$ 至 $CTSet$
3. **END FOR**
4. **FOR** $task_j = task_1 \to task_N$ **DO**
5. **FOR** $i = 1 \to M$ **DO**
6. 添加 $BestInd_j$ 内变量值 x_i^j 至 $CTSet_i$
7. **END FOR**
8. **END FOR**
9. 基于每一 $CTSet_i$, 形成候选测试用例集 $CTSet$
10. **FOR** $X^r = X^1 \to X^{2N}$ **DO**
11. 基于 X^r 执行被测 MPI 程序, 并计算 $F_j(X^r)$, 用于组成 $RealFit_j$
12. **IF** $\left(1/N\right)\sum_{j=1}^{N} F_j(X^r) == 1$ **THEN**
13. X^r 即为覆盖 P_{np}^* 的测试用例 TC, 且 $CovFlag = 1$
14. **END IF**
15. **END FOR**
16. **RETURN** $CovFlag$, TC, $CTSet$, $RealFit_j$

为此, 首先将每一优化任务 $task_j$ $(j = 1,2,\ldots,N)$ 中的 $BestInd_j$ 添加至候选测试用例集 $CTSet$ 中; 然后, 运用拉丁超立方采样原理, 将每一 $BestInd_j$ $(j = 1,2,\ldots,N)$ 内变量值 x_i^j 组合为一个集合, 记为 $CTSet_i = \{x_i^j | j = 1,2,\ldots,N\}$; 最后, 依次在每一 $CTSet_i$ $(i = 1,2,\ldots,M)$ 中, 不放回随机抽取一个变量值, 用于形成一个候选测试用例, 并添加至 $CTSet$, 直至 $CTSet_i$ 为空. 至此, 能够形成含有 $2N$ 个候选测试用例的集合, 即 $CTSet = \{X^r | r = 1,2,\ldots,2N\}$, 其中, X^r

表示第 r 个候选测试用例. 注意, 基于不放回随机抽取的变量值组成一个候选测试用例时, 要避免组成的候选测试用例与任一 $BestInd_j$ 重复.

可以看出, 上述形成的候选测试用例集不仅考虑了覆盖 P^*_{np} 的全局需求, 而且也考虑了覆盖 P^*_{np} 内每条目标子路径 p^*_j 的局部需求. 得到 $CTSet$ 后, 将基于每一候选测试用例 X^r $(r=1,2,\ldots,2N)$ 执行被测 MPI 程序, 并计算 X^r 面向 P^*_{np} 内每条目标子路径 p^*_j 的真实适应值, 记为 $F_j(X^r)$. 鉴于此, 可得到所有候选测试用例面向 p^*_j 的真实适应值集合, 记为 $RealFit_j=\{F_j(X^r)|r=1,2,\ldots,2N\}$. 此外, 当 X^r 同时覆盖 P^*_{np} 内所有目标子路径时, 即 $\left(1/N\right)\sum_{j=1}^{N}F_j(X^r)=1$, X^r 即为覆盖 P^*_{np} 的测试用例, 记为 TC.

算法 5 给出了候选测试用例集的形成策略, 即函数 $CovFlag\_CTSet\_Etc(BestInd_j)$. 算法 5 的输入为每一优化任务 $task_j$ 中最优进化个体 $BestInd_j$, 输出为覆盖 P^*_{np} 的标志 $CovFlag$、测试用例 TC、候选测试用例集 $CTSet$ 及其对应每条目标子路径的真实适应值 $RealFit_j$. 算法 5 中, 首先添加 $BestInd_j$ 至 $CTSet$ 中 (行 1–3); 然后, 形成候选测试用例集 (行 4–9); 接着, 基于每一候选测试用例, 执行被测 MPI 程序, 并计算对应 p^*_j 的 $RealFit_j$, 以及判定是否生成了测试用例 TC 等 (行 10–15); 最后, 返回 $CovFlag$、TC、$CTSet$, 以及 $RealFit_j$ (行 16).

2.4 代理模型更新

基于进化算法求解每一优化任务 $task_j$ 内优化模型 $OptMod_j$ 的过程中, 需要使用代理模型来估计 $task_j$ 内每一代种群内进化个体的适应值. 然而, 随着 $task_j$ 内 $OptMod_j$ 的进化求解, 相应代理模型 $SurrModel_j$ 很难持续保证进化个体适应值的高估计准确度. 鉴于此, 本节有必要利用面向 P^*_{np} 内每条目标子路径 p^*_j 的测试知识, 补充训练样本集, 用于更新代理模型.

已知, 通过第 2.3 节中算法 5 能够得到候选测试用例集 $CTSet$ 及其对应每条目标子路径 p^*_j 的真实适应值 $RealFit_j$, 其中, $CTSet=\{X^r|r=1,2,\ldots,2N\}$ 和 $RealFit_j=\{F_j(X^r)|r=1,2,\ldots,2N\}$. 此外, 第 2.2 节已经给出面向 p^*_j 的训练样本集, 即 $SampleSet_j=\{Sample^k_j|k=1,2,\ldots,SampSize\}$. 注意, $SampSize$ 的初始值为 $InSize$, 但随着 $task_j$ 内种群的不断进化, $SampSize$ 的值会变大. 可以看出, 这里将选择 $CTSet$ 和 $RealFit_j$ 内具有价值的训练样本, 用于补充 $SampleSet_j$.

考虑到面向 p^*_j 具有最高真实适应值的候选测试用例更接近覆盖 p^*_j 的期望测试用例, 因此, 将该候选测试用例补充至训练样本集, 并以此更新代理模型后, 所更新代理模型在期望测试用例附近区域的估计准确度将能够得到提升. 此外, 鉴于一个数组的平均值能够反映该数组的集中趋势, 因此, 这里选择某一距离 $SampleSet_j$ 内所有训练样本输入的平均值最远的候选测试用例, 并补充至训练样本集, 该训练样本集将具有更优异的均匀分布特性, 以此更新的代理模型将具备更好的异常值估计准确度.

基于以上分析, 这里将给出对应 p^*_j 的训练样本集 $SampleSet_j$ 补充方法, 其实现步骤如下所示: 一方面, 选择面向 p^*_j 具有最高真实适应值的候选测试用例, 并将该候选测试用例及其真实适应值分别作为训练样本的输入与输出, 补充至 $SampleSet_j$; 另一方面, 计算 $SampleSet_j$ 内所有训练样本输入的平均值, 基于此, 选择距离该平均值最远的候选测试用例, 也将该候选测试用例及其真实适应值分别作为训练样本的输入与输出, 补充至 $SampleSet_j$.

补充面向 p^*_j 的训练样本集 $SampleSet_j$ 后, 并不能直接用于更新代理模型 $SurrModel_j$, 原因在于, 反复地更新代理模型也需要一定的计算成本. 考虑到现有工作已经表明基于进化代数阈值来更新代理模型, 能够显著降低代理模型的更新成本[29], 因此, 本节也设定一个进化代数阈值, 记为 $UpdGen$, 并当每一 $task_j$ 内种群进化 $UpdGen$ 代数后, 基于面向 p^*_j 的最新训练样本集 $SampleSet_j$, 更新代理模型 $SurrModel_j$.

算法 6 给出了代理模型更新的伪代码, 即 $Mod\_Upd(iter,CTSet,RealFit_j)$. 该算法的输入为当前进化的代数 $iter$, 候选测试用例集 $CTSet$ 及其对应每条目标子路径 p^*_j 的真实适应值 $RealFit_j$, 输出为更新后的代理模型 $SurrModel_j$. 算法 6 中, 一方面, 基于 $CTSet$ 和 $RealFit_j$ 补充面向的 p^*_j 训练样本集 $SampleSet_j$ (行 1–5); 另一方面, 当种群进化到阈值时, 基于最新的 $SampleSet_j$, 更新和返回代理模型 (行 6–9).

算法 6. $Mod\_Upd(CTSet, RealFit_j)$.

输入: $CTSet$, $RealFit_j$;

输出: $SurrModel_j$.

1. 选择 $CTSet$ 内具有最高真实适应值的候选测试用例
2. 将选择的候选测试用例及其真实适应值补充至 $SampleSet_j$
3. 计算 $SampleSet_j$ 所有训练样本输入的平均值
4. 选择距离该平均值最远的候选测试用例
5. 将选择的候选测试用例及其真实适应值补充至 $SampleSet_j$
6. **IF** $iter\%UpdGen == 0$ **THEN**
7. 基于补充的 $SampleSet_j$, 更新代理模型 $SurrModel_j$
8. **END IF**
9. **RETURN** $SurrModel_j$

3 实 验

本节将所提方法应用于若干基准 MPI 程序路径覆盖测试中, 并通过与其他多种方法进行比较, 证明所提方法能否提高测试用例生成的有效性与效率. 首先, 提出研究问题; 然后, 介绍被测程序和实验设置; 最后, 给出并分析实验结果.

3.1 研究问题

本文一方面单独建立 MPI 程序目标路径内每条目标子路径的测试用例生成优化模型, 并基于多任务进化优化来求解每一优化模型, 以期生成覆盖 MPI 程序目标路径的测试用例, 也就是说, 将每条目标子路径的测试用例生成优化模型置于一个优化任务中, 并基于进化算法并行求解每一优化任务中优化模型的最优解, 用于覆盖 MPI 程序目标路径. 因此, 有必要验证基于多任务进化优化能否高效生成 MPI 程序目标路径的测试用例. 另一方面, 针对目标路径内每条目标子路径, 这里训练一个代理模型, 用于估计相应测试用例生成优化任务中进化个体的适应值, 进而代替 MPI 程序的实际执行. 鉴于此, 有必要验证基于代理模型能否降低 MPI 程序路径覆盖测试成本. 此外, 还需将所提方法与其他现有方法对比, 以进一步验证所提方法能否提高测试用例生成的有效性与效率.

RQ1. 基于多任务进化优化能否高效生成 MPI 程序目标路径的测试用例?

为了验证 RQ1, 我们对比所提方法和一个被称为 Approach-SingTask 的方法. 这里, Approach-SingTask 不在使用多任务进化优化框架, 即不在单独建立 MPI 程序目标路径内每条目标子路径的测试用例生成优化模型, 以及没有基于进化算法并行求解每一优化任务中优化模型的环节, 而是建立 MPI 程序目标路径的测试用例生成优化模型, 如公式 (5) 所示, 并基于进化算法求解公式 (5). 此外, Approach-SingTask 中仅需训练一个面向 MPI 程序目标路径的代理模型, 该模型训练样本的输入与输出分别为测试输入和 $Func(X)$. 除了上述提到的不同之处外, Approach-SingTask 中其他细节与本文提出方法完全相同. 此后, 分别使用所提方法和 Approach-SingTask 生成期望的测试用例, 如果使用所提方法生成测试用例的效率和有效性更高, 那么, 表示基于多任务进化优化能够高效生成 MPI 程序目标路径的测试用例.

$$OptMod = \max_{X \in D} Func(X) = \frac{1}{N}\sum_{j=1}^{N} F_j(X) \tag{5}$$

其中, X 为一个测试输入, 也称为一个进化个体; $Func(X)$ 表示 X 覆盖 MPI 程序目标路径 P^* 的覆盖程度. 可以看出, 当 $Func(X) = 1$ 时, X 即为覆盖 P^* 的测试用例.

RQ2. 基于代理模型能否降低 MPI 程序路径覆盖测试成本?

为了验证 RQ2, 我们对比所提方法和一个被称为 Approach-NoMod 的方法. 这里, Approach-NoMod 不需要训

练和更新代理模型, 而是通过被测 MPI 程序的实际执行来计算每一测试用例生成任务 $task_j$ 中种群内进化个体的适应值. 除了上面提到的无需训练和更新代理模型, 以及基于实际执行 MPI 程序计算进化个体适应值外, Approach-NoMod 的其他细节与本文提出方法完全相同. 此后, 分别使用所提方法和 Approach-NoMod 生成期望的测试用例, 如果使用所提方法生成测试用例的效率和有效性更高, 那么, 表示基于代理模型能够降低 MPI 程序路径覆盖测试成本.

RQ3. 所提方法能否提高测试用例生成的有效性与效率?

为了验证 RQ3, 我们对比所提方法和文献 [30] 中已有方法, 记为 Approach-EvoGen. 需注意的是, Approach-EvoGen 同样研究了 MPI 程序路径覆盖测试用例生成问题, 且为研究该问题的最新成果. 为更好地理解 Approach-EvoGen, 其工作原理给定如下: 首先设计了通信序列影响候选测试用例覆盖性能的评价指标, 并用于划分所有目标路径为若干组; 然后, 构建了每组目标路径对应的测试用例集生成优化模型; 最后, 基于多任务进化优化算法并行求解每一模型, 用于生成覆盖每组目标路径的测试用例集. 可以看出, Approach-EvoGen 仅从全局方面分析某一候选测试用例是否覆盖了目标路径, 而所提方法在每一优化任务中从局部方面寻找覆盖目标路径内每条目标子路径的最优进化个体, 并基于最优进化个体, 形成候选测试用例集, 进而从全局方面验证某一候选测试用例是否覆盖了目标路径, 最终形成覆盖所有目标路径的测试用例集. 显然, 所提方法恰好符合局部寻优加速全局最优的原理. 鉴于此, 这里分别使用所提方法和 Approach-EvoGen 生成期望的测试用例, 如果使用所提方法生成测试用例的效率和有效性更高, 那么, 证明所提方法能够提高测试用例生成的有效性与效率.

3.2 被测 MPI 程序

本节选择 7 个基准 MPI 程序作为被测程序, 这些程序的基本信息如表 1 所示. 表 1 中, IMB-MPI1 是 IMB 公司开发的用于进行 MPI 基准测试的组件, 用于检查 MPI-1 版本原语的性能[31]; HPL 是用于测试并行集群 Linpack 性能的基准 MPI 程序[32]; SUSY-HMC 是一个用于研究四维超对称杨-米尔斯理论的物理模拟 MPI 程序[33]; DepSolver、Kfray 和 ClustalW 均引用于文献 [34] 中的 MPI 程序, 其中, DepSolver 是一个并行多媒体 3D 静电解算器, Kfray 是一个用于创建逼真图像的光线追踪程序, ClustalW 是多基因序列比对的常用工具; mpiBLAST 为基因匹配 MPI 程序, 用于并行匹配所查询序列 (DNA 或氨基酸) 和数据库内序列之间相似的基因组序列片段[35].

表 1 被测 MPI 程序基本信息

被测程序	进程数	通信原语数	代码行数	目标路径数
IMB-MPI1	4	42	7092	1115
DepSolver	6	63	8988	398
Kfray	8	116	12728	665
HPL	3	70	15699	3469
SUSY-HMC	16	132	19201	2031
ClustalW	24	178	23265	2537
mpiBLAST	17	397	582179	12792

基于表 1 中被测 MPI 程序基本信息, 可以发现所选定的程序在进程数量、通信原语数、代码行数等方面分布较为广泛. 因此, 所选定的被测程序具有很好的代表性. 针对这些代表性程序, 获取的基本目标路径集也具有一定的典型性. 鉴于此, 如果所提方法能够高效求解这些代表性程序的覆盖测试问题, 那么相应的实验结果将更容易推广到其他 MPI 程序, 即具有良好的可扩展性.

3.3 实验设置

实验中使用的集群系统含有 24 个 H3C NaviData 5200 计算节点, 整个系统的计算资源超过 500 物理核, 可提供超过 16 Tflops 的单精度浮点计算整体理论峰值, 内存容量超过 4.5 TB, 物理可用空间超过 300 TB. 此外, 实验使用 MPI+C/C++编程语言和 Shark 机器学习库[36].

注意, 本文将采用文献 [4] 中提出的路径选择方法来得到表 1 中每一被测 MPI 程序的基本目标路径集, 记基

本目标路径集为 $B=\{P^*_{np}|np=1,2,\ldots,NumPath\}$, 其中, $NumPath$ 表示基本目标路径数. 关于表 1 中的每一被测 MPI 程序, 得到的基本目标路径集包括相应 MPI 程序中的所有分支和语句. 鉴于此, 当将所提方法应用于测试每一被测 MPI 程序的基本目标路径集时, 得到的实验结果将会令人信服和具有可扩展性.

为更好地理解拉丁超立方采样, 这里给出使用该采样方法的原因和工作原理, 其中, 原因为拉丁超立方采样能够高效地生成样本, 这些样本均匀分布于多维决策空间的超立方体中, 且无需获知决策空间确切的概率分布. 工作原理为: (1) 假设决策空间有 A 个维度, 则将每个维度划分为 B 个子区间; (2) 在每维度决策空间的每一子区间内随机采样一个值, 用于组成一个子区间采样集合; (3) 在每一子区间采样集合中不放回抽取一个值, 用于组成一个样本, 直至每一子区间采样集合变为空集.

考虑到文献 [17] 使用拉丁超立方体采样获得了 50 个测试输入, 并证明这些数量的测试输入能够满足 MPI 程序输入域的均匀分布特性. 鉴于此, 本文设置 $InSize$=50. 此外, 为设置代理模型更新阈值 $UpdGen$, 这里将沿用文献 [29] 中所设定的代理模型更新阈值, 即设定 $UpdGen=20$, 也就是说, 当每一优化任务中种群进化 20 代数后, 将更新相应的代理模型.

在每一测试用例生成优化任务中, 需要基于进化算法来求解相应的优化模型. 考虑到差分进化算法 (differential evolution, DE) 具有结构简单、容易实现、收敛快速、鲁棒性强等优点, 且在国际演化计算竞赛中, 差分进化算法被证明是速度最快的进化算法[37], 因此, 本文采用 DE 来求解每条目标子路径对应的优化模型. 针对差分进化算法 DE 的参数设置, 这里沿用文献 [38] 中设置的相关参数, 主要包括: 种群规模为 100, 最大进化代数 $MaxGen$ 等于 1 500, 缩放因子和交叉概率分别为 0.5 和 0.9.

为了验证 RQ1 到 RQ2, 本文使用了以下两个指标: 路径覆盖率和时间消耗. 路径覆盖率反映了有效性, 是指当使用所提方法或对比方法生成覆盖基本目标路径集合的测试用例时, 生成的测试用例所覆盖的目标路径数与目标路径总数的比率. 时间消耗反映了效率, 效率是指在应用所提方法或对比方法生成覆盖基本目标路径集合的测试用例时, 花费的时间或达到最大进化代数的时间. 显然, 路径覆盖率越高, 方法的时间消耗越低, 该方法就越有利. 为了减少随机因素对所提出方法或对比方法性能的影响, 每个方法对每一被测 MPI 程序的基本目标路径集独立运行 20 次. 注意, 每次运行所提方法或对比方法的任务是生成期望的测试用例集, 该用例集能够覆盖每一被测 MPI 程序的基本目标路径集.

当分别使用所提方法与对比方法生成期望的测试用例时, 会得到两组指标值. 考虑到 Welch's t-test 对于两组指标值数目不等、两组指标值方差不同等情况下的均值检验都很稳健, 而对于两组指标值方差相等的情况下也可以与 Student's t-test 输出一样的结果, 因此, 我们采用 Welch's t-test 来检验解析这两组指标的均值是否有显著差别, 并取显著性水平为 0.05. 如果 P 的值小于 0.05, 那么, 所提方法与对比方法具有明显差别.

3.4 实验结果与分析

(1) 验证 RQ1

为了验证 RQ1, 我们分别运行本文所提方法和 Approach-SingTask, 以生成覆盖每一被测程序的基本目标路径集的期望测试用例, 并计算 20 次运行的平均路径覆盖率和平均时间消耗. 表 2 列出了这两种测试用例生成方法的实验结果. 该表中, 第 1 和第 2 行是平均路径覆盖率; 第 3 行是两种方法的平均路径覆盖率百分比差异, 由 u^*-u 计算, 其中 u^* 和 u 分别是所提方法和 Approach-SingTask 的平均路径覆盖率; 第 4 和第 5 行是这两种方法的平均时间消耗; 第 6 行是平均时间消耗的约减率, 其由 $(v-v^*)/v\times100\%$ 计算, 其中 v^* 和 v 分别是所提方法和 Approach-SingTask 的平均时间消耗.

由表 2 的第 1–3 行可知: (1) 对于 7 个被测 MPI 程序, 所提方法和 Approach-SingTask 的平均路径覆盖率的平均值分别为 87.9% 和 72.2%; (2) 对于所有被测程序, 提出方法和 Approach-SingTask 的平均路径覆盖率的平均百分比差为 15.7%. 以上数据表明, 基于多任务进化优化的测试用例生成可以显著提高路径覆盖率.

从表 2 的第 4–6 行可以看出: (1) 对于 7 个被测 MPI 程序, 所提方法和 Approach-SingTask 的平均时间消耗的平均值分别为 81 165.8 min 和 130 994.6 min; (2) 面向所有被测 MPI 程序, 使用所提方法和 Approach-SingTask, 平

均时间消耗的约减率为 33.9%. 由此可以得出结论, 利用多任务进化优化的测试用例生成可以大大减少生成测试用例所需的时间消耗.

表 2 所提方法和 Approach-SingTask 的实验结果对比

指标		IMB-MPI1	DepSolver	Kfray	HPL	SUSY-HMC	ClustalW	mpiBLAST	平均值
平均覆盖率 (%)	所提方法	90.7	95.8	92.7	84.5	89.7	86.4	75.3	87.9
	Approach-SingTask	76.1	85.0	81.4	66.3	73.9	68.5	54.1	72.2
	百分比差异	14.6	10.8	11.3	18.2	15.8	17.9	21.2	15.7
平均时间消耗	所提方法 (min)	6093.5	2739.7	7596.4	51494.1	97686.3	142756.8	259793.7	81165.8
	Approach-SingTask (min)	8869.7	3853.3	10961.6	81997.0	144935.2	219964.3	446380.9	130994.6
	约减率 (%)	31.3	28.9	30.7	37.2	32.6	35.1	41.8	33.9
*P*值	路径覆盖率	0.025	0.061	0.039	<0.001	0.012	<0.001	<0.001	—
	时间消耗	0.028	0.062	0.041	<0.001	0.013	<0.001	<0.001	—

此外, 对于每一被测 MPI 程序记录的 20 次运行的路径覆盖率和时间消耗, 我们基于 Welch's t-test 计算相应的 *P* 值, 见于表 2 中第 7 和第 8 行.

根据表 2 中第 7 和第 8 行, 我们能够获得如下推论: (1) 针对 DepSolver 的路径覆盖测试, 运用所提方法和 Approach-SingTask 的路径覆盖率和时间消耗没有显著差异; (2) 对于其他测试程序, 使用提出方法和 Approach-SingTask 的路径覆盖率和时间消耗具有显著差异.

结合表 2 中第 1–6 行的数据, 我们可以得出结论使用所提方法生成测试用例的有效性和效率明显优于 Approach-SingTask. 因此, 验证了基于多任务进化优化能够高效生成 MPI 程序目标路径的测试用例.

(2) 验证 RQ2

为了验证 RQ2, 我们分别使用所提方法和 Approach-NoMod 来生成测试用例, 用于覆盖每一被测 MPI 程序的基本目标路径集. 表 3 列出了相关的实验结果, 表 3 中各行数据的含义可参考表 2.

表 3 所提方法和 Approach-NoMod 的实验结果对比

指标		IMB-MPI1	DepSolver	Kfray	HPL	SUSY-HMC	ClustalW	mpiBLAST	平均值
平均路径覆盖率 (%)	所提方法	90.7	95.8	92.7	84.5	89.7	86.4	75.3	87.9
	Approach-NoMod	83.5	91.2	86.8	72.9	80.4	75.6	60.9	78.8
	百分比差异	7.2	4.6	5.9	11.6	9.3	10.8	14.4	9.1
平均时间消耗	所提方法 (min)	6093.5	2739.7	7596.4	51494.1	97686.3	142756.8	259793.7	81165.8
	Approach-NoMod (min)	9331.5	3982.1	11474.9	85255.1	154322.7	231748.1	463090.4	137029.3
	约减率 (%)	34.7	31.2	33.8	39.6	36.7	38.4	43.9	36.9
*P*值	路径覆盖率	0.073	0.089	0.081	0.034	0.066	0.057	0.023	—
	时间消耗	<0.001	0.022	0.005	<0.001	<0.001	<0.001	<0.001	—

由表 3 的第 1–3 行可知: (1) 对于所有被测 MPI 程序, 所提方法和 Approach-NoMod 的平均路径覆盖率的平均值分别为 87.9% 和 78.8%; (2) 对于所有被测程序, 提出方法和 Approach-NoMod 的平均路径覆盖率的平均百分比差为 9.1%. 以上数据表明, 本文所训练的代理模型在提高路径覆盖率方面具有优势.

从表 3 的第 4–6 行可以看出: (1) 对于 7 个被测 MPI 程序, 所提方法和 Approach-NoMod 的平均时间消耗的平均值分别为 81165.8 min 和 137029.3 min; (2) 面向所有被测 MPI 程序, 使用所提方法和 Approach-NoMod, 平均时间消耗的约减率为 36.9%. 需明确的是, Approach-NoMod 没有使用代理模型来代替 MPI 程序的实际执行, 也就是说 Approach-NoMod 的运行过程中, 单纯依靠重复执行每一被测 MPI 程序, 计算每一测试用例生成任务中种群内进化个体适应值, 这需要花费的平均时间消耗为 3982.1–463090.4 min, 它们的平均值为 137029.3 min. 基于此, 所提方法在测试用例生成效率方法提升了 31.2%–43.9%, 它们的平均值为 36.9%. 相应地, 可证明本文所训练的代理模型可以大幅减少被测 MPI 程序重复执行所需的计算成本.

根据表 3 中第 7 和第 8 行, 我们能够获得如下推论: (1) 针对 IMB-MPI1、DepSolver、Kfray、SUSY-HMC 和 ClustalW 的路径覆盖测试, 使用提出方法和 Approach-NoMod 的路径覆盖率没有显著差异; (2) 面向 HPL 和 mpiBLAST 的路径覆盖测试, 使用提出方法和 Approach-NoMod 的路径覆盖率具有显著差异; (3) 针对所有被测 MPI 程序的路径覆盖测试, 使用提出方法和 Approach-NoMod 的时间消耗具有显著差异.

结合表 3 中第 1–6 行的数据, 我们可以得出结论使用所提方法生成测试用例的有效性和效率明显优于 Approach-NoMod. 因此, 验证了本文中基于代理模型能够降低 MPI 程序路径覆盖测试成本.

(3) 验证 RQ3

为了验证 RQ3, 我们分别使用所提方法和 Approach-EvoGen 来生成测试用例, 用于覆盖每一被测 MPI 程序的基本目标路径集. 表 4 列出了相关的实验结果, 表 4 中各行数据的含义可参考表 2.

表 4　所提方法和 Approach-EvoGen 的实验结果对比

指标		IMB-MPI1	DepSolver	Kfray	HPL	SUSY-HMC	ClustalW	mpiBLAST	平均值
平均路径覆盖率 (%)	所提方法	90.7	95.8	92.7	84.5	89.7	86.4	75.3	87.9
	Approach-EvoGen	78.2	87.5	82.9	67.0	75.5	71.4	55.6	74.0
	百分比差异	12.5	8.3	9.8	17.5	14.2	15.0	19.7	13.9
平均时间消耗	所提方法 (min)	6 093.5	2 739.7	7 596.4	51 494.1	97 686.3	142 756.8	259 793.7	81 165.8
	Approach-EvoGen (min)	8 921.7	3 672.5	10 521.3	81 221.0	143 445.4	215 644.7	427 292.3	127 245.6
	约减率 (%)	31.7	25.4	27.8	36.6	31.9	33.8	39.2	32.3
*P*值	路径覆盖率	0.030	0.071	0.065	<0.001	0.026	0.005	<0.001	—
	时间消耗	0.032	0.074	0.067	<0.001	0.028	0.007	<0.001	—

由表 4 的第 1–3 行可知: (1) 对于所有被测 MPI 程序, 所提方法和 Approach-EvoGen 的平均路径覆盖率的平均值分别为 87.9% 和 74.0%; (2) 对于所有被测程序, 提出方法和 Approach-EvoGen 的平均路径覆盖率的平均百分比差为 13.9%. 以上数据表明, 所提方法能够显著提高路径覆盖率.

从表 4 的第 4–6 行可以看出: (1) 对于 7 个被测 MPI 程序, 所提方法和 Approach-EvoGen 的平均时间消耗的平均值分别为 81 165.8 min 和 127 245.6 min; (2) 面向所有被测 MPI 程序, 使用所提方法和 Approach-EvoGen, 平均时间消耗的约减率为 32.3%. 由此可以得出结论, 所提方法可以显著减少生成测试用例所需的时间消耗.

根据表 4 中第 7 和第 8 行, 我们能够获得如下推论: (1) 针对 DepSolver 和 Kfray 的路径覆盖测试, 运用所提方法和 Approach-EvoGen 的路径覆盖率和时间消耗没有显著差异; (2) 对于其他测试程序, 使用提出方法和 Approach-EvoGen 的路径覆盖率和时间消耗具有显著差异.

结合表 4 中第 1–6 行的数据, 我们可以得出结论: 使用所提方法生成测试用例的有效性和效率明显优于 Approach-EvoGen. 因此, 验证了所提方法能够高效生成覆盖 MPI 程序目标路径的测试用例.

(4) 实验总结

为了确定基于多任务进化优化能否高效生成 MPI 程序目标路径的测试用例, 本文设计了研究问题 RQ1, 该研究问题的验证需要分别利用多任务和单任务进化优化生成覆盖 MPI 程序目标路径的测试用例. 通过验证 RQ1, 可以发现基于多任务进化优化的测试用例生成耗时更少, 且路径覆盖率更高, 这表明本文提出的多任务进化优化框架更有利于高效生成期望的测试用例.

为了判断基于代理模型能否降低 MPI 程序路径覆盖测试成本, 本文设计了研究问题 RQ2, 该研究问题的验证需要使用代理模型来降低被测 MPI 程序的实际执行成本, 以及不采用代理模型. 通过验证 RQ2, 能够发现使用代理模型的所提方法更快地生成了期望的测试用例, 这表明本文训练的代理模型在降低 MPI 程序路径覆盖测试成本方面具有显著优势.

为了判定所提方法能否提高测试用例生成的有效性与效率, 本文设计了研究问题 RQ3, 该研究问题的验证需要分别对比本文所提方法和当前应用于 MPI 程序路径覆盖测试的最新研究工作. 通过验证 RQ3, 可以发现本文所提方法在测试用例生成速度和覆盖能力方面具有优越性. 鉴于此, 可证明本文所提方法有助于推进 MPI 程序基本

路径覆盖测试的发展.

此外, 基于表 2–表 4 中的平均时间消耗和相应的约减率可知, 被测 MPI 程序规模越大, 约减率越大. 因此, 所提方法对于大型 MPI 程序具有更有利的效率优势. 除效率之外, 相较于对比方法, 基于所提方法的测试用例生成也拥有更高的平均路径覆盖率. 由此可见, 所提方法也适用于更多和更大的 MPI 程序, 从而支持所提方法的可扩展性.

4 对实验有效性的威胁

针对拉丁超立方体采样的采样尺寸 *InSize*, 如果 *InSize* 的值过大, 那么将需要大量执行被测 MPI 程序, 这将需要高昂的计算成本; 反之, 采样的测试输入不能够满足 MPI 程序输入域的均匀分布特性, 进而导致不利于精准分组目标子路径. 为了减轻该威胁, 我们继续引用文献 [17] 中设置的 *SamSize* 值.

差分进化算法 DE 的参数设置也会影响实验结果. 鉴于此, DE 的参数引用文献 [38]. 从实验结果可以看出, 上述参数的设值是合理的. 需注意的是, 使用除 DE 之外的其他类型进化优化算法, 如爬山算法、粒子群优化、萤火虫算法, 以及布谷鸟搜索算法等, 会面临不同的实验结果. 然而, 本文的创新之处并不在于使用何种进化优化算法, 而是在于通过使用某种进化优化算法验证所提测试用例生成框架的优越性.

所选的被测 MPI 程序影响实验结论. 如果所选程序不具有代表性, 那么, 实验结论就很难直接推广到其他程序中. 为了尽可能减轻这类威胁, 我们在实验中选择具有不同进程数、原语数, 以及代码行数的程序, 表明它们具有良好的代表性.

如果目标路径存在不可达或逻辑错误等缺陷, 那么基本路径覆盖测试将面临不可行测试问题. 为避免该问题, 本文运用已有的路径形成方法[4], 为每一被测 MPI 程序生成了一个基本目标路径集. 实验结果表明, 本文的 MPI 程序基本路径覆盖测试很好地规避了不可行测试问题.

此外, 时间消耗是反映测试用例生成效率的重要指标. 实验中, 以分钟的形式描述时间消耗, 该值与实现环境的配置, 如处理器数量与类型等密切相关. 显然, 在完成相同任务时, 具有不同配置的环境将具有不同的时间消耗. 为了缓解该威胁, 在相同环境中, 运行 20 次相同的任务, 并记录时间消耗, 基于此计算平均时间消耗.

5 总 结

为高效生成测试用例, 本文提出一种代理辅助多任务进化优化引导的 MPI 程序路径覆盖测试用例生成方法. 所提方法中, 首先面向目标路径内每条目标子路径, 训练相应的代理模型; 然后, 将每条目标子路径的测试用例生成优化模型置于一个优化任务中, 基于相应的代理模型, 估计进化个体的适应值, 并选择最优进化个体, 用于形成候选测试用例集; 最后, 在每条目标子路径对应的优化任务中, 基于候选测试用例集及其面向每条目标子路径的真实适应值, 更新相应的代理模型, 直至生成覆盖 MPI 程序目标路径的测试用例. 将所提方法应用于若干基准 MPI 程序, 并与其他多种方法比较. 实验结果表明, 所提方法能够显著提高测试用例生成的有效性和效率.

考虑到本文研究的问题是 MPI 程序路径覆盖测试用例生成问题, 且运行所提方法过程中, 需要基于 MPI 并行编程标准来实现多个任务中优化模型的并行进化求解, 因此, 将所提方法应用至如 OpenMP 等其他多线程并发程序的路径覆盖测试时, 仅需对并行进化求解的过程做出相应的修改, 这表明所提方法拥有良好的普适性. 此外, 尽管实验中提出方法在测试用例生成方面的性能优于对比方法, 但是仍有进一步提高有效性与效率的空间. 未来, 我们将更深入地研究目标子路径的测试用例生成方法, 建立更优异的优化模型. 同样地, 我们也将开发新型技术, 用于加速优化模型的求解效率. 考虑到符号执行具有快速精准解析约束公式的优点, 因此, 我们也将进一步研究可以用于 MPI 程序路径覆盖测试用例生成的符号工具, 该工具将能够处理 MPI 程序内广播 (broadcast)、分散 (scatter)、收集 (gather)、全互换 (alltoall) 等特殊通信操作, 从而支持一条 MPI 程序目标路径涉及进程之间的消息传递解析. 通过以上未来的工作, 我们相信生成测试用例的有效性与效率会拥有更显著的优势.

References:

[1] Ezhilarasi GA, Swarup KS. Parallel contingency analysis in a high performance computing environment. In: Proc. of the 2009 Int'l Conf.

on Power Systems. Kharagpur: IEEE, 2009. 1–6. [doi: 10.1109/ICPWS.2009.5442650]

[2] Alvioli M, Baum RL. Parallelization of the TRIGRS model for rainfall-induced landslides using the message passing interface. Environmental Modelling & Software, 2016, 81: 122–135. [doi: 10.1016/j.envsoft.2016.04.002]

[3] Yan J, Zhang J. An efficient method to generate feasible paths for basis path testing. Information Processing Letters, 2008, 107(3–4): 87–92. [doi: 10.1016/j.ipl.2008.01.007]

[4] Sun BC, Gong DW, Tian T, Yao XJ. Integrating an ensemble surrogate model's estimation into test data generation. IEEE Trans. on Software Engineering, 2022, 48(4): 1336–1350. [doi: 10.1109/TSE.2020.3019406]

[5] Qiao KJ, Yu KJ, Qu BY, Liang J, Song H, Yue CT. An evolutionary multitasking optimization framework for constrained multiobjective optimization problems. IEEE Trans. on Evolutionary Computation, 2022, 26(2): 263–277. [doi: 10.1109/TEVC.2022.3145582]

[6] Qiao KJ, Liang J, Yu KJ, Wang MH, Qu BY, Yue CT, Guo YN. A self-adaptive evolutionary multi-task based constrained multi-objective evolutionary algorithm. IEEE Trans. on Emerging Topics in Computational Intelligence, 2023, 7(4): 1098–1112. [doi: 10.1109/TETCI.2023.3236633]

[7] Wang C, Wu K, Liu J. Evolutionary multitasking AUC optimization [Research Frontier]. IEEE Computational Intelligence Magazine, 2022, 17(2): 67–82. [doi: 10.1109/MCI.2022.3155325]

[8] Wang Y, Lin JQ, Liu J, Sun GY, Pang T. Surrogate-assisted differential evolution with region division for expensive optimization problems with discontinuous responses. IEEE Trans. on Evolutionary Computation, 2022, 26(4): 780–792. [doi: 10.1109/TEVC.2021.3117990]

[9] Si LC, Zhang XY, Tian Y, Yang SS, Zhang LM, Jin YC. Linear subspace surrogate modeling for large-scale expensive single/multi-objective optimization. IEEE Trans. on Evolutionary Computation, 2023. [doi: 10.1109/TEVC.2023.3319640]

[10] Jin YC, Wang HD, Chugh T, Guo D, Miettinen K. Data-driven evolutionary optimization: An overview and case studies. IEEE Trans. on Evolutionary Computation, 2019, 23(3): 442–458. [doi: 10.1109/TEVC.2018.2869001]

[11] Ma EZ, Fu XF, Wang X. Scalable path search for automated test case generation. Electronics, 2022, 11(5): 727. [doi: 10.3390/electronics11050727]

[12] Khari M, Sinha A, Verdú E, Crespo RG. Performance analysis of six meta-heuristic algorithms over automated test suite generation for path coverage-based optimization. Soft Computing, 2020, 24(12): 9143–9160. [doi: 10.1007/s00500-019-04444-y]

[13] Cai GC, Su QH, Hu ZB. Automated test case generation for path coverage by using grey prediction evolution algorithm with improved scatter search strategy. Engineering Applications of Artificial Intelligence, 2021, 106: 104454. [doi: 10.1016/j.engappai.2021.104454]

[14] Semujju SD, Huang H, Liu FQ, Xiang Y, Hao ZF. Search-based software test data generation for path coverage based on a feedback-directed mechanism. Complex System Modeling and Simulation, 2023, 3(1): 12–31. [doi: 10.23919/CSMS.2022.0027]

[15] Tian T, Gong DW, Kuo FC, Liu H. Genetic algorithm based test data generation for MPI parallel programs with blocking communication. Journal of Systems and Software, 2019, 155: 130–144. [doi: 10.1016/j.jss.2019.04.049]

[16] Gong DW, Pan F, Tian T, Yang S, Meng FL. A feedback-directed method of evolutionary test data generation for parallel programs. Information and Software Technology, 2020, 124: 106318. [doi: 10.1016/j.infsof.2020.106318]

[17] Sun BC, Wang JX, Gong DW, Tian T. Scheduling sequence selection for generating test data to cover paths of MPI programs. Information and Software Technology, 2019, 114: 190–203. [doi: 10.1016/j.infsof.2019.07.002]

[18] Du XZ, He HM, Liu JL. Test data generation of deterministic MPI parallel program based on path coverage. In: Proc. of the 2021 Int'l Conf. on Frontiers of Electronics, Information and Computation Technologies. Changsha: ACM, 2021. 91. [doi: 10.1145/3474198.3478213]

[19] 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]

[20] Jin YC. Surrogate-assisted evolutionary computation: Recent advances and future challenges. Swarm and Evolutionary Computation, 2011, 1(2): 61–70. [doi: 10.1016/j.swevo.2011.05.001]

[21] Luo WJ, Yi RK, Yang B, Xu PL. Surrogate-assisted evolutionary framework for data-driven dynamic optimization. IEEE Trans. on Emerging Topics in Computational Intelligence, 2019, 3(2): 137–150. [doi: 10.1109/TETCI.2018.2872029]

[22] Li GH, Zhang QF, Lin QZ, Gao WF. A three-level radial basis function method for expensive optimization. IEEE Trans. on Cybernetics, 2022, 52(7): 5720–5731. [doi: 10.1109/TCYB.2021.3061420]

[23] Zhan DW, Xing HL. A fast kriging-assisted evolutionary algorithm based on incremental learning. IEEE Trans. on Evolutionary Computation, 2021, 25(5): 941–955. [doi: 10.1109/TEVC.2021.3067015]

[24] Jiao RW, Xue B, Zhang MJ. Investigating the correlation amongst the objective and constraints in Gaussian process-assisted highly constrained expensive optimization. IEEE Trans. on Evolutionary Computation, 2022, 26(5): 872–885. [doi: 10.1109/TEVC.2021.3120980]

[25] Wang HD, Jin YC, Sun CL, Doherty J. Offline data-driven evolutionary optimization using selective surrogate ensembles. IEEE Trans. on Evolutionary Computation, 2019, 23(2): 203–216. [doi: 10.1109/TEVC.2018.2834881]

[26] Lin QZ, Wu XF, Ma LJ, Li JQ, Gong MG, Coello CAC. An ensemble surrogate-based framework for expensive multiobjective evolutionary optimization. IEEE Trans. on Evolutionary Computation, 2022, 26(4): 631–645. [doi: 10.1109/TEVC.2021.3103936]

[27] 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]

[28] Song ZS, Wang HD, He C, Jin YC. A Kriging-assisted two-archive evolutionary algorithm for expensive many-objective optimization. IEEE Trans. on Evolutionary Computation, 2021, 25(6): 1013–1027. [doi: 10.1109/TEVC.2021.3073648]

[29] Tan Z, Wang HD. A kriging-assisted evolutionary algorithm using feature selection for expensive sparse multi-objective optimization. In: Proc. of the 2020 IEEE Congress on Evolutionary Computation. Glasgow: IEEE, 2020. 1–8. [doi: 10.1109/CEC48606.2020.9185825]

[30] Sun BC, Gong DW, Pan F, Yao XJ, Tian T. Evolutionary generation of test suites for multi-path coverage of MPI programs with non-determinism. IEEE Trans. on Software Engineering, 2023, 49(6): 3504–3523. [doi: 10.1109/TSE.2023.3263509]

[31] Li HB, Li SH, Benavides Z, Chen ZZ, Gupta R. COMPI: Concolic testing for MPI applications. In: Proc. of the 2018 IEEE Int'l Parallel and Distributed Processing Symp. Vancouver: IEEE, 2018. 865–874. [doi: 10.1109/IPDPS.2018.00096]

[32] Davies T, Karlsson C, Liu H, Ding C, Chen ZZ. High performance linpack benchmark: A fault tolerant implementation without checkpointing. In: Proc. of the 2011 Int'l Conf. on Supercomputing. Tucson: ACM, 2011. 162–171. [doi: 10.1145/1995896.1995923]

[33] Schaich D, DeGrand T. Parallel software for lattice N=4 supersymmetric Yang–Mills theory. Computer Physics Communications, 2015, 190: 200–212. [doi: 10.1016/j.cpc.2014.12.025]

[34] Yu HB. Combining symbolic execution and model checking to verify MPI programs. In: Proc. of the 40th Int'l Conf. on Software Engineering: Companion. Gothenburg: IEEE, 2018. 527–529.

[35] Darling AE, Carey L, Feng WC. The design, implementation, and evaluation of mpiBLAST. In: Proc. of ClusterWorld Conf. & Expo and the 4th Int'l Conf. on Linux Clusters: The HPC Revolution. 2003. 13–15.

[36] Igel C, Heidrich-Meisner V, Glasmachers T. Shark. The Journal of Machine Learning Research, 2008, 9: 993–996.

[37] Das S, Suganthan PN. Differential evolution: A survey of the state-of-the-art. IEEE Trans. on Evolutionary Computation, 2011, 15(1): 4–31. [doi: 10.1109/TEVC.2010.2059031]

[38] Brest J, Greiner S, Boskovic B, Mernik M, Zumer V. Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. IEEE Trans. on Evolutionary Computation, 2006, 10(6): 646–657. [doi: 10.1109/TEVC.2006.872133]

孙百才(1990－), 男, 博士, CCF 专业会员, 主要研究领域为智能软件工程, 机器学习.

姚香娟(1975－), 女, 博士, 教授, 博士生导师, CCF 高级会员, 主要研究领域为变异测试, 运筹优化.

巩敦卫(1970－), 男, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为智能软件工程, 智能优化理论及应用.