文章编号：1672-3961(2026)01-0063-09 DOI：10.6040/j.issn.1672-3961.0.2024.177

基于主动学习的符号执行路径探索策略

朱东晴[1]，张骏温[1]，何莲英[1]，王睿[1]，刘吉强[1,2]，张大林[2,3*]
（1.北京交通大学计算机与信息技术学院，北京 100044；2.北京交通大学智能系统与安全实验室，北京 100044；3.北京交通大学网络空间安全学院，北京 100044）

摘要：针对符号执行路径爆炸问题，设计一种基于主动学习引导的符号执行路径探索策略（path exploration strategy for symbolic execution based on active learning，ALS）。预测待测状态池中所有状态奖励值，自动标注预测准确的高奖励值状态，反馈给模型以更新预测模型；根据模型性能和预测精度判断是否进入下一轮测试；最后一轮模型更新结束后，部分预测模型将自动进行迭代训练，生成多个子模型，共同用于符号执行的路径探索。试验结果表明，在相同试验条件下，ALS相较其他基线方法往往能够实现更高代码覆盖和分支覆盖率，发现更多未定义行为违规数和真实世界程序漏洞，且其生成测试用例质量较高，用作模糊测试初始种子能发现更多程序路径。

关键词：主动学习；符号执行；路径探索策略；奖励值
中图分类号：TP391 **文献标志码**：A
引用格式：朱东晴，张骏温，何莲英，等. 基于主动学习的符号执行路径探索策略[J]. 山东大学学报(工学版)，2026，56(1)：63-71.
ZHU Dongqing，ZHANG Junwen，HE Lianying，et al. Path exploration strategy for symbolic execution based on active learning[J]. Journal of Shandong University（Engineering Science），2026，56(1)：63-71.

Path exploration strategy for symbolic execution based on active learning

ZHU Dongqing[1]，ZHANG Junwen[1]，HE Lianying[1]，WANG Rui[1]，LIU Jiqiang[1,2]，ZHANG Dalin[2,3*]
（1. School of Computer and Information Technology of Beijing Jiaotong University，Beijing 100044；2. Intelligent Systems and Security Laboratory of Beijing Jiaotong University，Beijing 100044；3. Cyberspace Security Institute of Beijing Jiaotong University，Beijing 100044）

Abstract：To address the issue of symbolic execution path explosion，a path exploration strategy for symbolic execution based on active learning（ALS）was designed.In this study，the reward values of all states in the test state pool were predicted. The states with accurately predicted high reward values were automatically annotated and fed back to the model to update the prediction model. Based on the model′s performance and prediction accuracy，a decision was made on whether to proceed to the next round of testing. After the final round of model updates，some of the prediction models automatically underwent iterative training to generate multiple sub-models，which were then collectively employed for symbolic execution path exploration.The experimental results demonstrated that under the same test conditions，higher code coverage and branch coverage were often achieved by ALS compared to other baseline methods. Moreover，more undefined behavior violations and real-world program vulnerabilities were discovered，and test cases of higher quality were generated. When these test cases were utilized as initial seeds for fuzz testing，more program paths could be discovered.

Keywords：active learning；symbolic execution；path exploration strategy；reward values

收稿日期：2024-07-22
第一作者简介：朱东晴（1999— ），女，江西九江人，硕士研究生，主要研究方向为机器学习与软件测试。E-mail：2421340087@qq.com
*通信作者简介：张大林（1983— ），男，内蒙古赤峰人，副教授，博士生导师，博士，主要研究方向为软件工程理论与技术。
E-mail：dalin@bjtu.edu.cn

0 引言

符号执行是一种旨在提高软件质量的程序分析技术，它通过使用符号化输入替代程序变量，并根据符号集对程序中的变量、语句和表达式进行模拟执行[1]。

符号执行自提出以来，一直面临路径爆炸这一挑战。路径爆炸是指目标程序的路径数量过多，难以全面地进行符号分析[2]。路径爆炸问题限制符号执行的可扩展性。针对路径爆炸的问题，符号执行技术需要一种有效的机制来选择和执行有希望的状态，这些状态能够以最少的执行时间实现最高的代码覆盖率。由于在决定探索哪个状态时，往往无法预测该状态是否能以合理的成本增加代码覆盖率，因此仅凭状态的即时属性选择状态是不现实的。状态选择决策依赖于选定状态的未来执行路径以及未来的状态选择决策。由于这一基本限制，符号执行工具的路径探索过程一直依赖于人工设计的启发式方法，但是由于这些启发式方法缺乏有效的预测能力，导致状态选择依据过于局限，致使这些启发式方法很容易被卡在偏向于某种度量属性的部分程序中，而无法到达程序的其他部分。如果能在综合这些度量属性的基础上，提供一种动态的状态选择预测能力，那么符号执行技术将会更有效地提升测试用例的代码覆盖率，同时也可检测出更多的安全违规。

主动学习是一种迭代的半监督学习过程，其核心思想是从大量未标注的样本中挑选高质量的样本进行人工标注，再将这些标注后的样本加入训练集中进行学习。由于只有少部分样本需要人工标注，能够显著降低标注成本，也能提升回归模型的性能[3]。同时，根据设计的选择策略，防止选择的样本与已有训练样本重复，进一步提升训练样本集的质量，因此可用于构建高性能回归模型或对数据集进行标记[4]。目前主动学习主要被应用于解决标注数据过少的问题中，但忽略符号执行问题中的偏向预测问题，导致符号执行效率不高。

针对上述分析，本研究提出一种基于主动学习的符号执行路径探索策略（active learning search，ALS），以解决上述问题。首先，ALS 对待测状态池中的所有状态进行奖励值预测；然后，将预测较为准确的高奖励值状态进行自动化标注，并反馈给符号状态的奖励值预测模型以引导更新该预测模型；最后，根据预测模型的性能及其测试生成的质量判断是否进入下一轮测试。通过上述步骤，达到依据状态的未来可能执行路径来选取具有最高估计奖励状态的目的。

1 相关工作

1.1 符号执行技术

针对路径爆炸问题，许多研究工作已经提出不同的解决方法。一类工作提出路径选择的优化算法，即符号执行路径搜索策略[5-8]。例如，符号执行工具 KLEE[9] 中提供多种路径搜索策略，包括深度优先搜索策略、广度优先搜索策略、随机状态搜索策略、随机路径搜索策略等。一类工作提出使用函数摘要技术和循环摘要技术[10-13]，通过复用已经探索过的代码的执行信息，避免对同一代码的重复执行，提高符号执行效率。然而，这种方法在复杂程序中往往无法有效避免冗余执行，特别是具有复杂控制流的程序。还有一类工作提出通过状态合并[14-16]，即将几条路径合并成一个状态，以有效地减少需要遍历的路径。虽然这类工作可以显著减少路径数量，但是状态合并会导致路径约束更加复杂，增大约束求解的难度，并且可能导致某些潜在错误的遗漏。还有一些工作通过与其他程序分析技术[16-19]相结合，比如程序切片、污点分析、类型检查和编译优化等技术，修剪掉不感兴趣的路径，从而缩小符号执行的搜索空间，然而这种方法可能依赖于某些特定的程序属性，限制程序分析技术的通用性和适应性。

近年来，研究者们探索多种基于机器学习的符号执行路径搜索优化方法，探索如何利用机器学习技术提升符号执行的性能和有效性。然而，现有方法在实践中仍存在一些局限性。例如，文献[20]使用监督学习方法优化符号执行路径选择，虽然能够选择较优的状态进行路径探索，但其依赖于大量标注数据，且在面对程序状态空间非常庞大时，效果会降低。文献[21]提出通过离线学习自动生成符号执行搜索的启发式规则，但这种方法的启发式规则生成过程较为固定，难以适应动态变化的程序特性。文献[22]使用在线学习自适应地切换启发式规则，以提高执行效率，但其在线学习过程可能会受到初期学习阶段不充分的影响，导致探索效果较差。文献[23]通过机器学习处理涉及外部函数调用或符号执行中浮点运算的复杂路径条

件,对于一般路径的优化效果有限。文献[24]通过学习有用的模板减少被测程序的输入空间,虽然能够缩小搜索范围,但这种方法的适用性受到程序结构和测试环境的限制,可能无法广泛应用于所有类型的程序。

1.2　主动学习

根据主动查询方式的不同,主动学习主要分为基于数据流的主动学习和基于数据池的主动学习[25]。

基于数据流的主动学习方法中的未标记样本实例会按顺序提交至主动查询模块,由主动查询模块决定是否对当前的样本实例进行数据标注。基于流的主动学习方法不能对未标注样例逐一比较,需要对样例的相应评价指标设定阈值,当提交给选择引擎的样例评价指标超过阈值时,进行标注。此方法需针对不同的任务进行调整,难以作为一种成熟的方法投入使用。基于数据池的主动学习方法通过主动查询策略从未标注样本集中选择最有价值的样本进行数据标注,之后将其放入已标注样本集供模型学习模块进行学习。相较基于流的主动学习方法,基于池的主动学习方法每次都能选出当前样例池中对分类贡献度最高的样例,既降低了查询样例成本,也降低了标注代价,得到广泛使用。文献[26]提出一种基于池的回归主动学习方法,并将其应用于声学情感计算。文献[27]使用基于高维数据和卷积神经网络分类器的主动学习方法进行图像分类。本研究针对符号执行路径探索过程,提出一种基于状态池的主动学习方法,通过选择较优的符号状态来引导多个模型的迭代更新,实现更高的代码覆盖率和缺陷检测能力,从而解决符号执行面临的路径爆炸问题。

2　符号执行路径搜索策略

基于符号执行的路径探索策略依据符号状态等属性度量特征呈现多样化[2],但符号执行算法通常采用文献[28]所提算法,如算法 1 所示。

算法 1　符号执行算法

输入　一个编译后的程序 P,一个状态选择策略 S。

输出　一组测试用例集 $\boldsymbol{T}$。

初始化　符号状态集 $\boldsymbol{t}$ 和测试用例集 $\boldsymbol{T}$ 为空。

(1)更新符号状态集 $\boldsymbol{t}$ 和状态选择策略 S;

(2)while 符号状态集 $\boldsymbol{t}$ 不为空且未超时,do 根据状态选择策略 S 选择符号状态 $\boldsymbol{t}'$进行路径探索;

(3)　if 执行到 EXIT 指令或遇到安全冲突,则调用外部约束求解器生成一个测试用例;

(4)更新符号状态集 $\boldsymbol{t}$ 和程序分支 f;

(5)返回测试用例集 $\boldsymbol{T}$。大多数符号执行算法最终目标是在给定一个输入程序和状态选择策略情况下,在规定时间内找到一组覆盖率最大的测试用例,即

$$\max \frac{C}{E}, \tag{1}$$

式中,C 为测试用例 $\boldsymbol{T}$ 的代码覆盖率,E 为算法 1 的执行时间。

由于待处理状态数量是程序分支数量指数级,且难以预判所选状态对应的测试及代码覆盖率。因此,直接实现式(1)几乎不可行,需构建一个可预测状态的状态选择策略,以选出有望生成高质量测试的状态。

3　基于主动学习的符号执行路径探索策略

3.1　ALS 方案描述

ALS 框架如图 1 所示,各环节的具体实现方式如下。

(1) 测试生成。对于训练程序集合 $\boldsymbol{N}_0$,将多种启发式路径搜索策略应用于符号执行算法,得到 1 组含已探索状态的测试用例 $\boldsymbol{T}^*$。

(2) 特征提取与标签标注。为 $\boldsymbol{T}^*$ 中每条测试用例的每个状态分别提取 1 组特征向量 $\boldsymbol{F}$,计算每个符号状态的奖励值,得到初始监督数据集 $\boldsymbol{T}^*$ 并构建学习模型。初始化已标注的训练集程序 $\boldsymbol{L}_{\mathrm{R}}$ 为 $\boldsymbol{\varnothing}$,用剩余的程序文件 $\boldsymbol{N}_1$ 更新预测模型。

(3) 训练模型。更新监督数据集 $\boldsymbol{T}^*$,反馈最新测试结果给学习模型以更新预测模型。综合多种启发式策略和预测模型优点,优先探索单位时间内代码覆盖数较高的状态,生成高质量、高覆盖测试用例。

(4) 状态预测。在集合 $\boldsymbol{N}_1$ 上每次选择 1 个程序进行符号执行,即用更新后的预测模型进行基于主动学习的符号状态奖励值预测。

(5)模型评估。若模型预测均方误差 E_{MS} 小于阈值 e_1,上一轮训练整体损失 L_k 小于阈值 e_2,生成最终模型并测试剩余测试集程序进行测试,否则,使用预测模型预测测试中每个状态的奖励值,并选

择部分预测较为准确的高奖励值状态进行自动化标注,然后将其作为新的标注数据集加入原有的监督数据集中,跳回(2)进行下一轮的训练。本研究均方差阈值设置为 3.0,模型损失阈值设置为2.5,其他测试结果好坏标准则主要依据具体的测试程序本身。

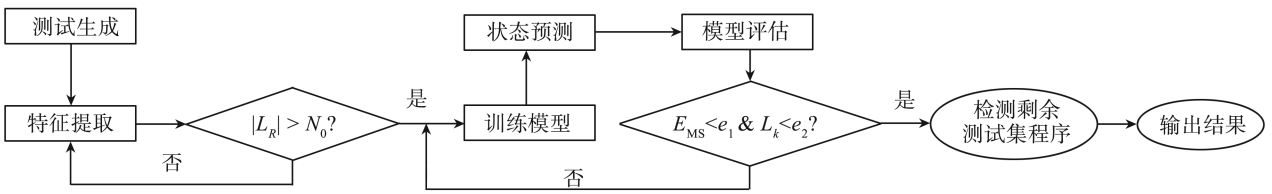

图 1 ALS 的符号执行与模型更新流程

Fig.1 The symbol execution and model update process of ALS

3.2 特征设置

为获取简单高效预测依据,本研究选取 KLEE 中已有的启发式路径探索策略所使用的特征以及符号状态的其他相对重要的特征,作为 ALS 的奖励预测输入,在保留启发式算法部分优势[28]的情况下,最大化符号状态的多种特征效果。本研究从符号状态中选取的特征主要分为静态特征和动态特征。

(1) 静态特征

本研究选取静态特征如下。

① stack:表示状态的当前调用栈大小。

② constraints:表示状态路径约束的词袋表示。

(2) 动态特征

本研究选取的符号状态动态特征是在程序执行过程中收集的,主要包括当前基本块的后继块数量、当前生成的测试用例数量等度量信息。

① successors:表示状态当前基本块后继数。

② genTestCases:表示当前生成测试用例数。

③ coverage:表示状态最新分支和探索的程序路径分别实现的新指令数和行覆盖数

④ depth:表示状态所在的路径已执行分支数。

⑤ cp_inst_count:表示状态当前函数内部执行的指令数。

⑥ inst_count:表示状态当前指令执行次数。

⑦ instSinceCovNew:表示自上次覆盖新指令以来,状态执行指令数。

⑧ subpath:表示状态子路径被访问次数。

3.3 训练模型

奖励值预测的目标是对程序文件生成的状态进行评分,从而判断在符号执行过程中是否选择该状态进行进一步路径探索。ALS 采用前馈神经网络[29](feedforward neural network, FNN)和循环神经网络[30](recurrent neural network, RNN)来构建预测模型。这两种神经网络已被证明是高校的回归模型,并且广泛应用于各类预测任务中[31]。

主动学习阶段,为避免异常值对模型拟合产生负面影响,ALS 有选择地从待测状态池选择部分状态 S_1 进行训练集更新操作,丢弃其他未被选择状态 S_2。算法 3 描述 ALS 模型框架具体生成过程,包括生成训练集和训练模型 2 个主要过程。

(1)生成训练集

为获取程序文件有效测试用例数据,先用 KLEE 对程序文件逐一进行符号执行,主要采用 KLEE 固有的启发式路径探索策略提取符号状态及其重要特征,得到每个状态对应特征向量 $\boldsymbol{F}$;再用 gcov 等工具为各状态计算 1 个奖励值 R 作为标签,且规定该奖励值由每个符号状态所属的所有测试的代码覆盖数 c 与该状态及其分支状态耗时 E' 之比度量,即

$$R_s=\frac{c}{E'}, \tag{3}$$

式中,R 为符号状态的奖励值,c 为符号状态所属所有测试的代码覆盖数,E' 为符号状态及其分支的总耗时。

计算各状态奖励值后,结合之前得到的 $\boldsymbol{F}$ 构建模型训练集。符号执行过程中,状态及其产生的测试取决于当前步骤未知的未来选择,无法在符号执行完成前确定各状态奖励值,需用预测模型对符号执行时状态奖励值进行有效预测。

(2)训练子模型

先在已测得的初始训练程序集合 $\boldsymbol{N}_0$ 所对应训练集 $\boldsymbol{T}^*$ 上,训练 1 个 FNN 预测器和 1 个 RNN 预测器评估各状态奖励值,再对这 2 个预测器通过高奖励值引导的主动学习进行优化,随后对优化后 FNN 预测器进行 2 轮迭代,生成 2 个改进 FNN 预测器,最终由 3 个 FNN 预测器和 1 个 RNN 预测器构成预测模型。

算法 2 描述 ALS 模型训练过程,其中 1 至 5 行

是初始模型训练,6 至 18 行是生成递增训练集并及 2 个初始模型主动学习,19 至 22 行是 FNN 迭代训练。选择符号状态时优先探索预测较为准确的高奖励值状态。鉴于许多真实世界程序含序列数据结构,RNN 能处理序列数据并捕捉序列中元素间时间依赖关系,本研究在 FNN 有效预测基础上引入 RNN 进行集成,以提高奖励值预测的准确性和效率。应用这 4 个预测模型进行符号执行测试时,ALS 平均分配时间预算给各模型,组合预测结果形成最终测试结果。

算法 2　ALS 的模型训练算法

输入　初始训练程序集合 $\boldsymbol{N}_0$,初始训练集 $\boldsymbol{T}^*$,待测文件集 $\boldsymbol{N}_1$,选取的状态数 n,迭代的次数 r。

输出　路径探索模型 ALS。

初始化　模型集。

(1) 使用初始训练集 $\boldsymbol{T}^*$ 分别训练初始 feed 模型和初始 RNN 模型

(2) for $i \in \boldsymbol{N}_1$ do;

(3)　if 初始模型不满足阈值 e_1 和 e_2;

(4)　　迭代更新模型,更新训练集 $\boldsymbol{T}^*$;

(5)　更新模型集;

(6) for $j \in r$ do;

(7)　　使用文件集 $\boldsymbol{N}_0$ 和 $\boldsymbol{N}_1$,继续迭代训练子类模型;

(8)　　更新模型集;

(9)返回路径探索模型 ALS。

3.4　状态预测

本研究扩展现有符号执行工具 KLEE,重点执行奖励值预测模型所选符号状态。ALS 用训练好的初始预测模型,从待测试文件集合 $\boldsymbol{N}_1$ 中选程序结构最复杂的项目文件作为首个测试文件进行预测,其余程序文件通过随机采样法选取。每次预测后,用 KLEE 及奖励值计算方法对预测结果进行评判,评判准则为模型预测均方误差、生成测试用例耗时、测试覆盖情况等测试结果。通过测试结果实时更新奖励值预测模型参数以提升预测精度。

4　试验与分析

4.1　试验环境及数据集

试验平台基于 Intel(R) Xeon(R) Gold 6130 CPU @ 2.10 GHz 与 128 GB RAM 中的 docker 容器,操作系统为 Ubuntu 18.04。试验数据集选用 coreutils(版本 8.31)套件及 6 个真实世界程序。训练模型用 coreutils 套件中部分程序文件,模型测试用 coreutils 部分套件以及 6 个真实世界的程序文件,coreutils 套件中所有软件包使用相同输入配置命令。试验所用程序项目基本信息如表 1 所示。

表 1　coreutils 套件及真实世界程序基本信息

Table 1　The basic information about the coreutils suite and the real-world programs

程序名	版本	程序代码行数	二进制文件大小/KB	符号输入设置
coreutils	8.31	713 583	140	-sym-args 0 1 10 -sym-args 0 2 2 -sym-files 1 8 -sym-stdin 8
diff	3.7	186 156	544	--sym-args 0 2 2 A B --sym-files 2 50
grep	3.6	191 522	583	--sym-args 0 2 2 --sym-arg 10 A --sym-files 1 50
patch	2.7.6	98 090	2 447	--sym-args 0 2 2 A B --sym-files 2 50
readelf	2.36	18 042	2 434	-a A --sym-files 1 100
make	4.3	117 421	474	-n -f A --sym-files 1 40
sqlite	3.33.0	161 663	2 048	--sym-stdin 20

4.2　复杂度分析

选取 7 种基线方法,其中 4 种为 KLEE 中固有启发式策略:随机路径搜索策略(random-path search, rps)、随机状态搜索策略(random-state search, rss)、nurs:cpicnt(nurc)策略、nurse:depth(nurd)策略[32-33],3 种为本研究自行集成的策略:sgs 混合策略、rpsnd 混合策略和 learch 策略[20]。其中,sgs 混合策略由 sgs:1、sgs:2、sgs:4 这 3 种策略组成,rpsnd 混合策略由 random-path、nurs:cpicnt、nurs:depth 这 3 种策略组成。其中,rps 需要构建一个二叉树表示执行路径,其时间复杂度为 $O(n^2)$,空间复杂度为 $O(2^d+n)$,d 为路径深度。rss 的时间复杂度为 $O(n)$,空间复杂度为 $O(n)$。非均匀搜索策略 nurs:cpicnt 和 nurs:depth、的时间复杂度都为 $O(n^2)$,空间复杂度都为 $O(n)$。混合策略 sgs 由 3 个运行时间相等的独立子策略构成,每个子策略需存储各状态子路径探索历史,因此 sgs 时间复杂度为 $O(n^2)$,空间复杂度为 $O(n)$。rpsnd 混合策略时间复杂度为 $O(n^2)$,空间复杂度为 $O(2^n+3n)$。learch 策略复杂度取决于特征提取复杂度和机器学习模型复杂度,时间复杂度为 $O(nh^2+fn)$,空间复杂度为 $O(fn)$,其中 f 为特征数量,h 为隐藏层维度。

本研究所提 ALS 时间复杂度主要由样本选择时间复杂度 $O(nm)$、模型更新时间复杂度 $O(nh^2+nfh+nth^2)$ 和模型评估时间复杂度 $O(n)$3 部分组成,其中 t 为序列长度。ALS 时间复杂度为这 3 部分复杂度之和与迭代次数 k 的乘积。ALS 空间复杂度包含符号状态存储空间复杂度 $O(fn)$ 和模型存储空间复杂度 $O(3p\_FNN+p\_RNN)$,因此 ALS 空间复杂度为 $O(fn)+O(3p\_FNN+p\_RNN)$,其中,p_FNN 为 FNN 的参数数量,p_RNN 为 RNN 参数数量。相比基线方法,ALS 在优化符号执行路径探索时,时间和空间复杂度较高,但通过有效样本选择和模型更新,显著提高符号执行路径探索效率。

本研究从代码覆盖能力、安全违规检测能力、混合测试 3 个方面对 ALS 有效性进行验证。

4.3 试验结果

4.3.1 ALS 代码覆盖能力有效性验证

选取 37 个 coreutils 项目文件和 6 个真实世界程序,各进行 5 轮测试,取 5 轮测试结果平均值作为试验结果。

对 37 个 coreutils 程序在不同搜索策略下进行 1 h符号执行,代码行覆盖情况如表 2 所示。由表 2 可知,ALS 策略相比其他搜索策略覆盖了最多的代码行。

37 个 coreutils 程序在不同搜索策略经过 1 h 符号执行后,所达到的最佳覆盖程序数及分支覆盖率结果如表 3 所示。由表 3 可知,在 37 个 coreutils 程序中,ALS 的最佳覆盖数和分支覆盖率最高,对 18 个程序实现最佳覆盖,并且 ALS 分支覆盖率比次优策略高出 3.69%。

表 2 37 个 coreutils 程序在不同搜索策略下符号执行 1 h 覆盖的代码行数对比

Table 2 Comparison of the number of code lines covered by symbol execution in 1 hour for 37 coreutils programs with different search strategies

搜索策略	rss	nurc	nurd	sgs	learch	rps	rps_nc_nd	ALS
覆盖的代码行数	459	490	521	530	537	539	541	548

表 3 37 个 coreutils 程序在不同搜索策略下符号执行 1 h 的最佳覆盖数以及平均分支覆盖率

Table 3 The optimal coverage and BCOV(%) of 37 coreutils programs under symbol execution for 1 hour with different search strategies

搜索策略	rss	nurc	nurd	sgs	learch	rps	rps_nc_nd	ALS
最佳覆盖数	6	4	7	11	15	12	15	18
分支覆盖率/%	21.05	21.61	22.68	22.93	23.05	23.02	22.50	26.74

对 6 个真实世界程序在不同搜索策略下进行 2、5 h 的符号执行测试,代码行覆盖情况分别如表 4 和表 5 所示。由表 4 和表 5 可知,仅 rps_nc_nd 和 learch 两种策略偶尔存在与 ALS 代码覆盖数大致相等的情况。在 2、5 h 符号执行时间下,ALS 平均代码覆盖数比次优策略分别提高 39.4%和 23.6%。

4.3.2 ALS 违规检测能力有效性验证

本研究采用 Clang 未定义行为检测器(undefined behavior sanitizer, UBSan)对输入程序进行检测,并标记 5 种安全违规行为,分别为整数溢出、越界数组读写、指针溢出、过大的位移操作以及空指针解引用,当符号执行工具检测到以上任意一种违规行为时,则会生成对应的违规行为的测试用例,用于检测该漏洞。

37 个 coreutils 程序在不同搜索策略下进行 1 h 符号执行所检测到的未定义违规数结果如表 6 所示。实验结果表明,ALS 的违规检测能力明显优于其他策略。单个搜索策略中,nurd 表现最佳,组合策略中,ALS 违规检测能力相较次优策略 sgs 提高 6%。

表 4 6 个真实世界程序在不同搜索策略下运行 2 h 的代码行覆盖数

Table 4 The number of lines covered by running 6 real-world programs for 2 hours with different search strategies

程序名	覆盖数/行							
	rss	nurc	nurd	sgs	learch	rps	rps_nc_nd	ALS
diff	486	487	482	484	764	485	540	764
grep	0	1 818	0	0	569	0	1 818	1 869
make	1 438	1 383	2 041	1 453	2 053	1 427	2 059	2 059
patch	294	219	216	302	848	219	219	873
readelf	396	408	444	331	429	421	486	921
sqlite	2 116	2 122	2 151	2 211	7 020	0	2 214	9 683

表 5　6 个真实世界程序在不同搜索策略下运行 5 h 的代码行覆盖数

Table 5　The number of lines covered by running 6 real-world programs for 5 hours with different search strategies

程序名	覆盖数/行							
	rss	nurc	nurd	sgs	learch	rps	rps_nc_nd	ALS
diff	486	487	482	484	1 495	485	540	1 495
grep	0	1 818	0	0	569	0	1 818	1 869
make	1 434	1 383	2 315	1 453	2 489	1 427	2 355	2 513
patch	294	219	216	302	860	219	219	882
readelf	399	931	948	409	897	459	934	934
sqlite	2 197	2 218	2 285	2 211	9 065	505	2 222	11 305

6 个真实世界程序在不同搜索策略下运行 2、5、8 h 所发现的 ubsan 违规数如图 2 所示。实验结果表明 ALS 违规检测能力优于其他策略。不论对何种测试数据,sgs 违规检测能力仅次于 ALS,且随运行时间增加,代码覆盖数和违规检测能力往往都会趋于平稳,达到对应搜索策略搜索的上限。

表 6　37 个 coreutils 程序在不同搜索策略下运行 1 h 发现的 ubsan 违规数

Table 6　The number of ubsan violations found by running 37 coreutils programs for 1 hour with different search strategies

搜索策略	rss	nurc	nurd	sgs	learch	rps	rps_nc_nd	ALS
漏洞数量/个	24	23	30	89	83	29	76	95

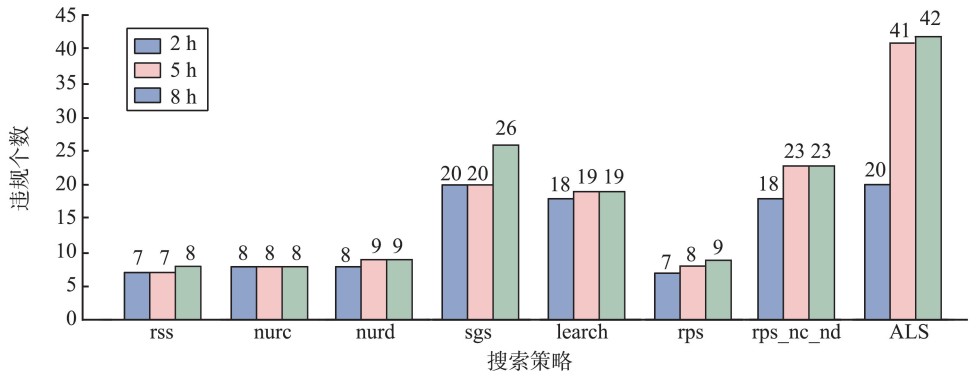

图 2　6 个真实世界程序程序在不同搜索策略下分别运行 2、5、8 h 发现的 ubsan 违规数

Fig.2　The number of ubsan violations found by running 6 real-world programs respectively for 2 hour, 5 hour and 8 hour with different search strategies

4.3.3　ALS 在混合测试方面的有效性验证

本研究结合符号执行和模糊测试技术,实施混合测试法进一步验证 ALS 的有效性。使用不同搜索策略进行符号执行以生成测试用例,将这些测试用例作为模糊测试工具 AFL(american fuzzy lop)的初始种子[34],以此观察不同搜索策略在模糊测试中的表现。实验随机选取 3 个真实世界程序,对每个程序进行 5 次测试,实验结果取平均值。符号执行生成测试用例时间设定为 8 min,模糊测试执行时间设定为 1 h。8 种符号执行路径探索策略生成的测试用例在模糊测试中的表现如表 7 所示。实验结果表明,相同实验条件下,ALS 生成的测试用例性能最佳,触发路径数量比次优策略高 8.69%。

4.4　ALS 在主动学习框架下的有效性分析

将本研究所提符号执行路径探索策略 ALS 与文献[35]多策略主动学习搜索策略(multi-strategy active learning search, MS-ALS)进行比较。在代码覆盖能力方面,使用相同部分数据集进行符号执行时,ALS 在与训练集结构相似的 coreutils 测试数据上稍逊于 MS-ALS,少覆盖 3.31%;ALS 在真实世界程序上代码覆盖能力显著更优,超出 14.63%。在漏洞检测能力验证和混合测试有效性验证方面,不同数据集的表现也有所不同。ALS 在与训练集结构类似的 coreutils 测试数据上表现稍微欠佳,但在真实世界程序上的各种表现都明显优于 MS-ALS。主要原因在于:ALS 集成多种机器学习模型优点,增强对未知且复杂路径的适应性,而 MS-ALS 训练时对数据过度拟合,对训练数据集依赖性强,导致其在真实世界程序中表现不如 ALS;ALS 采用相对简单的主动学习策略,避免复杂策略可能带来的高价值状态判断错误、资源消耗过大等问题,在不同程序场景中表现更稳定,MS-ALS 的多策略选择在面对真实世界程序的多样化时,可能效果不佳。

表7 3个真实世界程序进行模糊测试1 h所发现的路径数

Table 7 The number of paths discovered by fuzzing for 3 real-world programs about 1 hour

程序名	路径数/个							
	rss	nurc	nurd	sgs	learch	rps	rps_nc_nd	ALS
patch	1 023	901	1 118	1 108	1 204	1 056	1 083	1 259
readelf	1 185	1 317	1 253	1 123	1 343	1 247	1 527	1 594
sqlite	1 327	1 340	1 246	1 157	1 816	935	1 199	1 889

5 结论

本研究针对符号执行技术面临的挑战及主流符号执行路径探索策略局限性,提出一种基于主动学习的符号执行路径探索策略ALS。它采用基于主动学习的状态选择机制,利用测试结果和模型预测均方误差等考量是否进行下一轮奖励值预测模型训练,精准识别较高奖励值状态,提高测试效率。综合多种启发式搜索策略及机器学习模型优点,借助KLEE平台自动标注符号状态奖励值,避免专家标注繁琐的问题,提高路径探索效率。实验结果表明,ALS策略能为符号执行路径探索提供更优决策,有效提升软件测试效率和质量。

在未来工作中,可从控制流图等方面丰富符号状态特征,优化基于池的主动学习策略,从而提高路径探索模型的代码覆盖率;也可在尝试多种机器学习模型高效集成同时,对庞大数量的符号状态及其分支进行有效删减,以提高软件测试效率。

参考文献:

[1] 吴皓, 周世龙, 史东辉, 等. 符号执行技术及应用研究综述[J]. 计算机工程与应用, 2023, 59(8): 56-72.
WU Hao, ZHOU Shilong, SHI Donghui, et al. Review of symbolic execution technology and applications[J]. Computer Engineering and Applications, 2023, 59(8): 56-72.

[2] CADAR C, SEN K. Symbolic execution for software testing: three decades later[J]. Communications of the ACM, 2013, 56(2): 82-90.

[3] 陈德蕾, 王成, 陈建伟, 等. 基于门控循环单元与主动学习的协同过滤推荐算法[J]. 山东大学学报(工学版), 2020, 50(1): 21-27.
CHEN Delei, WANG Cheng, CHEN Jianwei, et al. GRU-based collaborative filtering recommendation algorithm with active learning[J]. Journal of Shandong University (Engineering Science), 2020, 50(1): 21-27.

[4] 龚楷伦, 翟婷婷, 唐鸿成. 一种面向多标签分类的在线主动学习算法[J].山东大学学报(工学版), 2022, 52(2): 80-88.
GONG Kailun, ZHAI Tingting, TANG Hongcheng. An online active learning algorithm for multi-label classification[J]. Journal of Shandong University (Engineering Science), 2022, 52(2): 80-88.

[5] BALDONI R, COPPA E, D'ELIA D C, et al. A survey of symbolic execution techniques[J]. ACM Computing Surveys (CSUR), 2019, 51(3): 1-39.

[6] AVGERINOS T, CHA S K, REBERT A, et al. Automatic exploit generation[J]. Communications of the ACM, 2014, 57(2): 74-84.

[7] MA K K, YIT PHANG K, FOSTER J S, et al. Directed symbolic execution[C]//Static Analysis: 18th International Symposium. Venice, Italy: Springer-Verlag, 2011: 95-111.

[8] ZHANG Y F, CHEN Z B, WANG J, et al. Regular property guided dynamic symbolic execution[C]// Proceedings of the 37thInternational Conference on Software Engineering. Florence, Italy: IEEE, 2015: 643-653.

[9] CADAR C, DUNBAR D, ENGLER D R. Klee: unassisted and automatic generation of high-coverage tests for complex systems programs[C]// 8th USENIX Symposium on Operating Systems Design and Implementation. San Diego, USA: USENIX, 2008: 209-224.

[10] GODEFROID P. Compositional dynamic test generation [C]//Proceedings of the 34th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming languages. Nice, USA: ACM, 2007: 47-54.

[11] ANAND S, GODEFROID P, TILLMANN N. Demand-driven compositional symbolic execution[C]//Tools and Algorithms for the Construction and Analysis of Systems: 14th International Conference, TACAS 2008. Berlin, Germany: Springer, 2008: 367-381.

[12] GODEFROID P, LUCHAUP D. Automatic partial loop summarization in dynamic test generation[C]// Proceedings of the 2011 International Symposium on Software Testing and Analysis. New York, USA: ACM, 2011: 23-33.

[13] XIE X F, CHEN B H, LIU Y, et al. Proteus: computing disjunctive loop summary via path dependency analysis[C]//Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering. New York, USA: ACM, 2016: 61-72.

[14] KINDER J. Efficient symbolic execution for software testing[C]//FMCAD. Lausanne, Switzerland: IEEE,

2014: 5.
[15] 邓维, 李兆鹏. 形状分析符号执行引擎中的状态合并[J]. 计算机科学, 2017, 44(2): 209-215.
DENG Wei, LI Zhaopeng. State merging in shape analysis symbolic execution engine[J]. Computer Science, 2017, 44(2): 209-215.
[16] KHOO Y P, CHANG B Y E, FOSTER J S. Mixing type checking and symbolic execution[C]//Proceedings of the 31st ALM SIGPLAN Conference on Programming Language Design and Implementation. New York, USA: ACM, 2010: 436-447.
[17] SHOSHITAISHVILI Y, WANG R Y, HAUSER C, et al. Firmalice-automatic detection of authentication bypass vulnerabilities in binary firmware[C]//NDSS 2015. San Diego, USA: Internet Society Press, 2015: 8-11.
[18] CHA S K, AVGERINOS T, REBERT A, et al. Unleashing mayhem on binary code[C]//2012 IEEE Symposium on Security and Privacy. San Francisco, USA: IEEE, 2012: 380-394.
[19] GAO F J, WANG L Z, LI X D. Bovinspector: automatic inspection and repair of buffer overflow vulnerabilities[C]//Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering. New York, USA: ACM, 2016: 786-791.
[20] HE J X, SIVANRUPAN G, TSANKOV P, et al. Learning to explore paths for symbolic execution[C]//Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. New York, USA: ACM, 2021: 2526-2540.
[21] CHA S, HONG S, LEE J, et al. Automatically generating search heuristics for concolic testing[C]//Proceedings of the 40th International Conference on Software Engineering. New York, USA: ACM, 2018: 1244-1254.
[22] CHA S, OH H. Concolic testing with adaptively changing search heuristics[C]//Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. New York, USA: ACM, 2019: 235-245.
[23] LI X, LIANG Y J, QIAN H, et al. Symbolic execution of complex program driven by machine learning based constraint solving[C]//Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering. New York, USA: ACM, 2016: 554-559.
[24] CHA S, LEE S, OH H. Template-guided concolic testing via online learning[C]//Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering. New York, USA: ACM, 2018: 408-418.
[25] Settles B. Active learning literature survey[R]. University of Wisconsin-Madison, USA: Computer SciencesDepartment, 2009.
[26] WU D R. Pool-based sequential active learning for regression[J]. IEEE Transactions on Neural Networks and Learning Systems, 2018, 30(5): 1348-1359.
[27] BELUCH W H, GENEWEIN T, NÜRNBERGER A, et al. The power of ensembles for active learning in image classification[C]// 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition. Salt Lake City, USA: IEEE, 2018: 9368-9377.
[28] LI Y, SU Z D, WANG L Z, et al. Steering symbolic execution to less traveled paths[C]//Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications. Indianapolis, USA: ACM, 2013: 19-32.
[29] SVOZIL D, KVASNICKA V, POSPICHAL J. Introduction to multi-layer feed-forward neural networks[J]. Chemometrics and Intelligent Laboratory Systems, 1997, 39(1): 43-62.
[30] LIPTON Z C, BERKOWITZ J, ELKAN C. A critical review of recurrent neural networks for sequence learning[EB/OL]. (2015-05-29)[2024-07-22]. https://arxiv.org/abs/1506.00019
[31] SAHA S, RAGHAVA G P S. Prediction of continuous B-cell epitopes in an antigen using recurrent neural network[J]. Proteins: Structure, Function, and Bioinformatics, 2006, 65(1): 40-48.
[32] BUSSE F, NOWACK M, CADAR C. Running symbolic execution forever[C]//Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis. New York, USA: ACM, 2020: 63-74.
[33] BURNIM J, SEN K. Heuristics for scalable dynamic test generation[C]//2008 23rd IEEE/ACM International Conference on Automated Software Engineering. L′Aquila, Italy: IEEE, 2008: 443-446.
[34] OGNAWALA S, HUTZELMANN T, PSALLIDA E, et al. Improving function coverage with munch: a hybrid fuzzing and directed symbolic execution approach[C]//Proceedings of the 33rd Annual ACM Symposium on Applied Computing. Pau, France: ACM, 2018: 1475-1482.
[35] HE L Y, ZHANG D L, ZHU D Q, et al. Path exploration strategy for symbolic execution based on multi-strategy active learning[C]// Proceedings of the 15th Asia-Pacific Symposium on Internetware. New York, USA: ACM, 2024: 165-168.

(编辑:管昕昉)