基于深度强化学习的类集成测试序列生成方法

张颖辉[1,2],张艳梅[1,2,3],张志成[4],姜淑娟[1,2],丁艳茹[1,2],袁 冠[1,2,3]

(1. 中国矿业大学矿山数字化教育部工程研究中心,江苏徐州 221116;
2. 中国矿业大学计算机科学与技术学院,江苏徐州 221116; 3. 广西可信软件重点实验室(桂林电子科技大学),广西桂林 541004;
4. 南方科技大学工学院计算机科学与工程系,广东深圳 518055)

摘 要: 类集成测试序列的生成是面向对象软件测试中的关键步骤,当类的测试序列不同时,相应的测试代价也不相同. 在集成测试中生成一个合理的类集成测试序列可以有效降低软件测试的代价. 本文将深度强化学习中的Advantage Actor-Critic算法应用于解决类集成测试序列生成问题. 首先,利用类间各种依赖关系构建与智能体交互的环境模型;然后,记录智能体从初始状态到终止状态的路径,即每次选择的动作对应每次选择集成到序列的类编号;最后,得出最终的类集成测试序列. 实验结果表明,本文方法所得到的类集成测试序列花费的总体测试桩复杂度,在选取的7个项目中有5个表现最佳,在剩余2个项目中表现中等.

关键词: 集成测试;测试序列;深度强化学习;advantage actor-critic;测试桩复杂度

基金项目: 国家自然科学基金(No.61673384,No.71774159);中国博士后基金特别资助(No.2021T140707);广西可信软件重点实验室研究课题(No.kx201609);

中图分类号: TP311 **文献标识码:** A **文章编号:** 0372-2112(2023)02-0455-12

电子学报 URL:http://www.ejournal.org.cn **DOI:**10.12263/DZXB.20210688

Generation Method of Class Integration Test Order Based on Deep Reinforcement Learning

ZHANG Ying-hui[1,2], ZHANG Yan-mei[1,2,3], ZHANG Zhi-cheng[4], JIANG Shu-juan[1,2], DING Yan-ru[1,2], YUAN Guan[1,2,3]

(1. *Mine Digitization Engineering Research Center of the Ministry of Education*, *China University of Mining and Technology*, *Xuzhou*, *Jiangsu* 221116, *China*;
2. *School of Computer Science and Technology*, *China University of Mining and Technology*, *Xuzhou*, *Jiangsu* 221116, *China*;
3. *Guangxi Key Laboratory of Trusted Software*, *Guilin*, *Guangxi* 541004, *China*;
4. *Department of Computer Science and Engineering*, *College of Engineering*, *Southern University of Science and Technology*, *Shenzhen*, *Guangdong* 518055, *China*)

Abstract: The generation of class integration test order is the key step in object-oriented software testing. When the class integration test order is different, the corresponding test cost is different. Generating a reasonable class integration test order in integration testing can effectively reduce the cost of software testing. This paper applies the advantage actor-critic algorithm in deep reinforcement learning to solve the problem of class integration test order generation. Firstly, the environment model of interaction with agents is constructed by using various dependencies between classes. Then, the path of the agent from the initial state to the termination state is recorded, that is, each selected action corresponds to each selected class number integrated into the order. Finally, the final class integration test order is obtained. The experimental results show that the total test stubs complexity of class integration test order cost obtained by the method in this paper has the best performance in 5 out of 7 selected subjects, and the average performance in the remaining 2 subjects.

Key words: integration test; test order; deep reinforcement learning; advantage actor-critic; complexity

Foundation Item(s): National Natural Science Foundation of China (No.61673384, No.71774159); Postdoctoral Foundation of China (No.2021T140707); Guangxi Trusted Software Key Laboratory (No.kx201609)

收稿日期:2021-05-08;修回日期:2022-05-31;责任编辑:王天慧

1 引言

软件的开发周期中,软件设计与测试阶段的占比最大,一个合理的软件测试序列会降低测试阶段的成本和开发人员的改进时间. 在诸如Java这种面向对象的项目中,包含多个类且类间依赖关系错综复杂,无法直接确定类测试的先后顺序,因此在测试前,确定每个类的测试顺序十分有必要. 生成类的测试顺序被称为类集成测试序列(Class Integration Test Order,CITO)生成方法. CITO最早由Kung等人[1]提出,他们将学生分为两组,分别为Interviews系统(含有122个类)编写测试桩. 小组1按照随机序列进行编写测试桩,平均需要191个测试桩,花费152个小时;小组2按照尽可能优的测试序列进行编写测试桩,大概花费7个小时. 由此可知,合理的类集成测试序列对软件测试具有相当重要的研究意义.

根据测试需求不同,CITO结果的好坏评价标准大致分为两类:(1)生成类集成测试序列所需测试桩个数;(2)生成类集成测试序列所需的测试桩代价总和,即总体测试桩复杂度. Kung等人[1]是第一个将测试桩个数作为评价序列优劣的指标. 假设在进行集成测试时,系统内存在两个类A和B,类A依赖类B,且类A先于类B被测试,但是此时类B尚未被测试,为了顺利完成类A的测试,就需要模拟类B的某些功能模块,这些模拟的功能模块就是测试桩. 测试桩分通用测试桩和特定测试桩,其中通用测试桩是为当前类建立全部功能的测试桩,特定测试桩仅仅为当前类建立特定功能(建立其他类依赖当前类的个别功能). 由于每个测试桩所需的代价不同,因此,仅仅从测试桩个数的角度分析结果的优劣是不全面的. Briand等人[2]提出将类间的方法调用和属性依赖结合起来作为测试桩复杂度,并根据需求的不同,可以调节方法和属性在复杂度中的占比. 总体测试桩复杂度是衡量CITO优劣较早的标准,也是目前为止使用最为广泛的指标. 类集成测试序列的总体测试桩复杂度越低,表示按照此顺序进行类集成测试所花费的代价也就越低,因此本文以总体测试桩复杂度为衡量测试序列优劣的标准.

近些年来,人工智能相关算法被越来越广泛地应用在各领域的研究中. Czibula等人[3]将机器学习中的强化学习引入到CITO中,并获得了不错的效果. 对于动作空间小的情况,强化学习可以通过建立Q表的方式,经过不断训练更新Q表,使之逼近Q*函数. 但随着动作空间的增大,状态的增多,要想维护这样的Q表是不切实际的. 因此,本文引用神经网络的方式来近似Q*函数,即深度强化学习(Deep Reinforcement Learning,DRL),通过训练神经网络让智能体得到总体测试桩复杂度最低的测试序列.

本文工作主要有以下贡献:

(1)提出将DRL和CITO相结合,通过此机器学习算法来生成类集成测试序列,这个方法是全新的,目前尚未有人在此做过尝试.

(2)通过引入类净收益作为DRL中智能体执行动作后的奖励值,很好地解决了DRL和CITO的兼容性.

(3)为生成类集成测试序列开辟一个新的思路,DRL尚有可以改进的方式,之后可在此基础上根据CITO的需求优化DRL的算法.

本文后续将从以下几个方面进行展开:第二部分介绍CITO相关的现有的国内外研究现状;第三部分介绍DRL的知识背景;第四部分详述CITO与DRL的结合过程;第五部分分析和讨论实验结果;最后第六部分进行总结和展望.

2 相关工作

目前,现有的类集成测试序列生成方法主要包括基于图论的方法、基于启发式的方法、基于切片的方法、基于复杂网络的方法以及基于强化学习的方法.

基于图论的方法最初由Kung等人[4]提出,他提出构建项目的对象关系图(Object Relation Diagram,ORD)[4],若生成的图中含环路就先进行破环,然后再对不含环路的图进行逆向拓扑排序,由此得出类集成测试序列. Tai等人[5]根据强连接关系(继承、聚集等)和弱连接关系(关联)分别创建了主级顺序和次级顺序,为每条边赋值为起点入度与终点入度之和,根据边的权值由大到小破除环路. Le Traon等人[6]通过遍历ORD的所有节点查找叶边,根据叶边个数给予该节点权值,在破环的过程中删除权值大的节点的所有入边,但该方法未考虑类间依赖关系的强弱. Briand等人[7]结合了Tai等人[5]和Le Hanh等人[8]的方法,先通过Zaidma等人[9]的算法对ORD进行深度优先搜索,划分不同的强连通分量(Strong Connected Component,SCC),再依次删除权值最大的关联边来破环. Hashim[10]利用类间交互的方法个数和属性个数度量依赖关系,类间依赖关系强度越低,构建的测试桩复杂度就越低. 张艳梅[11,12]提出将静态依赖和动态依赖结合起来的删边规则,满足了测试桩个数低的要求. 除此之外,Chen等人[13]、郑磊[14]的研究也将动态依赖考虑在内. 在基于图论的方法中,破除环路需要多次迭代查找环路,耗费大量无用功,使测试成本增加.

对于基于启发式的方法,Briand等人[2]和Le Hanh等人[8]都曾尝试使用遗传算法来得出最优的测试序列. 他们的方案是建立在遗传算子不变的情况下,尽可能不打破强连接关系的环,检测交叉变异后是否生成了新的个体. 利用得出的类集成测试序列的总体测试

桩复杂度作为对新个体的评价,最终得到最佳类集成测试序列. Borner等人[15]引入了模拟退火的搜索方式,并将属性复杂度、方法复杂度和测试焦点作为优化的目标来生成类集成测试序列. Wang等人[16,17]对上述两种方法进行改进,利用类间耦合关系计算打破类间关系所需代价,以总体测试桩代价最小的原则得出新的测试序列. 除此之外,部分研究人员发现采用多目标优化的算法应用于生成类集成测试序列,取得了不错的结果. Vergilio等人[18]根据禁忌搜索的思想,通过不重复产生已存在的序列降低搜索时间,对于已产生的序列,取其中的最优解放入候选列表中. Assunção等人[19]研究了NSGA-II[20],SPEA2[21]以及PAES[22]这三种基于帕累托的多目标优化算法,并将它们应用于生成类集成测试序列,在实施过程中保留每代进化过程中的优秀个体,利用拥挤距离和分布密度来度量种群多样性,通过广度搜索和深度搜索寻找帕累托最优解. 应用启发式的方法解决CITO问题时,搜索空间大小通常是类的个数的阶乘,导致其时间复杂度过高,因此,对于类个数较多的项目,此类方法效率低下.

在基于切片的方法中,Jaroenpiboonkit等人[23]把研究重点放在了类上面,并同时对其进行相应切分,该方法无须构建测试桩便可以打破原有环路,直接得出类集成测试序列. 刘颖莲[24]根据ORD中的节点权值进行切片,并为每个切片进行赋值,然后进行集成测试. 但是切片技术作用单元是方法,其打破了类之间的结构,因此在实际应用中需要对不同的项目施加不同方式的切片方式,难以将其泛化.

赵玉丽等人[25]、王莹等人[26]通过复杂网络设计出基于节点重要度和测试桩复杂度的破环方法. 该方法根据类间调用关系及类自身的复杂度,定义出类的节点重要指标即复杂因子和影响因子,据此来设定节点重要值. 王莹等人[26]引入HITS[27]算法计算类的重要性,并根据类的重要性和测试桩复杂度得到最终的测试序列. 但是该方法得到的类集成测试序列总体测试桩复杂度偏高,相比其他方法,优势不明显.

Czibula等人[3]以考虑特定测试桩个数为评价指标,采用强化学习生成了类集成测试序列,且生成的序列所需测试桩个数在某些案例中少于传统方法. 该方法将类间方法依赖和属性依赖作为强化学习中的环境,智能体通过与环境的交互,得出反馈以达到训练智能体的目的,训练的最终结果即智能体得出最优类集成测试序列. 传统强化学习在解决类数量少的项目时,可以为其创建好的Q表格. 但随着类数量增多,这将增大巨大的内存消耗.

通过以上分析,本文针对Czibula等人[3]的强化学习方法随着类数量增多导致增大巨大的内存消耗这一问题,引入深度强化学习进一步研究. 其中,Advantage Actor-Critic(A2C)方法[28]是深度强化学习中的一个典型算法,已经被成功用于解决各种生活中遇到的难题,包括自动驾驶汽车[29]、游戏竞技[30]以及在2016年特别热门的Alpha Go[31]. 因此本文选用A2C用于生成类集成测试序列. 首先将智能体置于由不同类之间的关系构成的环境中,通过不断与环境交互,进而得出最小测试桩复杂度的路径,即最终的类集成测试序列.

3　本文方法

本节引用强化学习[32]中的深度强化学习[33]中的A2C[34]方法,用于解决类集成测试序列生成问题. 为了更合理地理解整个解决问题的流程,首先给出本文方法的框架,如图1所示.

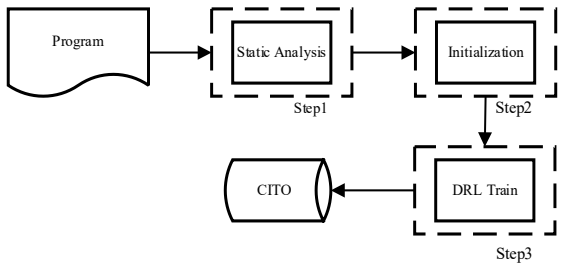


图1　基于DRL的CITO解决方法框架

其中Step1是对项目进行静态分析,得到类间的依赖关系,这些依赖关系包括方法依赖和属性依赖,再根据不同依赖占比得出耦合值. Step2是将第一步得出的耦合值加载到深度强化学习中的环境中,为智能体在不同状态下与环境交互的结果作为奖励依据. Step3是智能体在每轮与环境交互的同时训练神经网络. 最后根据智能体从初态至终态选择的不同动作作为类集成测试序列.

3.1　静态分析

类间依赖关系根据程序运行与否可以分为静态依赖关系与动态依赖关系. 本文主要通过Soot*对程序进行静态分析.

本节选用SIR**(Software-artifact Infrastructure Repository)中的elevator项目(电梯调度算法)进行举例说明. 首先对elevator中的类进行编号,如表1所示.

利用Soot对中的类分别进行类间属性依赖与类间方法依赖分析,如表2所示,表中已经省略值全为0的列和行. 表中括号左侧数据为属性依赖数,右侧为方法依赖数. 例如,在行编号为0、列编号为9的数值为(2,3)表示类0(Building)为类9(Person)提供了2个属

* http://www.sable.mcgill.ca/soot/

** http://sir.unl.edu/portal/index.html

表1 elevator 各类编号

类编号	类名称	类编号	类名称
0	Building	6	ElevatorState
1	DoorClosedException	7	Floor
2	Elevator	8	Logger
3	ElevatorController	9	Person
4	ElevatorFullException	10	PersonState
5	ElevatorMovingException	11	Simulator

表2 elevator 属性依赖和方法依赖

类编号	0	2	3	7	9	11
0	-	-	-	-	(2,3)	(0,2)
1	-	(0,2)	-	-	-	-
2	-	-	(4,33)	(2,2)	(2,9)	-
3	(1,7)	(2,1)	-	(2,2)	-	-
4	-	(0,1)	-	-	-	-
5	-	(0,2)	-	-	-	-
6	(1,2)	(1,0)	(1,0)	-	-	-
7	(2,2)	-	(1,15)	-	(1,10)	-
8	-	(1,24)	(1,27)	(1,9)	(1,4)	-
9	-	(2,1)	(2,2)	(3,9)	-	(0,4)
10	-	(0,2)	-	-	(1,2)	-

性依赖,3个方法依赖.

3.2 将数据加载到环境

通过静态分析得到各个类之间的属性依赖值与方法依赖值之后,即可通过式(1)以及式(2)来计算类间耦合值.

$$\mathrm{SCplx}(i,j)=\sqrt{W_A\cdot\overline{A(i,j)^2}+W_M\cdot\overline{M(i,j)^2}} \tag{1}$$

$$\overline{A(i,j)}=\frac{A(i,j)}{A_{\max}-A_{\min}} \tag{2}$$

$$\overline{M(i,j)}=\frac{M(i,j)}{M_{\max}-M_{\min}} \tag{3}$$

其中 $\mathrm{SCplx}(i,j)$ 是指序号为 i 的类与序号为 j 的类之间的耦合复杂度,W_A 和 W_M 分别为属性所占权值和方法所占权值,$A(i,j)$ 与 $M(i,j)$ 分别是两个类之间的属性复杂度和方法复杂度.将 $\mathrm{SCplx}(i,j)$ 的值赋值到 CostMatrix(代价耦合矩阵)中,简称为 c,即 $c[i,j]=\mathrm{SCplx}(i,j)$. 这个矩阵将作为深度神经网络中智能体每次交互所得奖励的重要依据. 同样以 elevator 为样例来演示代价耦合矩阵,此时计算得到的 $\mathrm{SCplx}(i,j)$ 并不会直接作为深度强化学习的奖励计算,而是在此基础上乘一个常数 c,该数值是深度强化学习里的超参数(需要不断尝试得出较合理的取值).

该代价耦合矩阵是根据式(1)、式(2)、式(3)和表2计算得出,其意义表示某一个类为另一个类构建的测试桩复杂度值. 例如在行编号为0、列编号为9的数值为0.359 35,表示类0(Building)为类9(Person)构建的类测试桩复杂度为0.359 35.

表3 elevator 耦合代价矩阵

类编号	0	2	3	7	9	11
0	0	0	0	0	0.359	0.043
1	0	0.043	0	0	0	0
2	0	0	1	0.356	0.403	0
3	0.232	0.354	0	0.356	0	0
4	0	0.021	0	0	0	0
5	0	0.043	0	0	0	0
6	0.177	0.182	0.177	0	0	0
7	0.356	0	0.367	0	0.278	0
8	0	0.544	0.605	0.262	0.197	0
9	0	0.354	0.356	0.564	0	0.086

3.3 A2C方法的运行

由于深度强化学习与类集成测试序列生成问题是两个领域内的研究方向,尚未有人将两者进行结合,因此本节内容主要阐述两者结合的方式.

3.3.1 状态的描述

在将DRL运用到CITO问题之前,需要建立对CITO问题的状态描述. 每一轮动作选择过程的状态转换包含以下五个步骤:

步骤1 根据项目的类个数创建等长的列表,列表中的每一个值表示对应类的测试次序,整个列表表示所有类的测试次序,初始值赋为[−1, −1, −1, −1].

步骤2 定义一个累加器count,表示已经集成的类个数,初始值赋为0.

步骤3 当选择某一个类加入到序列后,将类编号对应的测试次序置为count.

步骤4 重复执行步骤3,当count值等于类的个数时,停止执行步骤3.

步骤5 将测试次序与对应的类编号位置进行交换,再依据测试次序值由小到大排序进而得出类编号顺序,即为类集成测试序列.

以如图2所示的包含4个类的项目为例,对CITO问题状态的描述如下:

首先,在初始状态下,根据步骤1,将列表内测试次序均赋值为−1,并设置四个类编号分别为0,1,2,3,再根据步骤2,定义一个累加器count并赋初值为0;其次,假如选择的第一个类编号(动作)为2,根据步骤3,count值自增到1,将类编号2对应的测试次序赋值为当前count值,此时测试次序为[−1, −1, 1, −1];然后,根据步骤4,继续选择类编号,假如依次选择的类编号为3,1,0,这时count值等于类的个数,停止本轮类编号的选择,四个类[0, 1, 2, 3]对应的测试次序为[4, 3, 1, 2];最后,根据步骤5,将类编号与对应的测试次序交换,并对测试次序由小到大进行排序,得出对应的类编号分别为2,3,

1,0,即,本轮的类集成测试序列为[2,3,1,0].

其中,在执行步骤3时,还需要对类测试次序进行判断,如果不是-1,表示已经选过此类,若再次选择该类时,则会导致重选,即里面的数就不能置为count了,而是要置为"count+类个数",该数值用于判断当前序列是否为候选序列(即类集成测试序列不含重复的类). 在执行步骤5时,首先要对其判断,取列表中的最大值,如果最大值超过类个数,表示必定存在某一个类被选择超过一次,则该序列就不是候选序列,反之可以作为候选序列.

3.3.2　动作描述及动作选择机制

动作是指智能体每次于环境交互之前,要执行的操作. 在生成类集成测试序列中,动作是被看作一个类编号,其意义可以视为,在这一步,智能体将此编号的类集成到测试序列中,并据此与环境交互.

动作选择机制是指智能体选择动作的准则,目前常见的是ϵ-greedy[35]准则:

$$\pi(a|s)=\begin{cases}\epsilon/m+1-\epsilon, & \text{if } a^*=\arg\max\limits_{a\in A} Q(s,a)\\ \epsilon/m, & \text{else}\end{cases} \tag{4}$$

其中,ϵ的数值是动态变化的,随着训练轮数的增加而降低,可以在训练后期减少探索,更快地收敛. 选择动作时,以$\epsilon/m+1-\epsilon$的概率选择价值最大的动作,m是指还未被集成到序列的类个数,a表示动作;以ϵ/m的概率随机选择其他类. 动作价值网络和价值网络在初始阶段得出的经验数据的参考价值不大,需要智能体不断地与环境交互并训练网络,使其产生的经验更有意义.

3.3.3　奖励机制

DRL应用在CITO中的奖励机制主要基于中国矿Zhang等人[35]提出的奖励机制. 首先需要引入以式(5)、式(6)以及式(7):

$$\mathrm{TC}(i)=\sum_{u\in S_{\text{untest}}} c_{iu}\,(i\neq u) \tag{5}$$

$$\mathrm{TR}(i)=\sum_{u\in S_{\text{untest}}} c_{ui}\,(u\neq i) \tag{6}$$

$$\mathrm{NR}(i)=\mathrm{TR}(i)-\mathrm{TC}(i) \tag{7}$$

$\mathrm{TC}(i)$表示将选择第i个类为动作时,所需要的代价,该类依赖其他类(特指未集成到序列的类)的耦合复杂度的累加,u表示未被集成到序列的类编号,S_{untest}表示未被测试的类集合,c是代价耦合矩阵,该矩阵已经在3.2节中完成对环境的加载. $\mathrm{TR}(i)$表示将第i个类作为动作,所能为其他未集成到序列的类所产生的价值. $\mathrm{NR}(i)$表示净收益,是第i个类集成到序列带来的收益减去带来的代价. 其数值可正可负.

另外,在本文中需要对净收益进行调整,将其数值增大1 000倍,因为原始的净收益值相较于在强化学习达到终态的奖励值显得十分小. 因此要对其进行适当的放大,倍数1 000也是经过不断的尝试后得到的较理想的放大倍数.

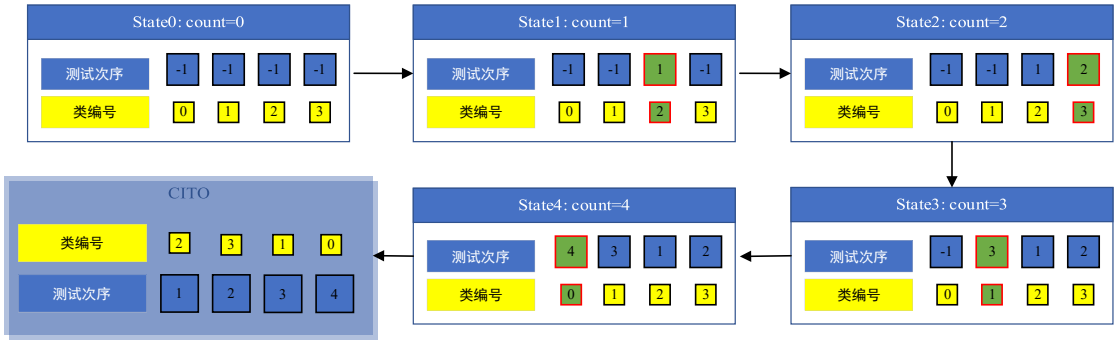


图2　DRL应用到CITO问题的过程图

3.3.4　优先经验回放与神经网络更新优化

智能体在每次选择动作与环境交互都会产生一个经验,这个经验就是神经网络的更新时所需要用到的数据. 智能体从状态S_t时选择动作A_t,与环境交互之后得到奖励值R_t以及下一个动作S_{t+1},其中将$[S_t,A_t,R_t,S_{t+1}]$作为一条经验存储到经验池中,以作为神经网络的更新依据.

在更新过程中,为了避免自举带来的问题,本文选择"Double DQN[35]+TD[36]"的方式进行网络更新:

步骤1　从经验池中获取$[S_t,A_t,R_t,S_{t+1}]$.

步骤2　将S_t作为current_model的输入,并得出$V(S_t)$和$\pi(S_t)$.

步骤3　将S_{t+1}作为target_model的输入,并得出$V(S_{t+1})$.

步骤4　计算误差$\delta_t=V(S_{t+1})-V(S_t)+R_t$.

步骤5　计算$-\log\left(\pi(S_t)\right)*\delta_t$.

步骤6　$\mathrm{loss}=\delta_t-\log\left(\pi(S_t)\right)*\delta_t$.

Step7:反向传播,更新策略网络和价值网络.

另外,对于不同的经验,应当赋予不同等级的优先级[37],在一条经验中,δ_t值越大,S_t和S_{t+1}差异越大,当前的经验更加值得学习,应当赋予相对高的优先级. 实践证明,通过添加优先经验回放机制会使训练收敛的速度加快. 由于每次执行动作状态必定会改变,因

此不必采用m-step TD[36]算法,而选用单步时分算法.

3.4 算法流程

根据3.1节至3.3节的描述,我们设计了如下算法用于生成类集成测试序列,如算法1所示.

算法1 基于A2C的CITO算法

输入: 类间属性依赖Attr_Deps, 类间方法依赖Meth_Deps, 迭代次数T, 状态特征维数n, 动作集合A, 折扣因子γ, 当前网络Q, 目标网络Q', 批量下降个数m, 价值网络学习速率α, 策略学习速率β, 目标网络更新频率c, 经验池H, 批处理大小batch_size .

输出: 测试顺序的最低成本mincost, 类集成测试序列.

构建代价耦合矩阵costMatrix, 初始化经验池H

随机初始化当前Q网络参数θ, 以及目标网络Q'的参数$\theta^{-}=\theta$;

FOR episode = 1, ***T*** **DO**

将costMatrix加载到环境中

初始化状态s_0 , done = False

初始化环境 env

WHILE NOT done **DO**

将$\phi(s_t)$作为Q网络的输入

得到其对应所有动作的Advantage值和对s_t的评价

以ϵ-greedy算法选取一个随机动作a_t

在状态s_t下执行动作a_t得到s_{t+1},r_t, done

将$\{\phi(s_t),a_t,r_t,\phi(s_{t+1}),\text{don}a_i\}$存入经验池中

IF $t\geqslant$ batch_size **THEN**

从H中抽取batch_size个样本$\{\phi(s_i),a_i,r_i,\phi(s_{i+1}),\text{done}_i\}$

计算出当前网络的Q值y_i和TD_error:

$y_i=r_i+\gamma V(s_{i+1};w)-\text{mean(advantage)}$

$\delta_i=V(s_{i+1};w)-y_i$

更新策略网络$\pi(A,S;\theta)$(Actor):

$$\theta\leftarrow\theta-\beta\cdot\delta_i\cdot\frac{\partial\ln(a_i|s_i;\theta)}{\partial\theta}$$

更新价值网络$V(S;w)$(Critic):

$$w\leftarrow w-\alpha\cdot\delta_i\cdot\frac{\partial V(s_i;w)}{\partial\theta}$$

IF $i\%C==1$ **THEN**

更新目标网络参数$w'=w$

END IF

IF done == True **THEN**

记录智能体动作轨迹以及总体测试桩复杂度

break;

END IF

END WHILE

IF cost < mincost **THEN**

mincost = cost

mincost_order = order

END IF

END FOR

算法具体步骤如下:

步骤1 根据项目中类间方法依赖和属性依赖,构建代价耦合矩阵,见算法第1行.

步骤2 开始进行循环训练,将类间耦合矩阵加载到环境. 将初始状态的向量作为网络的输入,根据贪婪算法选择动作,并与环境交互将返回结果作为经验保存在经验池中,智能体由下一状态的方式继续与环境交互.

步骤3 从经验池中取出m个经验作为网络训练的数据. 分别对Actor网络和Critic网络进行不同方式的训练和更新.

步骤4 如果当前智能体状态是终态,则结束本轮训练,更新最小类集成测试序列复杂度值,并进入下一轮.

步骤5 输出最小类集成测试桩复杂度及对应序列.

4 实验评估

4.1 实验对象

本文选取ANT, ATM, SPM, DNS, daisy, email-spl和BCEL 7个项目作为实验对象,其中ANT, ATM, SPM, DNS是Briand等人[38]使用的基准系统,其余3个是SIR(http://sir.unl.edu)的开源系统. 实验对象的详细信息如表5所示. 其中,3至6列分别为项目中类个数、依赖个数、环路个数、代码行数.

表5 项目详细信息

System	Description	Classes	Deps	Cycles	LOC
ANT	部署工具	25	83	654	4 093
ATM	自动柜员机系统	21	67	30	1 390
SPM	安全巡逻监视项目	19	72	1 178	1 198
DNS	域名解析系统	61	276	16	6 710
daisy	网络文件系统	23	36	4	1 148
email-spl	电子邮件工具	39	64	38	2 276
BCEL	创建、分析、操作Java类文件的插桩工具	45	294	416 091	3 033

4.2 实验环境

本文方法运行在Ubuntu系统下,包含4个CPU、1个GPU以及24G运行内存,软件方面采用python3.6和Pytorch1.7. 除了设置深度强化学习算法需要设置的参数之外,还需要设置有关CITO复杂度计算上的一些参数设置,为了保证对比实验的有效性,本文方法中CITO部分的参数设置与其他算法[2,3,5,6,35,39]设置值相同,如表6所示.

表6　参数设置

参数	含义	值	参数	意义	值
W_A	属性权重	0.5	ε-start	ε初始值	1
W_M	方法权重	0.5	ε-end	ε结束值	0.01
γ	折扣率	0.8	T	训练次数	200 000
c	奖励倍数	10 000	batch_size	批处理单位	64
min_value	最大惩罚	−150 000	ε_decay	ε降低速率	1 000×N
max_value	最大奖励	150 000	-	-	-

由3.2节可知,智能体的动作奖励值是依据代价耦合矩阵中的数,但是其真实数值十分小,在运行过程中不可以直接将此作为动作奖励值,因此本文在原数值基础上乘以倍数c;表格中min_value和max_value分别对应当智能体选择动作失败(非候选序列)和最终完成(是候选类)所对应的奖励值.ε-start是ε开始的值,ε-end是ε结束时的值,由于程序运行前期和后期所侧重的点不同,ε作为动作选择机制的重要指标也应当是变化的.智能体前期主要是为了探索尽可能多的道路,因此较大的ε值可用于探索;智能体由于前期已经探索了足够的道路,策略网络可以精准选出价值最大的动作,价值网络可以合理评估智能体动作的好坏,降低ε值可以更合理地使网络收敛.batch_size是神经网络每次更新时所需要的经验个数.

4.3　实验结果分析

首先以测试序列总体测试桩复杂度作为评价指标.我们对每个项目进行了10次实验,得出基于深度强化学习得出的总体测试桩复杂度箱型图如图3所示.

在这一部分,我们分别选择了第2节中的基于图论、基于启发式和基于RL这三类方法中的每一类方法中的部分典型方法与本文方法进行比较.其中,基于图论的方法选择Le Traon等人[6]的方法、Tai等人[5]的方法和Briand等人[38]的方法.Le Traon等人[6]的方法破环的过程中删除权重最大的节点入度边;Tai等人[5]的方法根据强弱连接关系构建了主次顺序,对每条边赋值为起点与终点之和后按照权值由大到小逐一破环;Briand

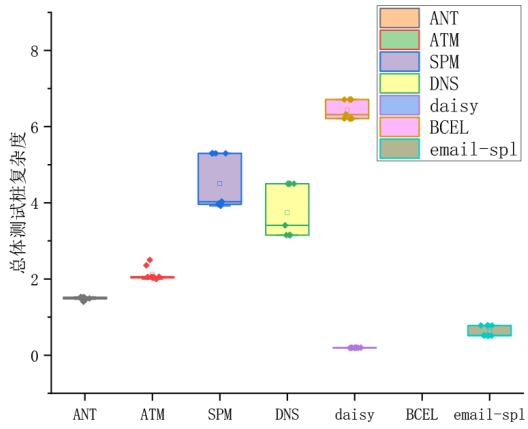


图3　总体测试桩复杂度箱型图

等人[38]的方法则是建立不同的连通分量,再依次破环.基于启发式的方法选择遗传算法GA[8]、随机交互算法RIA[16,17]和粒子群算法[39].遗传算法GA[8]对遗传算子进行一系列的交叉、变异等操作不断生成新的个体,以找到最佳的类集成测试序列;随机交互算法RIA[16,17]利用破除类间关系的代价,依据总体测试桩复杂度最小原则生成类集成测试序列;粒子群算法[39]PSO将测试序列映射为粒子群中的粒子,通过适应度函数确定粒子的速度和位置,进而得到最优的位置和适应度,再由映射关系反推出类集成测试序列.基于RL的方法[3]是使用了最原始的强化学习,通过类间关系搭建环境,智能体与环境不断交互,最终得出总体测试桩复杂度最低的类集成测试序列.

比较结果如表7和表8所示(由于在RL的文献[3]中并没有给出总体测试桩复杂度,因此这里不对其进行比较).在ANT,ATM,daisy,email-spl,BCEL项目中,本文采用的DRL方法所得结果平均总体测试桩复杂度分别1.4795,2.052,0.1928,0.52,7.1052,明显优于其他7种方法.在SPM项目中,本文的方法略高于RIA和PSO的方法;在DNS项目中,本文的方法略高于Briand的方法.在这两种项目中,深度强化学习方法表现不比传统

表7　总体测试桩复杂度对比表

Category	Method	ANT	ATM	SPM	DNS	daisy	email-spl	BCEL
Graph	Le Traon	3.72	3.37	8.4	5.02	-	-	8.23
	Tai	3.87	2.99	8.08	4.63	-	-	8.68
	Briand	3.31	2.7	5.82	**1.51**	-	-	8.58
Heuristic	GA	3.64	2.68	5.77	7.15	2.63	1.56	13.70
	RIA	2.32	2.39	3.25	8.56	0.33	0.95	7.95
	PSO	2.25	2.53	**3.01**	5.82	0.90	1.04	8.59
RL	DRL	**1.479 5**	**2.052**	4.028 3	3.409 1	**0.192 8**	**0.52**	**7.105 1**

表8 特定测试桩个数对比表

Category	Method	ANT	ATM	SPM	DNS	daisy	email-spl	BCEL
Graph	Le Traon	18	9.1	24.7	11	-	-	67.5
	Tai	28	8	20.2	27	-	-	128
	Briand	**11**	**7**	17	**6**	-	-	70
Heuristic	GA	12	**7**	17	**6**	12	27	71
	RIA	14.97	7.43	12.77	20.07	6	15	**53.60**
	PSO	14.33	7.83	**12.07**	17.33	8	19	55.47
RL	RL	19.33	7.33	21.33	14.33	4.13	**11.07**	88.33
	DRL	19	18	19	32	**4**	13	93

方法优秀，这是由于在这两个项目各自的类所创建的环境中，智能体陷入了局部最优．另外，由于Le Traon等人[6]的方法、Tai等人[5]的方法、Briand等人[38]的方法没有针对daisy和email-spl这两个项目进行实验研究，因此未进行对比．

图4为不同方法下各项目总体测试桩复杂度比较图，横坐标依次表示7个系统，纵坐标表示不同的解决方案花费的总体测试桩复杂度，其中，最后一列是本文采用的基于DRL方法．可以得出在生成类集成测试序列问题上，本文采用基于深度强化学习的方法在一定程度上优于现有其他算法．

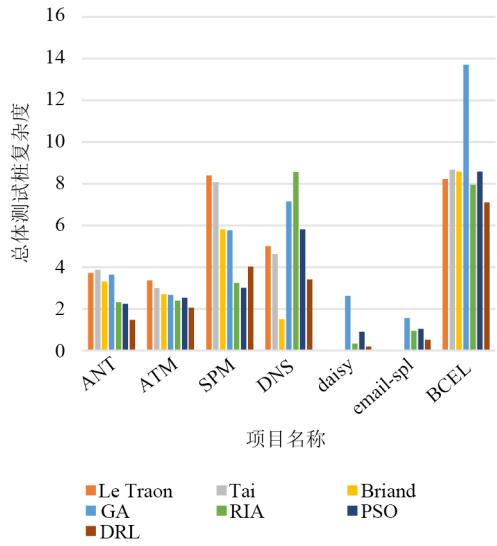


图4 总体测试桩复杂度对比图

本文方法在生成类集成测试序列时仅用测试桩复杂度来衡量测试代价，未考虑测试桩个数．但是，为方便后续研究人员对比，我们仍然统计了测试桩个数．且大部分现有算法在计算测试桩个数时，往往只统计了特定测试桩个数，忽略了通用测试桩个数．因此，此处也针对特定测试桩个数的结果进行进一步对比．

由表8可以得出本文所采用的基于DRL的方法在特定测试桩个数上优势不明显，主要原因有如下几条：(1)本文的方法目标侧重于总体测试桩复杂度的降低，并未考虑测试桩个数；(2)由于仅仅考虑总体测试桩复杂度，因此在构建特定测试桩时，方法依赖和属性依赖少的类会优先被智能体考虑为下一个选择的动作，而方法依赖与属性依赖个数较多的特定测试桩会被多个小的特定测试桩所代替．综上，在以特定测试桩个数作为测试序列优劣的衡量指标时，基于DRL的方法表现并不突出．

本文方法经过10次实验后得到的平均通用测试桩个数如表9所示，以备其他研究人员使用．

表9 通用测试桩个数

	ANT	ATM	SPM	DNS	daisy	email-spl	BCEL
DRL	11.33	14	17.33	22.5	4	12	18.34

由于我们实验了多次，这里仅给出其中一个结果作为演示，各个项目的通用测试桩具体信息如表10所示，其中，第一列是项目名称，第二列是所需的通用测试桩的类编号以及类名称．

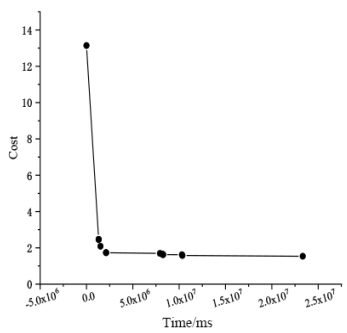
4.4 收敛速度

本文对所选系统进行分析，根据网络训练时间的推进，网络得出的总体测试桩复杂度会趋于平缓．如图5所示．

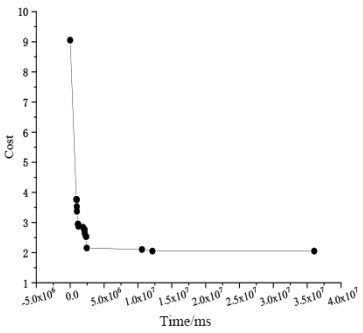
由于使用的是神经网络训练，因此其时间上的消耗与其他方法没有可比度．本文的方法训练的神经网络耗时长，传统方法在1 000 ms以内，但本文方法最终得出结果一般在10^8 ms以上，但是所得到的弥补结果是，类总体测试桩复杂度相对于其他方法较低．

表 10　通用测试桩详细信息

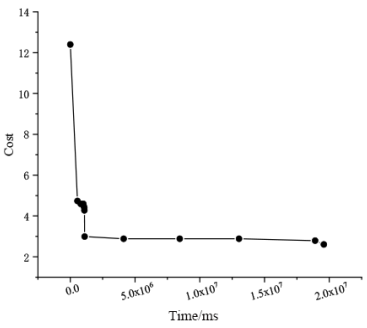
项目名称	通用测试桩信息
ANT	1:BuildEvent,3:BuildListener,14:PathTokenizer,16:ProjectComponent,19:Target,20:Task,21:TaskAdapter,17:ProjectHelper,22:TaskContainer,23:UnknownElement,4:BuildLogger
ATM	0:ReceiptPrinter,1:Display,2:Keyboard,3:CardReader,4:OperatorPanel,5:EnvelopeAcceptor,6:CashDispenser,8:Bank,9:Session,10:Transaction,11:WithdrawlTransaction,12:DepositTransaction,13:TransferTransaction,14:InquiryTransaction
SPM	3:CheckpointControllerIdleState,4:CheckpointControllerNormalCheckingState,5:CheckpointControllerState,6:CheckpointControllerViolationCheckingState,7:CheckpointControllerViolationState,8:CheckpointReferences,0:CardReaderInterface,12:DoorLockInterface,13:GuardDueTimer,14:GuardLateTimer,15:PatrolSchedule,18:SecurityZone
DNS	4:BitString,7:Compression,10:DNAMERecord,27:Options,48:WireParseException,31:Record,54:Message,60:ResolverListener,6:CNAMERecord,34:RRset,43:TTL,9:DClass,14:Flags,26:Opcode,28:OPTRecord,42:TSIGRecord,52:Header,47:UNKRecord,11:dns,55:MX_KXRecord,37:SimpleResolver,51:ExtendedResolver,21:NameSet
Daisy	17:util.Debug,1:daisy.Daisy,21:util.SplitPrintStream,14:DaisyUserThread
email-spl	2:defpackage.PL_Interface,3:defpackage.PL_Interface_impl,8:EmailSystem.Client,24:runspl.RunSPL,16:featuremodel.FeatureID,14:featuremodel.Configuration,21:featuremodel.RelevantModelIterator,7:EmailSystem.ClientKeyringEntry,36:TestSpecifications.SpecificationManager,13:featuremodel.Configuration1,34:TestSpecifications.Specification9
BCEL	7:ConstantCP,44:Visitor,5:Constant,16:ConstantObject,17:ConstantPool,1:Attribute,35:Node,10:ConstantFloat,11:ConstantInteger,12:ConstantInterfaceMethodref,15:ConstantNameAndType,0:AccessFlags,25:Field,34:Method,20:ConstantValue,21:Deprecated,36:PMGClass,37:Signature



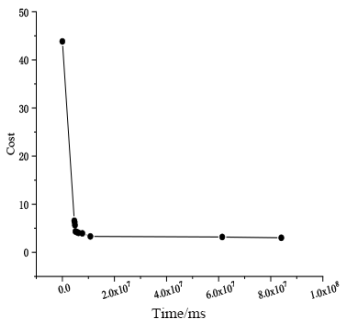
(*a*) ANT总体测试桩复杂度变化图



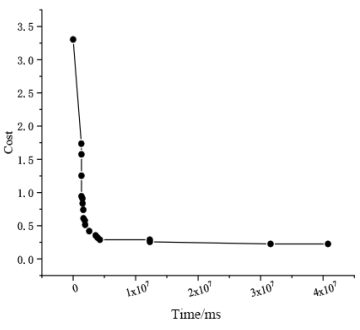
(*b*) ATM总体测试桩复杂度变化图



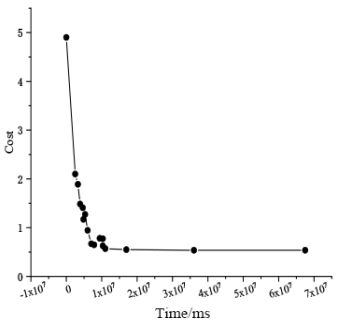
(*c*) SPM总体测试桩复杂度变化图



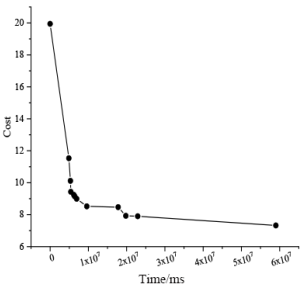
(*d*) DNS总体测试桩复杂度变化图



(*e*) daisy总体测试桩复杂度变化图



(*f*) email-spl总体测试桩复杂度变化图



(*g*) BCEL总体测试桩复杂度变化图

图5　总体测试桩复杂度变化图

5 结束语

本文采用了一种基于深度强化学习的类集成测试序列生成方法,该方法综合了强化学习与神经网络的相关知识,通过智能体与环境的不断交互,依次记录智能体的动作,生成类集成测试序列. 与其他方法生成的类集成测试序列所花费的总体测试桩复杂度相比较,本文的方法在某些系统上有效地降低了总体测试桩复杂度.

此外,本文的方案仍有可以改进的地方,在DNS以及BCEL这种项目中类个数多的情况下,深度强化学习由于维度较高,常常陷入局部最优的情况,这也是日后可以进一步改进的入口. 在下一步的研究中,可以将特定测试桩个数作为智能体在与环境交互时的一个奖惩依据,而不是完全侧重于总体测试桩复杂度上. 而对于时间上的消耗,目前还未有可以改进的方案,即使是用了时间更短的训练方式,其时间消耗相比较以往的方法也是较大的. 且本文方法涉及许多超参数,不同的超参数设置会有差异较大的结果,其中某些参数的设置将会在未来的工作中探索.

参考文献

[1] KUNG D, GAO J, HSIA P, et al. A test strategy for object-oriented programs[C]//Proceedings of the 19th Annual International Computer Software and Applications Conference. Texas: IEEE, 1995: 239-244.

[2] BRIAND L C, FENG J, LABICHE Y. Using genetic algorithms and coupling measures to devise optimal integration test orders[C]//Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering. Ischia: Association for Computing Machinery, 2002: 43-50.

[3] CZIBULA G, CZIBULA I G, MARIAN Z. An effective approach for determining the class integration test order using reinforcement learning[J]. Applied Soft Computing, 2018, 65: 517-530.

[4] KUNG D C, GAO J, PEI H, et al. On regression testing of object-oriented programs[J]. Journal of Systems and Software, 2015, 32(1): 21-40.

[5] TAI K C, DANIELS F J. Test order for inter-class integration testing of object-oriented software[C]//Proceedings of the 21st Annual International Computer Software and Applications Conference. Washington: IEEE, 1997: 602-607.

[6] LE TRAON Y, JÉRON T, JÉZÉQUEL J M, et al. Efficient object-oriented integration and regression testing[J]. IEEE Transactions on Reliability, 2000, 49(1): 12-25.

[7] BRIAND L C, LABICHE Y, WANG Y. An investigation of graph-based class integration test order strategies[J]. IEEE Transactions on Software Engineering, 2003, 29(7): 594-607.

[8] LE HANH V, AKIF K, LE TRAON Y, et al. Selecting an efficient OO integration testing strategy: An experimental comparison of actual strategies[C]//Proceedings of the European Conference on Object-Oriented Programming. Heidelberg: Springer, 2001: 381-401.

[9] ZAIDMAN A, DEMEYER S. Automatic identification of key classes in a software system using webmining techniques[J]. Journal of Software Maintenance and Evolution: Research and Practice, 2008, 20(6): 387-417.

[10] HASHIM N L, SCHMIDT H W, RAMAKRISHNAN S. Test order for class-based integration testing of java applications[C]//Proceedings of the 5th International Conference on Quality Software. Victoria: IEEE, 2005: 11-18.

[11] 张艳梅, 姜淑娟, 张红昌. 一种基于动态依赖关系的类集成测试方法[J]. 计算机学报, 2011(6): 1075-1089.
ZHANG Y M, JIANG S J, ZHANG H C. A method of class integration testing based on dynamic dependency[J]. Chinese Journal of Computers, 2011(6): 1075-1089. (in Chinese)

[12] 张艳梅. 基于依赖性分析的面向对象程序测试技术研究[D]. 徐州: 中国矿业大学, 2012.
ZHANG Y M. Research on Testing Technology of Object-Oriented Programs Based on Dependency Analysis [D]. Xuzhou: China University of Mining and Technology, 2012. (in Chinese)

[13] CHEN Q, LI X. An order-assigned strategy of classes integration testing based on test level[C]//Proceedings of the 8th International Conference on Computer Supported Cooperative Work in Design. Heidelberg: Springer, 2004: 653-657.

[14] 郑磊. 面向对象集成测试的分层增量测试策略[D]. 上海: 上海交通大学, 2007.
ZHENG L. Layered and Incremental Strategy for Object-ed-Oriented Integration Testing[D]. Shanghai: Shanghai Jiaotong University, 2007. (in Chinese)

[15] BORNER L, PAECH B. Integration test order strategies to consider test focus and simulation effort[C]//Proceedings of the 1st International Conference on Advances in System Testing and Validation Lifecycle. Proto: IEEE, 2009: 80-85.

[16] WANG Z, LI B, WANG L, et al. Using coupling measure technique and random iterative algorithm for inter-class integration test order problem[C]//Proceedings of the

IEEE 34th Annual Computer Software and Applications Conference Workshops. Seoul: IEEE, 2010: 329-334.

[17] WANG Z, LI B, WANG L, et al. A brief survey on automatic integration test order generation[C]//Proceedings of the 23rd International Conference on Software Engineering and Knowledge Engineering. Miami Beach: ACM, 2011: 254-257.

[18] VERGILIO S R, POZO A, ÁRIAS J C G, et al. Multi-objective optimization algorithms applied to the class integration and test order problem[J]. International Journal on Software Tools for Technology Transfer, 2012, 14(4): 461-475.

[19] ASSUNÇÃO W K G, COLANZI T E, POZO A T R, et al. Establishing integration test orders of classes with several coupling measures[C]//Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation. Dublin: ACM, 2011: 1867-1874.

[20] DEB K, AGRAWAL S, PRATAP A, et al. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II[C]// Proceedings of the International Conference on Parallel Problem Solving From Nature. Berlin: Springer, 2000: 849-858.

[21] ZITZLER E, LAUMANNS M, THIELE L. SPEA2: Improving the strength Pareto evolutionary algorithm[J]. TIK-report, 2001: 103.

[22] KNOWLES J D, CORNE D W. Approximating the nondominated front using the pareto archived evolution strategy[J]. Evolutionary Computation, 2014, 8(2): 149-172.

[23] JAROENPIBOONKIT J, SUWANNASART T. Finding a test order using object-oriented slicing technique[C]//Proceedings of the 14th Asia-Pacific Software Engineering Conference. Aichi: IEEE Computer Society, 2007: 49-56.

[24] 刘颖莲. 面向对象软件集成测试策略研究[D]. 北京: 北京邮电大学, 2013.
LIU Y L. Research of Object-Oriented Software Integration Testing Strategy[D]. Beijing: Beijing University of Post and Telecommunications, 2013. (in Chinese)

[25] 赵玉丽, 王莹, 于海等. 基于复杂网络的类间集成测试序列生成方法[J]. 东北大学学报(自然科学版), 2015, 36(12): 1696-1700.
ZHAO Y L, WANG Y, YU H, et al. An inter-class integration test order generation method based on complex networks[J]. Journal of Northeastern University (Natural Science), 2015, 36(12): 1696-1700. (in Chinese)

[26] 王莹, 于海, 朱志良. 基于软件节点重要性的集成测试序列生成方法[J]. 计算机研究与发展, 2016, 53(3): 517-530.
WANG Y, YU H, ZHU Z L. A class integration test order method based on the node importance of software[J]. Journal of Computer Research and Development, 2016, 53(3): 517-530. (in Chinese)

[27] ZAIDMAN A, CALDERS T, DEMEYER S, et al. Applying webmining techniques to execution traces to support the program comprehension process[C]//Proceedings of the 9th European Conference on Software Maintenance and Reengineering. Manchester: IEEE Computer Society, 2005: 134-142.

[28] HESSEL M, MODAYIL J, HASSELT H V, et al. Rainbow: Combining improvements in deep reinforcement learning [EB/OL]. (2017-10-06) [2021-05-08]. https://arxiv. org/abs/1710.02298.

[29] 黄志清, 曲志伟, 张吉, 等. 基-于深度强化学习的端到端无人驾驶决策[J]. 电子学报, 2020, 48(9): 1711-1719.
HUANG Z Q, QU Z W, ZHANG J, et al. End-to-end autonomous driving decision based on deep reinforcement learning[J]. Acta Electronica Sinica, 2020, 48(9): 1711-1719. (in Chinese)

[30] 刘春阳, 谭应清, 柳长安, 等. 多智能体强化学习在足球机器人中的研究与应用[J]. 电子学报, 2010, 38(8): 1958-1962.
LIU C Y, TAN Y Q, LIU C A, et al. Application of multi-agent reinforcement learning in robot soccer[J]. Acta Electronica Sinica, 2010, 38(8): 1958-1962. (in Chinese)

[31] WANG F Y, ZHANG J J, ZHENG X, et al. Where does AlphaGo go: From church-turing thesis to AlphaGo thesis and beyond[J]. IEEE/CAA Journal of Automatica Sinica, 2016, 3(2): 113-120.

[32] QIANG W, ZHONG L Z. Reinforcement learning model, algorithms and its application[C]//Proceedings of the 2011 International Conference on Mechatronic Science. Jilin: IEEE, 2011: 1143-1146.

[33] VAN HASSELT H, GUEZ A, SILVER D. Deep reinforcement learning with double q-learning[C]//Proceedings of the 30th AAAI Conference on Artificial Intelligence. Phoenix: Arizona, 2016: 2094-2100.

[34] JANG H C, HUANG Y C, CHIU H A. A study on the effectiveness of A2C and A3C reinforcement learning in parking space search in urban areas problem[C]//Proceedings of the 2020 International Conference on Information and Communication Technology Convergence. Jeju Island: IEEE, 2020: 567-571.

[35] ZHANG M, JIANG S J, ZHANG Y M, et al. A multi-lev-

el feedback approach for the class integration and test order problem[J]. Journal of Systems and Software, 2017, 133: 54-67.

[36] TAN A H, LU N, XIAO D. Integrating temporal difference methods and self-organizing neural networks for reinforcement learning with delayed evaluative feedback[J]. IEEE Transactions on Neural Networks, 2008, 19(2): 230.

[37] SCHAUL T, QUAN J, ANTONOGLOU I, et al. Prioritized experience replay[EB/OL]. (2015-11-18)[2021-05-08]. https://arxiv.org/abs/1511.05952.

[38] BRIAND L C, LABICHE Y, WANG Y. Revisiting strategies for ordering class integration testing in the presence of dependency cycles[C]//Proceedings of the 12th International Symposium on Software Reliability Engineering. Hong Kong: IEEE, 2001: 287-296.

[39] 张悦宁, 姜淑娟, 张艳梅. 基于梦境粒子群优化的类集成测试序列生成方法[J]. 计算机科学, 2019, 46(2): 168-174.
ZHANG Y N, JIANG S J, ZHANG Y M. Approach for generating class integration test sequence based on dream particle swarm optimization algorithm[J]. Computer Science, 2019, 46(2): 168-174. (in Chinese)

作者简介

张颖辉 男,1996年生,河南商丘人. 中国矿业大学计算机科学与技术硕士研究生. 主要研究方向为类集成测试序列生成、深度强化学习.

张艳梅(通讯作者) 女,1982年生,唐山人. 博士,中国矿业大学计算机科学与技术学院副教授. 主要研究方向为软件分析与测试、强化学习.
E-mail: ymzhang@cumt.edu.cn

张志成 男,1998年生,湖北宜昌人. 南方科技大学工学院计算机科学与工程系硕士研究生. 主要研究方向为自动程序修复、移动应用分析与软件测试.

姜淑娟 女,1966年生,山东莱阳人. 博士,中国矿业大学教授、博士生导师. 主要研究方向为软件分析与测试、缺陷预测、故障定位、进化计算等.
E-mail: shjjiang@cumt.edu.cn

丁艳茹 女,1996年生,河北沧州人. 中国矿业大学计算机科学与技术学院博士研究生. 主要研究方向为软件分析与测试,强化学习.
E-mail: yrding@cumt.edu.cn

袁 冠 男,1982年生,江苏睢宁人. 博士,中国矿业大学教授、博士生导师. 主要研究方向为时空大数据技术以及计算智能等.
E-mail: yuanguan@cumt.edu.cn