软件学报 ISSN 1000-9825, CODEN RUXUEW
2025,36(7):3109−3133 [doi: 10.13328/j.cnki.jos.007340] [CSTR: 32375.14.jos.007340]

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

面向智能体路径规划算法的动态随机测试方法*

张逍怡[1], 李 幸[1], 刘 洋[2], 郑 征[3], 孙昌爱[1]

[1](北京科技大学 计算机科学与通信工程学院, 北京 100083)
[2](北京交通大学 机械与电子控制工程学院, 北京 100044)
[3](北京航空航天大学 自动化与电气工程学院, 北京 100191)
通信作者: 刘洋, E-mail: yangliu2@bjtu.edu.cn

摘 要: 智能体路径规划算法旨在规划某个智能体的行为轨迹, 使其在不碰到障碍物的情况下安全且高效地从起始点到达目标点. 目前智能体路径规划算法已经被广泛应用到各种重要的物理信息系统中, 因此在实际投入使用前对算法进行测试, 以评估其性能是否满足需求就非常重要. 然而, 作为路径规划算法的输入, 任务空间中威胁障碍物的分布形式复杂且多样. 此外, 路径规划算法在为每个测试用例规划路径时, 通常需要较高的运行代价. 为了提升路径规划算法的测试效率, 将动态随机测试思想引入到路径规划算法中, 提出了面向智能体路径规划算法的动态随机测试方法 (dynamic random testing approach for intelligent agent path planning algorithms, DRT-PP). 具体来说, DRT-PP 对路径规划任务空间进行离散划分, 并在每个子区域内引入威胁生成概率, 进而构建测试剖面, 该测试剖面可以作为测试策略在测试用例生成过程中使用. 此外, DRT-PP 在测试过程中通过动态调整测试剖面, 使其逐渐优化, 从而提升测试效率. 实验结果显示, 与随机测试及自适应随机测试相比, DRT-PP 方法能够在保证测试用例多样性的同时, 生成更多能够暴露被测算法性能缺陷的测试用例.

关键词: 软件测试; 路径规划算法; 动态随机测试; 快速扩展随机树生成算法; 测试剖面生成
中图法分类号: TP311

中文引用格式: 张逍怡, 李幸, 刘洋, 郑征, 孙昌爱. 面向智能体路径规划算法的动态随机测试方法. 软件学报, 2025, 36(7): 3109–3133. http://www.jos.org.cn/1000-9825/7340.htm

英文引用格式: Zhang XY, Li X, Liu Y, Zheng Z, Sun CA. Dynamic Random Testing Approach for Intelligent Agent Path Planning Algorithms. Ruan Jian Xue Bao/Journal of Software, 2025, 36(7): 3109–3133 (in Chinese). http://www.jos.org.cn/1000-9825/7340.htm

Dynamic Random Testing Approach for Intelligent Agent Path Planning Algorithms

ZHANG Xiao-Yi[1], LI Xing[1], LIU Yang[2], ZHENG Zheng[3], SUN Chang-Ai[1]

[1](School of Computer and Communication Engineering, University of Science and Technology Beijing, Beijing 100083, China)
[2](School of Mechanical, Electronic and Control Engineering, Beijing Jiaotong University, Beijing 100044, China)
[3](School of Automation Science and Electronic Engineering, Beihang University, Beijing 100191, China)

Abstract: Path planning algorithms for intelligent agents are designed to plan the behavior trajectory of an agent so that it can safely and efficiently reach the target point from the starting point without colliding with obstacles. Currently, path planning algorithms have been widely applied in various critical cyber-physical systems. Therefore, it is essential that the path planning algorithms be tested before being put into use to evaluate whether their performance can meet the requirements. However, the distribution patterns of threat obstacles in the task space, which are the inputs of the path planning algorithm, are complex and diverse. Moreover, a relatively high operational cost is usually required when the path planning algorithm plans a path for each test case. To improve the testing efficiency of the path planning

* 基金项目: 国家自然科学基金 (62302035, 62403044, 62372021); 中央高校基本科研业务费专项资金 (2023JBMC017, FRF-IDRY-23-016)
本文由“新兴软件与系统的可信赖性与安全”专题特约编辑向剑文教授、陈厅教授、杨珉教授、周俊伟教授推荐.
收稿时间: 2024-08-26; 修改时间: 2024-10-15; 采用时间: 2024-11-25; jos 在线出版时间: 2024-12-10
CNKI 网络首发时间: 2025-04-17

algorithms, this study adapts the concept of dynamic random testing into path planning algorithms and proposes the dynamic random testing approach for intelligent agent path planning algorithms (DRT-PP). Specifically, DRT-PP discretely divides the path planning task space and introduces the threat generation probability within each sub-region, thus constructing the test profile. This test profile can be used as a testing strategy in the process of test case generation. Furthermore, the test profile is dynamically adjusted by DRT-PP during the testing process to make it gradually optimized, thereby improving the testing efficiency. Experimental results show that, compared with random testing and adaptive random testing, the DRT-PP approach can not only ensure the diversity of test cases but also generate more test cases that can expose the performance defects of the tested algorithm.

Key words: software testing; path planning algorithm; dynamic random testing (DRT); rapidly-exploring random tree (RRT); test profile generation

路径规划算法旨在规划某个智能体的行为轨迹, 使其能够在不碰到威胁障碍物 (以下简称威胁) 的情况下安全且高效地从起始点到达目标点[1]. 目前, 智能体的路径规划算法 (path planning algorithm) 在工业界和日常生活的各种系统中被广泛应用, 如自动驾驶汽车的寻路[2]、无人机的航迹生成[3]、自动机器臂的行为规划[4]、智能仓库机器人物流[5]等各种任务都能够被转换为 (或部分转换为) 路径规划问题. 然而, 随着路径规划算法被应用在各种复杂且重要的任务 (如救援[6]、侦察[7]、运输[8]等) 中, 任何相关系统的失效都会造成巨大的经济损失. 因此, 在实际投入使用前对这些路径规划算法进行充分测试, 以确认该算法能够满足系统的性能需求就至关重要.

近 5 年学者们陆续开展面向路径规划算法测试的相关研究工作. 例如, 文献 [9] 基于场景的对称性以及生成路径的连通性等属性构建蜕变规则, 并提出蜕变测试方法. Li 等人[10]针对 UAV 内部的模块提出了蜕变测试方法. 这些研究大都基于路径规划的测试预期问题提出特定的解决方案或工具[11]. 总体来说, 相关研究仍处于起步阶段, 尚未发现有学者针对智能体的路径规划算法提出完整的动态测试方法, 而规划场景的复杂性以及规划算法的不确定性为路径规划算法带来了巨大挑战, 具体表现为以下两个方面.

(1) 场景的复杂性. 实际的路径规划场景十分复杂[12], 场景中的威胁 (如建筑物、极端天气等) 可能存在多样的分布. 通常情况下, 路径规划算法的失效表现为智能体按照算法生成的路径行动时做出了一系列错误决策, 导致相关系统功能或性能需求无法得到满足, 而这种错误的决策序列通常是由多个威胁的特定分布模式所触发的. 例如, 算法生成的路径会倾向于进入由多个威胁共同构成的“陷阱”中[13], 从而导致生成路径的总长度增加, 进而违反性能需求, 而这种特定的威胁分布模式很难通过传统随机测试[14]方法找到.

(2) 规划算法的不确定性. 另一方面, 现实任务场景 (如机器人、无人机救援等) 通常会对算法有较高的性能需求[15], 即需要智能体在短时间内规划出可行且质量较高的路径. 为了同时照顾计算效率及生成路径的质量, 当前应用最广泛的路径规划算法大多通过对地图的随机采样来快速寻找规划问题的可行解, 例如本文所讨论的快速扩展随机树算法 (rapidly-exploring random tree, RRT) 就通过在安全区域随机采点来生成一棵覆盖整个任务空间的随机树, 然后再通过从目标叶子节点向根节点进行回溯来构建一条安全的路径[16,17]. 然而这一类算法都具有一定程度的不确定性: 即使是任务场景相同, 算法生成的路径也会不同. 也就是说, 智能体在某个场景中完成了任务并不意味着其在相同场景中总会做出正确的决策, 即我们无法通过现有执行结果直接判断算法在特定场景是否存在性能风险 (如是否路径过长). 因此, 如何制定合理的测试策略, 以发现路径规划算法的性能失效风险就成为一大挑战.

由于上述挑战的存在, 所采用的测试方法应具备以下两个特点.

(1) 测试方法应具备一定的随机性, 以保证生成测试用例的多样性. 无论所采用的测试方法是基于何种理论或是启发式策略, 其都需要具备探索整个输入空间的能力, 因为只有这样才能保证所有失效测试用例都有一定概率被生成, 即在某种程度上避免陷入局部最优.

(2) 测试方法应具备寻优能力, 即能够在测试过程中逐渐能够生成更有价值的测试用例. 一条低质量 (如长度过长等) 的路径通常是由多个威胁的特定分布模式导致的, 而这种威胁分布通常比较隐蔽且不易被发现. 这就要求测试方法能够根据当前的测试信息进行搜索, 以逐步找到能够暴露算法问题的威胁分布模式.

为了解决上述挑战, 本文提出面向智能体路径规划算法设计的测试方法, 以有效生成能够暴露被测算法潜在性能失效的测试用例. 具体来说, 本文借鉴了动态随机测试 (dynamic random testing, DRT)[18]的思想, 提出了面向智能体路径规划算法的动态随机测试方法 (dynamic random testing approach for intelligent agent path planning algorithms, DRT-PP). DRT 通过构建测试剖面, 将输入空间划分为若干区域, 各个区域具有不同的概率被选为测试用例生成区域. 因此, 测试剖面可以看成是一个基于输入空间的概率分布, 而 DRT 将测试剖面视为一个暂时的测试策略, 在测试过程中依照当前的测试剖面按概率从输入空间生成测试用例, 这样既可以保持测试的随机性又能在测试用例生成时具有一定的倾向性. 此外, DRT 引入反馈机制, 即在测试过程中实时对当前的测试信息进行提取和分析, 并根据这些测试信息动态地调整测试剖面, 使那些有价值测试用例的生成概率逐渐升高, 进而提升测试效率. 可见, DRT 的基本思想与智能体路径规划算法的测试需求相契合.

目前关于 DRT 的相关研究大致可以分为两个方面. 一方面是对 DRT 的相关理论进行分析, 例如, 文献 [18] 讨论了在 DRT 中引入聚类方法来划分子空间的可能性. 然而, 与模型测试、变异测试等测试思想类似, DRT 是较高层次的测试思想或方法论, 不能直接作为具体的测试方法使用. 另一方面, 由于被测软件种类繁多, 不同的被测对象可能拥有不同的输入空间, 其错误的触发机理也各不相同, 因此对于不同的被测对象, 我们都需要根据相应的领域知识, 重新设计完整的测试方法和测试策略. 而本文将 DRT 思想引入到路径规划算法中, 基于路径规划的领域知识对测试方法进行了如下设计.

(1) 测试剖面构建. 路径规划问题的输入是连续空间上障碍物的分布, 因此我们不易基于传统的等价类划分思想建测试剖面. 为此, 我们首先对规划的任务空间进行离散化, 将其划分为一个个格子, 每个格子被视作子区域; 接着在每个子区域内引入一个包含威胁的概率, 进而构建测试整个输入空间的测试剖面. 也就是说, 一个测试剖面就是被分成若干格子的空间以及每个子区域包含威胁的概率. 通过此方法构建的测试剖面既能保证随机的测试的执行, 且保证了测试剖面的连续性, 使我们能够进一步设计合理的测试剖面更新策略.

(2) 测试剖面更新. 评价路径规划算法的性能指标很多, 本文考虑路径长度方面的性能需求, 因为路径的长度是衡量路径质量的首要指标[19], 其同时可以间接反映任务的执行时间、耗油量等指标. 由于作为被测对象的 RRT 路径规划算法具有不确定性, 我们无法直接度量某场景下生成路径的质量. 为此, 本文借鉴蜕变测试 (metamorphic testing, MT) 思想[9,10], 通过对一个测试用例进行多次仿真得到多条生成路径并计算这些路径长度的方差来评估测试用例的失效度. 在该定义下, 失效度较大的测试用例一方面意味着被测算法在执行该测试用例时稳定性较差, 另一方面也说明该测试用例下生成的路径与最优路径差距较大 (实际上, 我们还可以根据其他性能需求定义各种评价指标 (如路径平滑性), 而这些指标理论上同样适用于 DRT-PP), 而这些现象都反映了算法在实际应用场景中的性能风险. 在此基础上, 我们设计了反馈机制来动态调整测试剖面: 当发现某个测试用例具有较大的生成路径方差时, 测试剖面中与该测试用例威胁分布相关的子区域包含威胁的概率就会增加; 反之, 相关子区域包含威胁概率减小.

由于目前尚未发现面向路径规划算法的完整的动态测试方法, 我们基于软件测试领域经典且高效的随机测试 (random testing, RT) 及自适应随机测试 (adaptive random testing, ART) 思想设计了的面向路径规划算法的 RT 及 ART 测试方法, 并将其作为基线与 DRT-PP 进行对比. 实验结果表明, DRT-PP 在保证测试多样性的同时能够生成更多有效的测试用例, 平均失效度提升了约 1.5 倍 (与 RT 和 ART 相比). 此外, DRT-PP 能够发现 RT 和 ART 无法找到的具有高失效度的测试用例. 总的来说, 本文的贡献如下.

(1) 将动态随机思想引入路径规划算法的测试中, 以高效地生成有价值的路径规划场景.

(2) 通过引入威胁生成概率矩阵来构建路径规划算法的测试剖面, 解决了场景空间划分问题.

(3) 设计了基于多次执行路径差异的测试评价指标以及测试剖面矩阵的更新策略, 解决了对失效测试用例的寻优问题.

本文第 1 节介绍智能体路径规划问题、作为本文被测对象的 RRT 路径规划算法以及与路径规划测试相关的

国内外研究进展. 第 2 节介绍 DRT-PP 测试方法, 包括基于测试剖面的构建及更新策略. 第 3 节对 DRT-PP 的有效性进行实验评估. 第 4 节对本研究进行风险分析. 最后在第 5 节对全文进行总结.

1 背景及相关工作

1.1 智能体路径规划问题的定义及描述

智能体的路径规划是机器人领域的核心问题之一. 最早对该问题的研究始于 20 世纪 70 年代[4], 研究者通过将其转化为在位形空间中搜索无碰撞点的问题来构造完整且连续的路径或运动序列. 目前路径规划问题已经在多个领域被广泛讨论, 而在应用到某个具体场景时, 通常需要考虑该场景的特定约束. 无论是自动驾驶汽车的寻路[2]、无人机的航迹生成[3]、自动机械臂的行为规划[4], 还是仓储机器人的物流调度[5], 这些典型任务场景均可转化为(或部分转化为) 路径规划问题。例如, 自动驾驶的感知约束[20]、无人机场景需考虑气动模型及参数[21]、智能物流场景需要考虑具体的物流任务约束 (如多次分配)[22], 而机械臂则需要考虑其自由度以及各自由度之间的运动约束[23]. 本文为了方法的通用性, 考虑最基本的智能体路径规划问题, 具体的表述如下.

定义 1 (环境). 定义智能体执行任务时的任务空间 M 为一个 $X\times Y$ 的连续二维空间, 该空间中的一个点 p 可以用其坐标 (x, y) 来表示. 令 $Obs=\{obs_1,obs_2,\ldots\}$ 为空间中的威胁障碍物 (以下简称“威胁”) 集合, 其中每个威胁 $obs\in Obs$ 都可以看成是任务空间 M 中的子区域, 即 $obs\subset M$. 我们假设任何该区域中的点 $p\in obs$ 都被该威胁占据. 任务空间 M 和威胁集合 Obs 共同组成了某次规划任务的环境, 记为 $E=\langle M,Obs\rangle$.

定义 2 (智能体). 我们将智能体 ag 定义为任务空间 M 中一个具备运动能力的质点. 假设在 τ 时刻该智能体的位置为 $p_\tau=(x,y)$, 则在下一时刻, 即 $\tau+1$ 时刻, 其位置可以是 $p_{\tau+1}=(x+\delta^x,x+\delta^y)$, 满足 $(\delta^x)^2+(\delta^y)^2=r^2$, 其中, $r>0$ 表示智能体在单位时间内的行动步长, 即智能体在单位时间内移动的距离.

为了简化问题, 本文采用固定的智能体行动步长, 这也是经典 RRT 算法的基本假设. 在实际规划问题中, 智能体步长可以是动态可变的, 在某个时刻, 智能体也可以停在原来位置不动, 然而仅考虑路径生成本身, 这些假设可以通过取较小的步长进行近似.

定义 3 (路径). 假设智能体 ag 在任务空间 M 中运动, 定义智能体 ag 在连续时刻 $\tau_0,\tau_1,\ldots,\tau_{N^{\text{time}}}$ 的位置序列为其在该时段的路径, 记为 $pth=\left[\langle\tau_0,p_{\tau_0}\rangle,\langle\tau_1,p_{\tau_1}\rangle,\ldots,\left\langle\tau_{N^{\text{time}}},p_{\tau_{N^{\text{time}}}}\right\rangle\right]$. 这里, 为了简化问题, $p_{\tau_0},p_{\tau_1},\ldots,p_{\tau_{N^{\text{time}}}}$ 为 ag 在各时刻的位置信息, 其满足定义 2 中智能体的基本运动规律, 即 (在后面的公式中, 我们统一用 $A.B$ 表示 A 的子元素 B):

$$(p_{\tau_{i+1}}.x,p_{\tau_{i+1}}.y)=(p_{\tau_i}.x+\delta^x_{\tau_i,\tau_{i+1}},p_{\tau_i}.y+\delta^y_{\tau_i,\tau_{i+1}}) \tag{1}$$

其中, $[\delta^x_{\tau_i,\tau_{i+1}},\delta^y_{\tau_i,\tau_{i+1}}]$ 表示智能体从 τ_i 时刻到 τ_{i+1} 时刻的运动向量, 其满足:

$$(\delta^x_{\tau_i,\tau_{i+1}})^2+(\delta^y_{\tau_i,\tau_{i+1}})^2=r^2 \tag{2}$$

需要指出, 相较于以上 3 个定义, 实际场景可能会更加复杂, 例如任务空间并非定义 1 中所说的矩形, 空间中的障碍物也可能有很多种 (如建筑物这样的实体威胁以及自然灾害或恶劣天气这样的概率威胁[24]), 而智能体也可能不是定义 2 中的质点, 而是具有复杂几何形状的三维刚体 (如智能车、物流机器人、无人机等), 其在某时刻的行为也可能是多样的 (如加速、减速、换挡等)[25]. 为了测试方法的通用性及简洁性, 本文讨论路径规划问题的最基本假设, 并试图通过设计通用的测试方法发现路径规划算法本身的性能问题. 基于以上 3 个定义, 智能体的路径规划问题 (如图 1 所示) 可如下表示.

智能体路径规划问题: 现有环境 E, 包含的任务空间 M 以及障碍物集合 Obs. 假设环境中有一智能体 ag, 令 $task=\langle p^B,p^E\rangle$ 为智能体的规划任务, 其中 p^B 表示智能体的初始位置, p^E 表示智能体的目标位置. 智能体路径规划问题要求智能体在规定时间内生成从起始位置 p^B 到达目标位置 p^E 的安全可行的路径, 即生成路径 $pth=\left[\langle\tau_0,p_{\tau_0}\rangle,\langle\tau_1,p_{\tau_1}\rangle,\ldots,\left\langle\tau_{N^{\text{time}}},p_{\tau_{N^{\text{time}}}}\right\rangle\right]$ 满足以下条件.

(1) 智能体从起点出发到达终点, 即 $p_{\tau_0}=p^B$ 且 $p_{\tau_{N^{\text{time}}}}=p^E$.

(2) 路径 pth 中任意两个相邻的位置点 p_{τ_i} 与 $p_{\tau_{i+1}}$ 满足运动约束, 即公式 (1) 和公式 (2).

(3) 为了避免规划中出现“死循环”“死胡同”等情况, 要求每次规划时间不能超过 τ^{limit}, 即 $\tau_{N^{\text{time}}} \leqslant \tau^{\text{limit}}$.

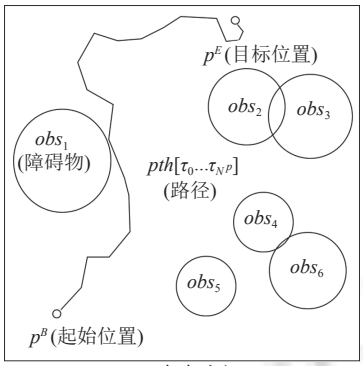

图 1 智能体路径规划问题示意图

1.2 基于快速随机探索树的路径规划算法

路径规划算法旨在解决第 1.1 节中提出的智能体路径规划问题. 算法根据智能体获取到的当前环境信息, 在满足问题约束前提下生成合理的路径, 使得智能体按照该路径行动能够快速、准确地到达目标点, 并且避开沿途的威胁区域. 目前的相关研究工作大致可分为基于图搜索的算法、基于进化的算法、基于随机采样的算法. 基于图搜索的路径规划算法将路径规划问题转化为图论中的路径搜索问题, 并使用 A*[26]、混合 A*以及跳点搜索[27]等算法生成路径. 一般的智能体路径规划问题可以转化为钢琴搬运问题, 其规划可行解的复杂度已经被 Reif 证明具有 PSPACE-hard 的下界[28], 这意味着没有能够解决该问题所有实例的多项式时间算法. 因此, 像是 A*这样的最优搜索算法很可能无法满足实际规划的时间约束, 而目前很多研究工作都试图在有限时间内产生近似最优的可行路径. 例如, 基于进化的路径规划算法 (如遗传算法 (GE)[29]、差分进化算法 (DE)[30]、蚁群优化算法 (ACO)[31]、粒子群优化算法 (PSO)[30]等) 将环境及路径进行编码, 并通过多次迭代的方式找到可行解并逐步提高可行解的质量. 然而进化算法的普遍问题是容易陷入局部最优, 这使得生成的路径质量很可能无法随着规划时间的增加而得到改善, 有些情况下甚至无法得到可行解. 因此, 目前在各种物理信息系统中应用最广泛的是基于随机采样的路径规划算法[1,8]. 该类算法结合了随机几何图 (random geometric graph, RGG) 理论, 将路径的生成转化为基于随机图论的路径搜索问题[32]. 以 RGG 理论为支撑, 基于随机采样方法具有许多优秀的数学特性, 包括概率完备性和渐近最优性[16]. 具体来说, 基于随机采样的路径规划方法在考虑各种约束的同时, 通过随机扩展的方式将搜索引导向未知区域, 进而快速地将搜索覆盖到整个任务空间, 最终找到可行路径. 此外, 由于该类方法在默认条件下对整个任务空间的初始搜索概率是均等的, 并不依赖特定领域的启发式信息, 因此其具有良好的普适性. 其中, 本文所采用的快速扩展随机树算法 (RRT) 就是目前应用最广泛的智能体路径规划方法[8]. RRT 算法利用了树结构的优势, 通过随机采样策略, 快速地生成一棵遍布于整个空间的随机树 (如图 2 所示), 并通过反向回溯的方式以线性的时间复杂度生成一条叶子节点到根节点的路径作为规划问题的解. 具体来说, 通过 RRT 算法进行路径规划时可以分为构建随机树和生成路径两个环节.

(1) 构建随机树. 任务空间 M 中的一棵树可以表示为 $Tree = \{nd_0, nd_1, \ldots, nd_{N^{\text{tree}}}\}$, 其中, 任意节点 $nd \in Tree$ 可以写成 $\langle nd^p, p^c, ND^s \rangle$ 的形式, 这里, p^c 表示该节点在 M 中的坐标, $nd^p \in Tree$ 表示节点 nd 的前驱节点 (树结构每个节点只有一个前驱), 节点集合 $ND^S \subset Tree$ 表示节点 nd 的后继节点集合 (树结构的每个节点可以有多个后继节点). 需指出, RRT 随机树的后继节点上限个数是可以设置的, 本文采用经典的二叉树设置, 即固定每个节点的最大后继节点个数为 2. 此外, 为了验证算法的稳定性, 还将在第 4 节特别探讨当生成树呈现为三叉树结构时测试算法的

具体表现. 记 nd_0 为根节点, 满足 $nd_0.nd^p = None$; 若节点 nd 满足 $nd.ND^s = \varnothing$, 即该节点的后继为空, 则称该节点为叶子节点.

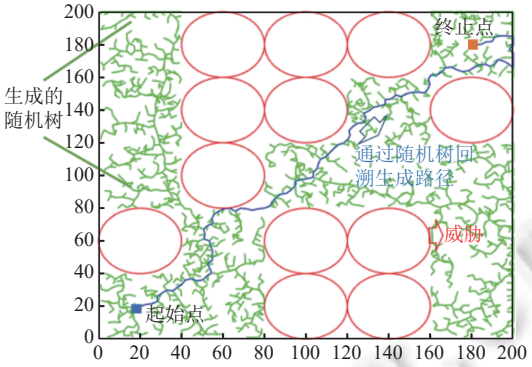

图 2 RRT 路径规划算法示意图

基于上述定义, 现已知环境 E 包含任务空间 M 和威胁集合 Obs, 规划任务为 $task$ 包含起点 p^B 和终点 p^E, 则随机树的具体构建方法如下.

Step 1. 初始化. 在 M 空间内初始化一棵只有根节点的树 $Tree = \{nd_0\}$, 并将根节点的位置设为任务的起始点, 即令 $nd_0.p^c = task.p^B$.

Step 2. 随机采样. 假设当前已经生成的随机树为 $Tree$, 在空间 M 中随机生成一个临时的坐标点 p^{tmp}, 并在 $Tree$ 中搜索与 p^{tmp} 最近且后继节点未满的节点 nd', 即:

$$nd' = \underset{nd \in Tree \wedge |nd.ND^s| < 2}{\arg\min} dis(nd.p^c, p^{\text{tmp}}) \tag{3}$$

其中, dis 为距离函数.

Step 3. 更新树. 求 nd' 到 p^{tmp} 的向量 $\boldsymbol{np}$:

$$\boldsymbol{np} = [p^{\text{tmp}}.x - nd'.p^c.x, p^{\text{tmp}}.y - nd'.p^c.y] \tag{4}$$

Step 4. 尝试以 nd' 为父节点沿着 $\boldsymbol{np}$ 方向扩展一个新节点 nd^{new}, 该节点在 nd' 到 p^{tmp} 的连线上, 并且 nd^{new} 到 nd' 的距离等于智能体的步长 $ag.r$. 具体来说, 令:

$$(\delta x, \delta y) = ag.r \cdot \left(\frac{\boldsymbol{np}.x}{\|\boldsymbol{np}\|}, \frac{\boldsymbol{np}.y}{\|\boldsymbol{np}\|} \right) \cdot (\delta x, \delta y) \tag{5}$$

则有:

$$nd^{\text{new}} = \langle nd', (nd'.x + \delta x, nd'.y + \delta y), \varnothing \rangle \tag{6}$$

公式 (6) 将新的叶子节点 nd^{new} 的前驱节点设为 nd', 即令 $nd^{\text{new}}.nd^p = nd'$, 并将其后继节点集设为空.

Step 5. 检查新节点 nd^{new} 的有效性, 即其是否与威胁障碍物发生碰撞, 如公式 (7) 所示.

$$\exists obs \in E.Obs,\ nd^{\text{new}}.p^c \in obs \tag{7}$$

若公式 (7) 满足则说明新的节点会引发碰撞, 那么我们删除该节点, 并返回 Step 2 继续采样; 反之, 则说明新的节点是安全的, 那么我们将该节点加入当前的树中, 即令:

$$Tree = Tree \cup \{nd^{\text{new}}\} \tag{8}$$

同时, 向其父节点 nd' 的后继节点集中加入 nd^{new}, 即令:

$$nd'.ND^s = nd'.ND^S \cup \{nd^{\text{new}}\} \tag{9}$$

其中, nd' 表示节点 nd 的子节点集合.

Step 6. 停止条件. 判断新节点 nd^{new} 与规划任务的目标点 $task.p^E$ 的距离, 若该距离大于阈值 θ^G, 则返回 Step 2, 继续随机树的构建; 反之, 若 nd^{new} 与 $task.p^E$ 的距离小于阈值 θ^G, 说明我们已经找到到达终点的路径, 则将 nd^{new} 记

作最后生成的节点 nd^{fin}, 并停止随机树的构建.

(2) 生成路径. 通过上述方法, 我们可以构建一棵在任务空间 M 中的随机树 $Tree$, 并且保证以下条件, 那么我们可以通过回溯的方法来生成最后的路径.

1) 该随机树中的所有节点都是安全的 (即在威胁的外部).

2) 最后加入的新节点能够到达目标点.

具体来说, 从最后生成的节点 nd^{fin} 开始, 沿着前驱节点反向回溯至随机树的根节点 nd_0, 就可以生成最终路径了, 即 RRT 算法最后生成的路径为:

$$pth=\left[\langle\tau_0,p_{\tau_0}\rangle,\langle\tau_1,p_{\tau_1}\rangle,\ldots,\left\langle\tau_{N^{\text{time}}},p_{\tau_{N^{\text{time}}}}\right\rangle\right] \tag{10}$$

满足:

$$p_{\tau_0}=task.p^B=nd_0.p^c,$$

$$p_{N^{\text{time}}}=task.p^E=nd^{\text{fin}}.p^c,$$

$$\forall i\in(0,N^{\text{time}}),p_{\tau_i}=nd_i.p^c,$$

其中,

$$nd_i=nd_{i+1}.nd^p.$$

若当前随机树包含节点总数为 N^{tree}, 则每添加一个新节点的复杂度为 $O(N^{\text{tree}})$, 而通过回溯法生成最终路径的复杂度为 $\log(N^{\text{tree}})$. 此外, 我们还可以通过划分子区域并进行哈希映射的方式来减少 Step 2 的计算时间. 因此, RRT 算法可以在多项式时间内迅速生成扩展到整个任务空间的随机树, 并最终找到路径, 如图 2 所示. 此外, 从概率意义上来说, 随机树中最先到达目标点的分支很可能具有较短的路径长度, 因此 RRT 在一定程度上保证了生成路径的质量. 此外, 许多 RRT 的改进方法, 如改进快速随机探索树算法 (RRT*)[16]、快速行进树算法 (FMT)[33]等, 都具有渐近最优性, 即随着迭代次数或样本数量的增加而收敛到最优解. 本研究旨在提供多智能体路径规划的通用框架, 因此我们将最基本 RRT 路径规划算法作为被测对象.

1.3 面向路径规划算法的软件测试方法

软件测试旨在通过设计合理的测试用例并将其作为输入运行被测系统, 来发现被测软件中的问题[34]. 针对软件本身、软件的功能及性能需求、软件的错误形式及触发机理的多样性和复杂性, 学者们基于不同的思想及理念提出了各种测试方法及策略, 如解决 Oracle 问题的蜕变测试[35]、解决测试充分性问题的变异测试[36]等.

随机测试 (random) 被认为是最基本的自动化测试用例生成方法[37], 该方法通过在被测系统的输入空间进行随机采样的方式来构造测试用例. 尽管 RT 方法简单, 但随机测试能够保证输入空间中任何可能的测试用例都有一定概率被选中, 这就为测试效率提供了一个下界[37]. 因此, 目前很多测试方法都将随机测试与被测软件的自身属性以及相关领域知识结合, 提出新的测试方法, 旨在保留测试用例多样性的同时提升测试效率. 例如, 大量测试与调试的经验都表明导致失效的测试用例往往在输入域中聚集成连续的区域. 基于此 Chen 等人[38]提出了自适应随机测试 (adaptive random testing, ART) 方法. ART 在测试用例生成阶段引入反馈, 试图使生成的测试用例均匀地分布于输入域, 以提高失效检测效率, 例如最著名的 ART 算法 FSCS-ART 通过从固定大小的候选测试集中选择与当前测试信息差异最大的测试用例作为下一个测试输入来达成 ART 的目标[39]. 目前基于不同测试需求, 学者还提出了 DF-FSCS[40]等多种 ART 算法.

尽管与 RT 相比, ART 能够在较短的时间内生成多样的测试用例, 然而 ART 算法大多在被测软件的输入空间完成闭环, 无法根据测试结果进行搜索或寻优; 然而被测软件的失效或其他性能缺陷的相关信息很可能在其输出信息上得到反映, 而这些信息很可能被 ART 忽略. 为此, Cai 等人[41]将控制论思想引入软件工程中, 提出自适应测试 (AT) 方法, 同时将测试的输入和输出作为反馈信息, 并以此来控制测试的执行过程. 此外, 文献 [18,42] 将 AT 与 RT 结合, 提出了动态随机测试 (dynamic random testing, DRT) 算法. DRT 引入测试剖面的概念, 将其作为测试策略, 并根据历史测试信息动态调整测试剖面本身, 使得测试策略逐渐优化, 进而更好地发现软件的缺陷.

近年来, 随着集群路径规划逐渐成为研究热点, 相关的测试问题也逐渐受到学者的关注. 例如, 在自动驾驶领域, 文献 [25,43,44] 就通过引入可避免碰撞的概念以及考虑实际交通中的车辆的行为模式对智能车的路径规划器进行测试, 以发现关键的交通事故以及其他需求违反模式. Gambi 等人[45]则通过构建事故示意图的方式对自动驾驶的规划器进行测试. Zhang 等人[2]则基于自动驾驶路径规划器的测试信息对其失效的原因进行了分析. 朱宇等人[46]讨论了自动驾驶汽车在变道 (lane-change) 场景下的测试方法, 而夏春燕等人[47]则面向自动驾驶汽车建立了路口 (crossing) 场景的测试生成模型. 然而针对通用路径规划算法本身的测试研究还未系统性地展开, 现有面向智能体路径规划算法的测试方法还相对较少. 现有工作主要着眼于针对这一领域的某些特定属性来生成测试用例. 例如, 文献 [9,13] 基于场景的对称性以及威胁对路径的影响关系构建蜕变规则, 进而提出蜕变测试方法. 文献 [19,48] 基于多智能规划分别提出了蜕变测试方法和自适应随机测试方法.

总体来说, 目前对路径规划算法的测试研究尚处于起步阶段. 现有相关工作大多基于算法的特定应用领域 (如自动驾驶) 进行测试. 尽管这些系统可能包含路径规划模块, 然而这些测试工作主要着眼于检查多个模块间的协作能否适应相应的任务需求, 而并非着眼于路径规划算法本身的性能风险. 现有针对路径规划算法测试的文章则主要针对路径规划的测试预期问题提出具体的蜕变规则和策略, 并没有涉及测试用例的生成策略. 截至目前, 尚未发现有学者为智能体路径规划算法提出一个完整的动态测试框架. 本文面向该问题提出了具有较强普适性的测试用例生成及优化方法: 该方法构建了通用且合理的测试剖面, 并基于较为通用的测试输入输出信息设计反馈机制来更新测试剖面. 此外, 我们还借鉴差分[49,50]和蜕变测试的思想定义了面向基于随机采样的路径规划算法的测试结果评价指标.

2 面向智能体路径规划算法的动态随机测试方法 (DRT-PP)

本节介绍面向智能体路径规划算法的动态随机测试方法. 具体来说, 第 2.1 节介绍智能体路径规划算法的测试框架, 第 2.2 节介绍 DRT-PP 的流程, 第 2.3 和 2.4 节分别介绍该方法的两个重要组成部分: 测试剖面的构建以及测试剖面的更新策略.

2.1 面向智能体路径规划算法的测试

软件测试旨在通过生成并运行一系列的测试用例, 来发现被测系统中潜在的失效风险. 本文讨论智能体路径规划算法的测试问题, 希望生成多样的路径规划场景作为测试用例来运行被测算法, 并对测试结果进行分析和评估.

定义 4 (测试用例). 将路径规划算法的测试用例 t 定义为元组 $\langle E, task\rangle$, 其中, $E=\langle M, obs\rangle$ 为规划环境, 包括 $X\times Y$ 的任务空间 M 以及威胁集合 Obs; $task=\langle p^B, p^E\rangle$ 为规划任务, 其中, p^B 和 p^E 为起始点和目标点. 本文主要针对威胁集合 p^B 和 p^E 的分布来设计测试用例, 并在测试过程中固定 $task$ 和 M, 因此测试用例 t 的量化特征可以用 $t.Obs$ 来表示 (在实际测试时, 我们可以通过调节环境 E 中 Obs 的分布以及仿真时的步长来等效不同的任务 $task$ 以及各种形状的任务空间 M, 因此, 这种简化并不失一般性).

一般来说, 软件测试的目的是发现被测软件中潜在的失效风险, 而本文旨在通过对智能体路径规划算法进行测试来暴露其失效风险. 在实际应用场景中, 根据不同的需求可以采用不同的标准来评价生成路径的质量, 如在自动驾驶场景会考虑路径的光滑度、最大转角、最大加速度减速度, 无人机场景会考虑油耗等指标. 本文讨论路径规划算法的一般需求, 即在不考虑特定场景约束的情况下, 将路径长度作为评估指标. 在相同条件下生成的路径越短, 路径质量就越高, 因为在一般情况下, 算法生成路径越短, 智能体在执行过程中所消耗的时间和资源就越低. 目前经典路径规划算法的概率收敛性都经过了证明, 因此只要算法实现正确且规划时间足够, 这些智能体都能够找到一条完整且无碰撞路径. 然而, 实际问题中受到时间和资源的限制, 规划算法无法求出最优路径, 其只能寻求在短时间内找到满足任务需求的可行解. 此外, 基于采样的路径规划算法大多包含一定程度的不确定性. 例如本文的被测对象, 即被广泛应用的 RRT 算法作为概率收敛算法能够保证智能体在较短的时间内规划出从起点到终点的连通路径, 但该路径并非最优的. 此外, 由于 RRT 的核心是随机树的构架, 因此其在同一场景下生成的路径与不相同. 因此, 在实际场景中 RRT 的性能 (如稳定性、寻优性等) 是需要被进一步考察的. 具体来说, 算法可能存在以下风险.

(1) 路径的质量不满足需求. 如果实际生成路径的长度与最优路径相差很大, 那么智能体按照实际路径行动时会花费大量额外的时间成本.

(2) 算法的稳定性较差. 由于算法的不确定性, 即使当前生成路径满足需求, 也无法保证算法相同场景下总能生成满足需求的路径.

由于上述风险的存在, 路径规划算法的测试存在测试预期 (oracle) 问题[51], 即在最优解未知的情况下我们无法判断当前的测试结果是否失效. 经验表明, 尽管算法的用户对路径长度 (或规划时间) 有需求, 但由于场景的复杂性, 通常无法在不考虑最优解的情况下对该需求进行量化. 例如, 假设算法在某个场景下生成路径很长, 其原因既可能是算法在性能上的失效也可能是场景本身过于复杂, 而我们也不能确定该路径与最优路径的偏差是否过大, 是否能够满足实际的性能需求.

为此, 本文借鉴蜕变测试思想来设计测试用例的评估准则. 假设用被测路径规划算法重复运行 N^{repeat} 次测试用例 t, 得到各次运行后生成路径的集合 $PTH^t=\{pth_1^t, pth_2^t, \ldots, pth_{N^{\text{repeat}}}^t\}$, 则我们得到蜕变关系: 若所有路径都具有较高的质量, 即与最优路径 pth^{t*} 的差距不大, 则这些路径彼此之间的差异也不会很大. 因此, 我们可以通过 PTH^t 中路径彼此之间的差异来评价被测算法在测试用例 t 上的表现, 因为尽管最优路径 pth^{t*} 未知, 但当 PTH^t 中的路径之间差异较大时, 则一定存在一条生成路径 $pth^t \in PTH^t$ 与最优路径 pth^{t*} 具有更大的差异, 即:

$$\exists i, j, \left|pth_i^t - pth_j^t\right| \geqslant d \rightarrow \exists k, \left|pth^{t*} - pth_k^t\right| > d \tag{11}$$

此外, 路径彼此之间具有较大的差距也能反映出被测算法具有较差的稳定性. 基于以上分析, 本文将同一个测试用例多次执行后生成路径的长度样本方差作为该测试用例输出结果的失效度, 即被测算法在测试用例 t 上的表现的失效度为:

$$S(t)=\left(\sum_{i\in[1,N^{\text{repeat}})}\left(\left|pth_i^t\right|-\overline{|pth^t|}\right)^2\right)\Big/(N^{\text{repeat}}-1) \tag{12}$$

在软件测试及程序证伪 (falsification) 领域, 度量需求满足程度的一种经典方法是计算实际输出与规格说明 (specification) 的差异度. 现考虑路径规划算法测试, 假设某个测试场景下最优路径为 pth^*, 且重复执行 N^{repeat} 次得到各路径的长度距离最优路径的离差为 $S^*(t)$, 则尽管我们不易得到 pth^* 并计算 $S^*(t)$, 但我们却可以证明 $S^*(t)>S(t)$. 也就是说, 失效度 $S^*(t)>S(t)$ 是测试输出与最优结果离差的一个下界, 其能够反映测试输出与最优路径的差异度, 这与基于路径长度的需求满足程度的概念相符. 因此尽管我们无法确切判断一个测试用例是否失效, 但对于 $S(t)$ 值较大的测试用例, 其不满足需求的可能性就越大. 而我们测试的目标也可以量化为生成使 $S(t)$ 尽可能大的测试用例.

总的来说, 作为智能体路径规划算法的测试框架包含以下部分.

(1) 测试用例生成. 基于某种测试用例生成策略生成测试用例 $t_1, t_2, \ldots$ 形成测试用例集 T, 其中, 每个测试用例 t 为特定的路径规划环境, 其特征可以量化为场景中的威胁分布, 即 $t.Obs$.

(2) 测试用例集执行. 多次执行每个测试用例 $t\in T$, 并记录测试结果, 生成路径 $pth_1^t, pth_2^t, \ldots, pth_{N^{\text{repeat}}}^t$.

(3) 测试结果的评估. 根据公式 (12) 计算测试用例 t 生成路径之间的标准差 $S(t)$, 并将其作为失效度. 对那些 $S(t)$ 值较大的测试用例做进一步分析.

2.2 动态随机测试方法框架

本文试图将 DRT 思想引入到路径规划算法中, 提出面向路径规划算法的自适应随机测试方法, 其总体框架如图 3 所示. 自适应随机测试属于软件控制论范畴, 其引入测试剖面概念, 并将其作为“控制器”或是“测试策略”来控制测试用例的生成和执行. 作为核心概念, 测试剖面可以看成是被测系统输入空间的一个概率分布 (或概率密度函数), 表示被测系统的每个输入被选取的概率或概率密度. 而 DRT 的基本思想与强化学习类似, 通过在测试过程中动态调整测试剖面, 来高效地达成测试目标. 具体来说, 在测试开始时, DRT 通常采取随机测试策略; 而在测试过程中, DRT 持续收集测试信息, 逐步加深对被测对象及其潜在失效的了解. 以此为依据, DRT 在每次执行后对测试剖面进行动态更新, 使得当前的测试策略能够更好地适配被测对象的特性, 以达到快速发现失效的目的.

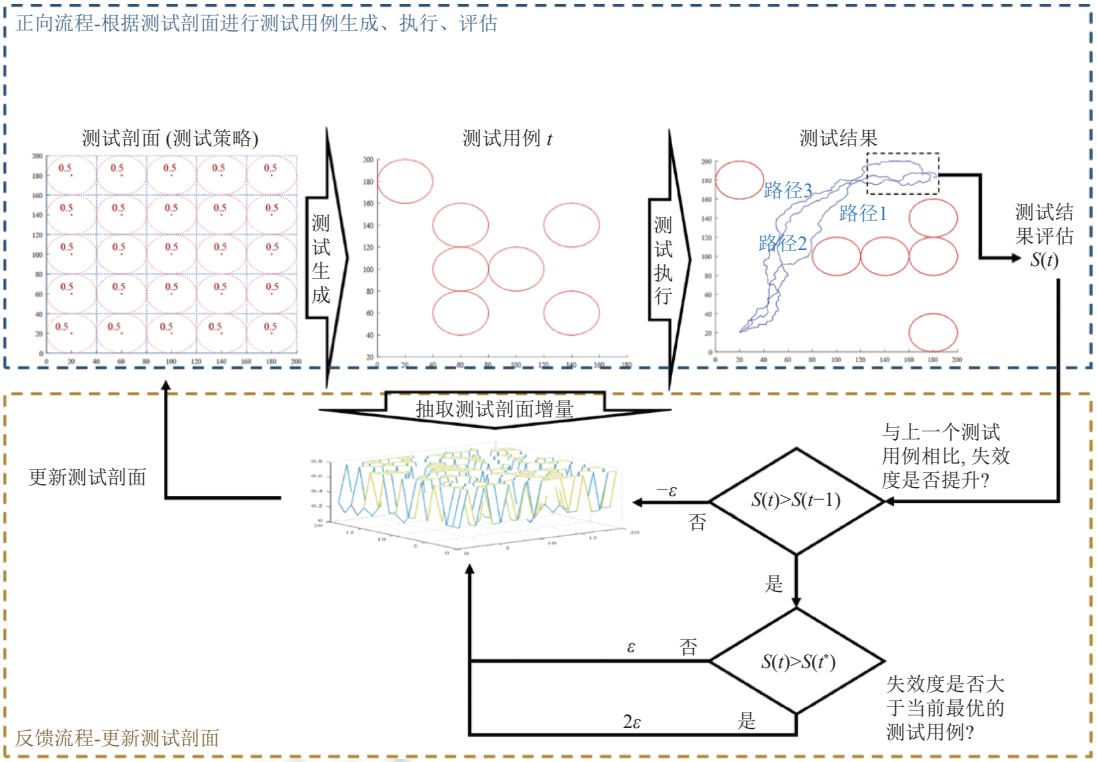

图 3 DRT-PP 框架图

基于该思想, DRT-PP 分为正向流程和反馈流程. 其正向流程是正常的测试过程, 包含测试用例生成、测试用例执行、测试用例的结果分析, 如图 3 的上半部分所示. 所不同的是, DRT-PP 根据路径规划场景的特性构建测试剖面, 并以该剖面作为测试策略生成测试用例. 根据定义 4, 路径规划算法的输入 (即测试用例 t) 为环境 E, 而我们假设任务场景 M 固定, 则生成一个测试用例就是要确定威胁集合 Obs 的分布. 因此, DRT-PP 的测试剖面应为威胁分布的概率密度分布函数: $h: O \to \mathbb{R}$, 其中, O 为所有可能的 Obs 分布的集合. 考虑到方法的可行性, 本文将场景空间进行某种程度的离散化, 例如将 200×200 的任务空间 M 按每 10 个单位进行切分, 则整个任务空间就包含 400 个子区域, 而在构造测试用例时我们需要考虑每个子区域内是否包含威胁. 经离散化后的测试剖面就可以表示为测试用例的概率函数, 即:

$$\mathcal{P}: \mathcal{T} \to [0,1] \tag{13}$$

满足:

$$\sum_{t\in\mathcal{T}} \mathcal{P}(t) = 1,$$

其中, $\mathcal{T}$ 为所有可能测试用例的集合, 而 $\mathcal{P}(t)$ 为测试用例 t 被生成的概率. 这样, 我们就可以根据测试剖面来生成测试用例, 这里将通过测试剖面 $\mathcal{P}$ 生成测试用例 t 的操作记为:

$$t = Generate(\mathcal{P}) \tag{14}$$

具体测试剖面的构建方法将在第 2.3 节介绍.

在测试过程中, DRT-PP 需要根据被测算法的输出实时动态地更新测试剖面, 以触发较大的失效度 $S(t)$, 如图 3 下半部分所示. 现假设已经生成并执行的测试用例集为 $T = \{t_1, t_2, \ldots, t_{N^T}\}$, 而执行这些测试用例后的得到的测试结果集合为 $\mathcal{L}^T = \{PTH^{t_1}, PTH^{t_2}, \ldots, PTH^{t_{N^T}}\}$, 则 DRT-PP 的测试剖面调整策略 φ 应与当前测试集 T、当前测试集的输出 $\mathcal{L}^T$ 以及当前的测试剖面 $\mathcal{P}_{N^T}$ 有关, 那么更新后的测试剖面 $\mathcal{P}_{N^T+1}$ 就可以按公式 (15) 计算.

$$\mathcal{P}_{N^T+1}=\varphi(T,\mathcal{L}^T,\mathcal{P}_{N^T}) \tag{15}$$

具体来说, 根据 DRT 的基本思想, 若当前测试用例 t 的输出结果能够逼近测试目标, 那么新测试剖面的调整方向就应趋向于更容易生成当前测试用例 t, 即做正向调整; 相反, 若当前测试用例 t 偏离了测试目标, 则测试剖面的调整方向就应趋向于避免生成当前测试用例 t, 即做负向调整. 现假设当前生成的测试用例为 t_{N^T}, 执行该测试用例 N^{repeat} 次得到的输出路径集合为 $PTH^{t_{N^T}}$, 通过公式 (12) 计算的失效度为 $S(t_{N^T})$. 那么当 $S(t_{N^T})$ 大于等于上一次执行的测试用例 $S(t_{N^T-1})$, 或是大于等于当前最优测试用例 t^* 的失效度 $S(t^*)$ 时, 我们进行正向调整, 记为 $\mathcal{P}_{N^T+1}=\varphi^+(T,\mathcal{L}^T,\mathcal{P}_{N^T})$; 反之, 当 $S(t_{N^T})$ 小于 $S(t_{N^T-1})$ 时, 则我们进行负向调整, 记为 $\mathcal{P}_{N^T+1}=\varphi^-(T,\mathcal{L}^T,\mathcal{P}_{N^T})$. 具体的测试剖面更新策略将在第 2.4 节中进行详细介绍.

综上所述, DRT-PP 的执行流程如下.

Step 1. 初始化测试任务. 令已经生成的测试用例集 $T=\varnothing$, 生成测试用例数量 $N^T=0$, 测试用例输出集合 $\mathcal{L}^T=\varnothing$; 设上一个测试用例失效度 $S(t_0)=-\infty$, 当前最优的测试用例为 $t^*=None$, 其失效度为 $S(t^*)=-\infty$. 令初始测试剖面为纯随机, 即对任意场景 t, 有 $\mathcal{P}(t)=1/|\mathcal{T}|$.

Step 2. 测试用例生成. 令 $N^T=N^T+1$, 根据公式 (14), 从当前测试剖面 $\mathcal{P}_{N^T}$ 生成当前测试用例 t_{N^T}. 并将 t_{N^T} 加入测试用例集合 T, 即令 $T=T\cup\{t_{N^T}\}$.

Step 3. 测试用例执行. 执行测试用例 t_{N^T} (假设重复执行 N^{repeat} 次), 得到 t_{N^T} 的输出的路径集合 $PTH^{t_{N^T}}$. 将 $PTH^{t_{N^T}}$ 加入 $PTH^{t_{N^T}}$, 即令 $\mathcal{L}^T=\mathcal{L}^T\cup\{PTH^{t_{N^T}}\}$.

Step 4. 测试结果评估. 根据公式 (12) 计算生成路径的失效度 $S(t_{N^T})$.

Step 5. 测试剖面更新. 调整测试剖面, 若 $S(t_{N^T})>S(t^*)$ 或 $S(t_{N^T})>S(t_{N^T-1})$, 则进行正向调整, 生成新的测试剖面 $\mathcal{P}_{N^T+1}=\varphi^+(T,\mathcal{L}^T,\mathcal{P}_{N^T})$; 若 $S(t_{N^T})>S(t_{N^T-1})$, 则进行负向调整, 即 $\mathcal{P}_{N^T+1}=\varphi^-(T,\mathcal{L}^T,\mathcal{P}_{N^T})$.

Step 6. 更新最优结果. 更新当前的最优测试用例, 即若 $S(t_{N^T})>S(t^*)$, 则令 $S(t^*)=S(t_{N^T})$, $t^*=t_{N^T}$.

Step 7. 停止条件判定. 若已经执行的测试用例数量达到测试资源的上限, 即 $N^T=N^{\text{limit}}$, 则停止测试, 输出生成的测试用例集 T、测试结果 $\mathcal{L}^T$、最优的测试用例 t^*; 反之, 若 $N^T<N^{\text{limit}}$, 则继续进行测试, 返回 Step 2.

2.3 测试剖面设计

面向智能体路径规划算法构建测试剖面就是要找到一个概率分布 $\mathcal{P}$ 来表征每个场景被选为测试用例的概率. 作为最为基础的测试策略, 随机测试 RT 理论上应该可以在整个可能的威胁分布空间 O 中按均匀概率分布随机选取某个威胁分布 Obs 作为测试用例, 此时每个场景被选中的概率为 $1/|O|$. 然而在路径规划测试中, 即使是纯随机策略也不易实现, 因为我们必须在考虑每个场景中包含威胁的数量、形状、位置的同时, 满足等概率选取. 此外, 威胁分布空间 O 中包含大量的无效测试用例. 图 4 就描绘了一个典型的无效测试用例: 多个威胁把无人机起始点完全包围, 导致可行路径不存在. 如果大量生成这样的测试用例, 不仅无法对算法性能做出有效的评估, 还会浪费大量测试资源.

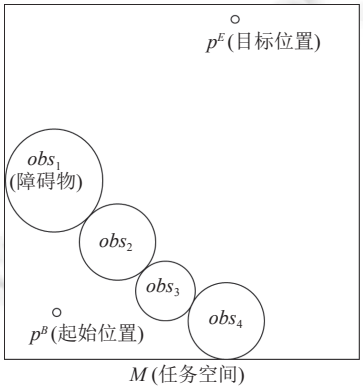

图 4 没有解的无效测试用例

因此, 在实际测试过程中我们有如下希望.

(1) 构建某种测试剖面, 并将其作为测试策略来选取测试用例, 以生成符合特定概率分布的测试用例, 并且相似的测试用例被选中的概率也应该相似, 这样可以防止生成一些意料之外的无效场景.

(2) 对于同一测试用例 t, 相似的测试剖面生成 t 的概率也应该相似, 这样方便我们设计测试剖面更新策略, 使测试剖面能够更趋向于生成 t 或避免生成 t.

基于此, 本节通过离散化思想提出一种基于为威胁区域离散化的测试剖面 (threat area discretization based profile, TAD 剖面).

假设智能体的任务空间 M 为一个 $X\times Y$ 的矩形区域, 将 M 离散化为若干格子样式的子区域 (如图 5 所示):

$$M=\bigcup_{i=1,2,\cdots,D_1;\ j=1,2,\cdots,D_2} M_{ij} \tag{16}$$

其中, D_1、D_2 分别为任务空间横向与纵向划分的份数. 每个子区域 M_{ij} 就可以看成是长为 $X/|D_1|$, 宽为 $Y/|D_2|$, 中心为 $\left(\frac{X}{|D_1|}\cdot\left(i-\frac{1}{2}\right),\frac{Y}{|D_2|}\cdot\left(j-\frac{1}{2}\right)\right)$ 矩形. 为了简洁, 本文只考虑正方形任务空间, 并且横纵划分的份数相等, 因为任何矩形任务空间都可以通过在正方形场景中填充威胁的方式得到, 因此在后面的论述中我们令 $D_1=D_2=D$.

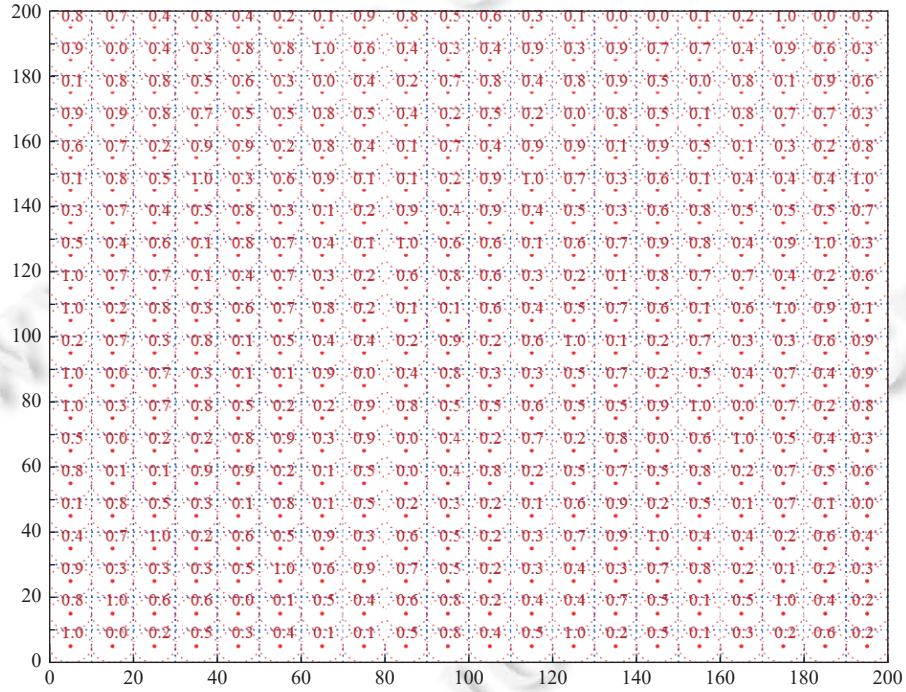

图 5 TAD 剖面矩阵

在此基础上, 我们通过指定每个子区域 M_{ij} 包含威胁的概率来设计测试剖面. 首先, 我们规定, 每个子区域包含威胁的情况是二值的, 即子区域 M_{ij} 不是被某个威胁填满 (即 $M_{ij}\in Obs$), 就等价于不包含威胁 (即 $\forall obs\in Obs$, $obs\cap M_{ij}=\varnothing$). 定义 $Q=(q_{ij})^{D\times D}$ 为任务空间 M 的威胁概率矩阵, 其中 q_{ij} 表示子区域 M_{ij} 中包含威胁的概率, 即 $q_{ij}=P(M_{ij}\in Obs)$(如图 5 中每个格子内部的数字所示). 借助威胁概率矩阵 Q, 我们就可以按照概率生成每个子区域 M_{ij} 的威胁, 进而生成整个测试用例. 因此, 对于每个 Q, 都可以确定一个测试剖面 $\mathcal{P}^Q$, 使得在离散化假设下的, 基于该剖面生成任何威胁分布 Obs 的概率都可以由公式 (17) 计算.

$$\mathcal{P}^{Q}: P(Obs)=\prod_{\{i,j|M_{ij}\in Obs\}} q_{ij} \prod_{\{i,j|\forall obs\in Obs,\ obs\cap M_{ij}=\varnothing\}} (1-q_{ij}) \tag{17}$$

这里, 我们称威胁概率矩阵 Q 为测试剖面 $\mathcal{P}^{Q}$ 的特征矩阵.

TAD 剖面将测试用例的输入空间离散化, 保证了离散化条件下测试用例个数是有限的. 另一方面, 从图 5 可以看出当离散化的精度足够高后, 威胁分布的多样性是能够得到保证的. 图 5 中场景横竖被分成了 20 份, 共得到 400 个子区域, 那么该分割下可能产生的威胁分布情况 (即测试用例总数量) 为 2^{400}. 特别地, 当所有的 $q_{ij}=0.5$(即 $Q=0.5^{D_1\times D_2}$) 时, 每个测试用例被选中的概率均为 $1/2^{400}$, 这样我们就实现了纯随机的测试策略 RT. 此外, 对相似的测试剖面来说, 生成同一个测试用例的概率也会相似. 假设测试用例 t 在子区域 M_{ij} 中包含威胁, 假设特征 Q 中的 q_{ij} 与 Q' 中的 q_{ij} 相差 δ, 则相应的测试剖面 $\mathcal{P}^{Q}$ 和 $\mathcal{P}^{Q'}$ 生成测试用例 t 的概率也相差 δ 倍. 因此, 我们可以通过公式 (18) 来度量两个测试剖面之间的距离.

$$Dis(\mathcal{P}^{Q},\mathcal{P}^{Q'})=\|Q-Q'\|_1 \tag{18}$$

2.4 测试剖面更新策略

DRT-PP 通过多次执行同一个测试用例得到的路径集合的方差 $S(t)$ 来对测试用例的执行结果进行评价. 由第 2.2 节可知, 我们希望测试剖面的更新策略 φ 具有以下特征.

(1) 那些更接近测试目标, 即得到较大失效度 $S(t)$ 的测试用例能够以更大的概率被测试剖面生成.

(2) 那些远离测试目标, 即得到较小失效度 $S(t)$ 的测试用例能够以较小的概率被测试剖面生成.

由图 3 可知, 我们通过比较当前测试用例 t_{N^T} 的失效度 $S(t_{N^T})$、上次执行测试用例 t_{N^T-1} 的失效度 $S(t_{N^T-1})$、目前最优的测试用例 t^* 的失效度 $S(t^*)$ 来决定生成下一个测试用例 t_{N^T+1} 的测试剖面是趋近还是远离当前测试用例 t_{N^T}. 然而, 无论是趋近或是远离, 我们都需要将当前测试用例 t_{N^T} 与第 2.3 节中设计的测试剖面建立联系: 即给定测试用例 t, 我们需要得到一个“临时的”测试剖面 $\mathcal{P}^{Q^t}$, 使得通过 $\mathcal{P}^{Q^t}$ 生成测试用例 t 的概率较高.

假设测试用例 t 由某个经过离散化 TAD 剖面生成, 则 t 的威胁分布 $t.Obs$ 可用矩阵 $A^t=(a^t_{ij})^{D\times D}$ 来表示, 其中,

$$a^t_{ij}=\begin{cases}1, & M_{ij}\in t.Obs\\ 0, & \forall obs\in t.Obs,\ obs\cap M_{ij}=\varnothing\end{cases} \tag{19}$$

也就是说, 当第 2.3 节中的子区域 M_{ij} 被某个威胁 $obs\in t.Obs$ 填满时 $a^t_{ij}=1$, 反之 $a^t_{ij}=0$. 如果我们将 A^t 作为特征矩阵构造测试剖面 $\mathcal{P}^{A^t}$, 则该剖面生成测试用例 t 的概率就为 100%. 为了更好地体现测试用例 t 的特征, 我们引入扩散机制, 将距离 t 中威胁较近的子区域的威胁生成概率也一并进行放大. 具体来说, 我们根据如下规则构建矩阵 $Q^t=(q^t_{ij})^{D\times D}$.

(1) 若 $a^t_{ij}=1$, 则 $q^t_{ij}=1$.

(2) 若 $a^t_{ij}=0$, 即子区域 M_{ij} 不包含威胁, 则在那些包含威胁的子区域中, 找到距离 M_{ij} 最近的子区域, 记为 $M_{i'j'}$, 并令:

$$q^t_{ij}=0.5\sqrt{(i-i')^2+(j-j')^2} \tag{20}$$

也就是说, 子区域与威胁的距离增加一个单位, 其包含威胁的概率就减少一半. 最后, 我们将 Q^t 称为测试用例 t 的剖面特征矩阵, 并借助该矩阵对当前测试剖面进行调整.

假设当前测试用例为 t_{N^T}, 其失效度为 $S(t_{N^T})$; 上次执行的测试用例为 t_{N^T-1}, 其失效度为 $S(t_{N^T-1})$; 当前最优的测试用例为 t^*, 其失效度为 $S(t^*)$. 假设当前的测试剖面为 $\mathcal{P}^{Q_{N^T}}$, 测试剖面特征为 Q_{N^T}. 现求出测试用例 t_{N^T} 的剖面特征矩阵为 $Q^{t_{N^T}}$, 则可以通过公式 (21) 求取更新后测试剖面 $\mathcal{P}^{Q_{N^T+1}}$:

$$Q_{N^T+1}=(1-\delta)Q_{N^T}+s\cdot\varepsilon(Q^{t_{N^T}}-0.25\cdot\mathbf{1}) \tag{21}$$

其中, $\mathbf{1}$ 为全 1 矩阵; ε 是融合系数, 也称为测试剖面调整幅度; s 表示测试剖面的调整方向. 公式 (21) 将当前的测试剖面特征 Q_{N^T} 与当前测试用例的剖面特征 $Q^{t_{N^T}}$ 进行融合, 得到新的测试剖面特征 Q_{N^T+1}. 具体如下.

(1) 若当前测试用例的失效度大于目前最优测试用例的失效度, 即 $S(t_{N^T})>S(t^*)$ 时, 我们做较大幅度正向调整

φ^{++}, 即令 $s=2$.

(2) 若当前测试用例的失效度小于目前最优测试用例的失效度, 但好于上一个测试用例的失效度, 即 $S(t_{N^T})>S(t_{N^T-1})$ 时, 我们做正向调整 φ^{+}, 即令 $s=1$.

(3) 若当前测试用例的失效度小于上一个测试用例失效度, 即 $S(t_{N^T})<S(t_{N^T-1})$ 时, 我们做负向调整 φ^{-}, 即令 $s=-1$.

需指出, 这里测试剖面调整幅度 ε 为一个正数 (实验中分别取 0.05、0.1、0.15 和 0.2). $Q^{t_{N^T}}-0.25\cdot\mathbf{1}$ 将 $Q^{t_{N^T}}$ 中所有的 q_{ij} 减去一个值, 这样不仅可以增加距离测试用例 t_{N^T} 中威胁较近的子区域包含威胁的概率, 也能降低距离 t_{N^T} 中威胁较远的子区域包含威胁的概率. 此外, 我们对 Q_{N^T+1} 进行了缩放以保证 Q_{N^T+1} 中不含有大于 1 或者小于 0 的概率值. 此外, 为了防止 Q_{N^T+1} 的值越加越大, 我们将通过 Q_{N^T+1} 生成威胁的期望个数归一化为:

$$(1-\delta)NEQ_{N^T}+\delta NEQ^{t_{N^T}} \tag{22}$$

公式 (22) 不仅将原测试剖面与当前测试用例的剖面的分布趋势相融合, 还融合了二者的期望威胁数量, 以保证测试过程中场景的威胁数量不会持续增大并陷入某个局部最优. 最后我们得到新的测试剖面 $\mathcal{P}^{Q_{N^T+1}}$.

从第 2.2 节可知, 与随机测试相比, DRT-PP 额外的计算量主要来自测试剖面的更新, 即公式 (20)–公式 (22). 公式 (21) 和公式 (22) 都是矩阵的线性运算, 尽管公式 (20) 进行了指数运算, 但由于所有的运算都是基于整数, 因此我们可以在测试前执行该计算, 并在接下来的测试过程中直接使用存储结果. 因此, 相较于被测路径规划算法本身的执行代价, DRT-PP 的执行代价是可以忽略不计的. 此外, 假设有最优的测试用例 t 能够达到很高的失效度 $S(t)$; 而更新后的测试剖面 Q_{N^T+1} 是根据 t 的特征矩阵 Q^t 构建的, 因此其比更新前的测试剖面 Q_{N^T} 更容易生成 t, 因此该策略满足测试剖面更新的基本需求.

3 实验评估

3.1 研究问题

本节通过实验研究对 DRT-PP 方法的有效性进行验证, 具体来说, 我们试图回答以下 3 个研究问题.

- RQ1: DRT-PP 方法对 RRT 路径规划算法潜在失效的检测能力如何?
- RQ2: DRT-PP 方法生成的测试用例是否具有多样性?
- RQ3: 测试剖面的更新幅度 ε 是否会影响 DRT-PP 方法的有效性?

RQ1 旨在评估 DRT-PP 算法的有效性, 即 DRT-PP 是否能够发现失效度 $S(t)$ 较大的测试用例. 此外, 测试用例的多样性是路径规划算法能够被全面评测的关键, 而 DRT-PP 方法生成的测试用例能否涵盖不同的威胁分布模式是衡量其是否会陷入局部最优以及是否能够发现更多有意义的失效的重要指标, 相关问题将会在 RQ2 进行讨论. 最后, 尽管我们在设计 DRT-PP 时尽量考虑其通用性和普适性, 然而 DRT-PP 仍存在内部参数, 即测试剖面的更新幅度 ε, 而其带来的影响将会在 RQ3 中进行讨论.

除了 RQ1、RQ2 和 RQ3 之外, 我们还将对 DRT-PP 算法进行定性分析, 检查 DRT-PP 是否能够检测出那些规划结果不稳定的测试用例, 以及这些具有高失效度的测试用例是否有助于观测 RRT 算法的潜在问题.

3.2 实验设计

3.2.1 被测对象设置

我们用 Matlab 实现作为被测对象的 RRT 路径规划算法以及 DRT-PP 测试方法: 将任务场景 M 设为 200×200 的方形场景, 即 $X=200$, $Y=200$; 起始点和终止点分别设在左下角和右上角, 其坐标分别为 (5,5) 和 (195,195); 到达目标的判定阈值为 3, 即当智能体距离目标为 3 个单位时到达目标.

针对 RRT 算法, 我们将随机树生长的步长 (即公式 (5) 中的 *ag.r*) 设为 3. 通过实验表明, RRT 算法在路径规划过程中随机采样点的数量一般小于 10 000. 因此, 为了防止生成的无效测试用例 (如“死胡同”场景) 耗费大量的测试资源, 我们将随机采样点的上限设为 20 000. 此外, 一般情况下, RRT 最后生成的路径长度 $|pth|$ 是小于 150 的, 因此, 当 RRT 采样点数量超过 20 000 却仍没有找到可行路径时, 则令路径的长度为 200. 此外, RRT 生成的树结构

可以有很多种, 这里我们选择经典的二叉树结构.

3.2.2 DRT-PP 设置

对于 DRT-PP 方法, 我们将任务空间 M 的分割份数设为 20, 即令 $D_1 = D_2 = D = 20$. 如图 5 所示, 当对任务空间 M 做足够细致的划分时, 我们就能够涵盖多样的威胁分布, 如当 $D = 20$ 时可能的威胁分布情况 (即测试用例) 就有 2^{400} 种. 在测试开始时, 我们将初始测试剖面的特征矩阵设为 $Q_0 = (0.5)^{20\times 20}$, 即每个子区域包含威胁的概率为 0.5, 即纯随机测试 (见公式 (17)). 此外, DRT-PP 的可调参数就是每次测试用例执行过后测试剖面的更新幅度 ε, 根据已有 DRT 相关工作[18], 我们分别考察 0.05、0.1、0.15、0.2 这 4 个参数, 即测试剖面的融合率为 5%、10%、15%、20%. 这里, 我们用 DRT-05、DRT-10、DRT-15、DRT-20 分别代表这 4 个参数下的 DRT-PP 方法.

3.2.3 基线方法设置

本文采用随机测试 RT 作为 DRT-PP 的对比方法. 具体来说, 我们将测试剖面始终设置为 $Q_0 = (0.5)^{20\times 20}$, 这样就能对离散化的输入空间进行纯随机采样. 目前, 现有工作并没有针对路径规划算法提出整体性的测试方法, 而为了进一步凸显 DRT-PP 的优势, 我们基于自适应随机测试思想, 设计一种面向路径规划算法的自适应随机测试方法 ART, 并将其同样作为测试策略. 具体来说, 每当生成下一个测试用例 t_{N^T+1} 时, 我们先通过随机方法生成 N^{Can} 个候选测试用例, 并计算各测试用例到已生成测试用例集的距离, 最后选择距离已生成测试用例集最近的测试用例作为 t_{N^T+1}. 这里, 在计算候选测试用例 t^c 和已经生成测试用例集 T 的距离 $dis(t^c, T)$ 时, 我们分别计算 t^c 与 T 包含的所有测试用例 $t \in T$ 的距离, 并将距 t^c 最近的测试用例的距离作为 $dis(t^c, T)$ 的值. 而在计算两个测试用例 t_1 和 t_2 的距离时, 我们先根据公式 (19) 计算它们的威胁分布矩阵 A^{t_1} 和 A^{t_2}, 并将这两个矩阵差的 1-范数 (见公式 (18)) 作为 t_1 和 t_2 的距离. 可以看出, ART 同样旨在提升生成测试用例的多样性, 但其没有考虑场景威胁分布的连续性以及测试过程中对于失效测试用例的寻优性. 对于 ART 算法, 候选集的数量是可以调整的参数, 本文采用较为通用的配置, 即候选测试用例集包含的测试用例数量为 3 (记为 ART3), 同时我们也考察了候选测试用例集的数量为 5 时的情况 (记为 ART5).

3.2.4 测试过程设置

为了计算失效度 $S(t)$, 我们在实验过程中将每个测试用例 t 执行 10 次, 即令 $N^{\text{repeat}} = 10$, 并保存每次执行后的输出路径 $PTH^t = \{pth_1^t, pth_2^t, \ldots, pth_{10}^t\}$. 进一步地, 对于每一种测试策略 (包括 RT、ART、DRT-PP) 的每一次测试过程, 我们生成并执行 1 000 个测试用例 $t_1, t_2, \ldots, t_{1000}$, 并计算它们的失效度 $S(t_1), S(t_2), \ldots, S(t_{1000})$, 即令 $N^{\text{limit}} = 1000$. 无论是 RT 还是 DRT-PP 都具有随机性, 此外, 在执行 DRT-PP 时, 测试剖面的演变过程还受初始测试剖面下纯随机测试生成的测试用例 t_1 的影响, 因此, 对于每一个测试策略, 我们都重复执行 10 次, 并观测整体的测试效果. 总的来说, 包括 RT、ART (包含 ART3、ART5) 和 DRT-PP (包含 DRT-05、DRT-10、DRT-15、DRT-20) 在内, 共有 7 种测试配置, 每个配置下, 我们进行 10 次测试流程, 每次测试流程生成 1 000 个测试用例, 并且每个测试用例要被 RRT 算法执行 10 次, 因此, 路径规划算法总共被执行的次数为 700 000 次. 据统计, 每次 RRT 算法的执行时间约为 2 s, 执行一个测试用例的时间 (即重复 10 次) 约为 20 s, 因此, 单线程运行实验的总时间约为 16.2 天 (实际我们采用并行方式运行实验). 这里需指出, DRT-PP 方法在测试剖面更新时的计算时间通常小于 5 ms, 因此, DRT-PP 的计算时间远远小于测试用例的执行时间 (见第 2.4 节的讨论), 因此本节不对 DRT-PP 的效率做进一步讨论.

3.2.5 评价指标设置

本文通过计算失效度 $S(t)$ 来评价测试用例质量, 即我们希望生成失效度较大的测试用例. 因此, 一个测试流程完成时, 我们会得到 1 000 个测试用例的 $S(t)$ 值, 而我们通过计算失效度 $S(t)$ 的平均值来评价生成测试用例集的整体质量. 假设某测试策略生成的测试用例集为 $T = \{t_1, t_2, \ldots, t_{1000}\}$, 则 $S(t)$ 平均值可由公式 (23) 计算.

$$S^{\text{avg}} = \frac{\sum_{i=1}^{1000} S(t_i)}{1000} \tag{23}$$

此外, 对于每次测试, 我们同样考察 $S(t)$ 的最大值, 因为失效度 $S(t)$ 最大的测试用例最有可能暴露被测算法的失效风险. 同样对于测试用例集 T, $S(t)$ 的最大值可由公式 (24) 计算.

$$S^{\max} = \max_{t \in T} S(t) \tag{24}$$

此外, 为了回答 RQ2, 我们对不同测试方法生成测试用例的多样性进行评价. 由公式 (19) 可知, 一个测试用例 t 可以用矩阵 $A^t = (a_{ij})^{20\times 20}$ 表示, 其中, 每个 a_{ij} 表示该测试用例中子区域 M_{ij} 是否包含威胁. 基于此, 我们通过 1-范数 (即两个测试用例间不同元素的数量) 来度量两个测试用例之间的差异, 即两个测试用例差异可由公式 (25) 计算.

$$Dis(t_1, t_2) = \left\| A^{t_1} - A^{t_2} \right\| \tag{25}$$

对于整个测试用例集 T 的多样性, 我们先考虑那些 $S(t)$ 不为 0 的测试用例, 因为 $S(t) = 0$ 意味着该测试用例没有可行解, 是无效测试用例. 因此定义测试用例集 $T^{\neq 0} = \{t \in T | S(t) > 0\}$ 为 T 中那些 $S(t)$ 不为 0 的测试用例所组成的子集, 即:

$$T^{\neq 0} = \{t \in T | S(t) > 0\} \tag{26}$$

则测试用例集 T 的多样性程度 Div 被定义为有效测试用例间差异和的平均值, 即:

$$Div = \frac{\sum\limits_{i \neq j,\ t_i, t_j \in T^{\neq 0}} Dis(t_i, t_j)}{(|T^{\neq 0}|) \cdot (|T^{\neq 0}| - 1)} \tag{27}$$

由于无效测试用例不存在可行解, 因此即使无效测试用例与其他测试用例间存在差异, 也不能为被测算法的评估提供帮助, 因此我们将生成无效测试用例所浪费的测试资源考虑进去后, 得到整体测试用例集的多样性程度 Div^{all}, 其计算公式如下:

$$Div = \frac{\sum\limits_{i \neq j,\ t_i, t_j \in T^{\neq 0}} Dis(t_i, t_j)}{(|T|) \cdot (|T| - 1)}.$$

最后由于所有的测试方法都会运行 10 次, 我们还会通过 t 检验来观察不同测试方法之间的有效性差异是否显著. 需指出, 在软件测试领域, 通常通过需求的违反程度、测试用例多样性等指标来评价生成测试用例集的质量. 本文基于路径规划算法这一问题领域设计了失效度及多样性指标, 其本质是软件测试领域的评价指标在路径规划算法这一具体问题上的体现. 例如, 在路径规划算法中, 路径长度需求是基本的性能需求, 而公式 (23) 和公式 (24) 通过计算同一个测试用例生成路径长度的差异, 反映了测试用例的输出与最优路径的长度的距离下限, 进而反映了需求违反程度, 类似评价指标也在现有工作中被使用[13,19,52]. 在软件测试领域, 测试多样性通常可以通过量化同一测试用例集中不同测试用例间的差异来计算[53], 而公式 (25) 计算了不同测试用例在输入空间的差异性, 进而得到多样性度量, 现有工作也使用了类似评价指标[48].

3.3 实验结果与分析

本节展示 DRT-PP、ART 和 RT 的对比实验结果, 分析其有效性及相关因素的影响分析, 并依次回答第 3.1 节中提出的 3 个研究问题.

3.3.1 有效性分析

表 1 展示了不同测试方法在测试 RRT 路径规划算法时的有效性. 表格第 1 列为测试方法: RT 为随机测试; ART 为自适应随机测试, 其中, ART3 和 ART5 分别表示候选测试集中包含测试用例数量为 3 和 5 两种配置; DRT-XX 为本文提出的 DRT-PP 方法, 其中, DRT-05、DRT-10、DRT-15、DRT-20 分别表示测试剖面更新幅度为 0.05、0.1、0.15、0.20. 表格每一列代表不同的评价指标, 其中, S^{avg} 表示测试过程中的平均失效度, $S^{\max}$ 表示得到的最大失效度, “>40”表示测试过程中失效度大于 40 的测试用例个数, “=0”表示测试过程中失效度为 0 的个数.

从表 1 可以看出, 在所有配置参数下, DRT-PP 方法的各种评价指标均优于 RT. 具体来说, DRT-PP 生成测试用例的平均失效度为 16.4, 约为 RT (平均失效度为 6.5) 的 2.5 倍. 这说明, 整体上与 RT 相比, DRT-PP 能够生成失效度较高的测试用例. 在软件测试环节, 测试人员通常会将最能够暴露失效风险的测试用例选出来优先进行检查和分析, 因此我们分析整个测试过程中得到的最大失效度 $S^{\max}$. 从表中可以看出, 大多数 DRT-PP 实例都能够得到

大于 40 的最大失效度, DRT-PP 方法生成测试用例的平均最大失效度为 41.1, 优于 RT 的 37.7. 从实验结果可知, 一般情况下, 测试用例的最大失效度在 0–40 之间, 而失效度大于 40 的测试用例很少. 而如果某个测试用例的失效度大于 40, 则说明 RRT 算法生成路径的标准差在 40 个步长以上, 这对长宽均为 200 的场景来说是较大的. 从表中可知, 执行 DRT-PP 方法得到失效度大于 40 的测试用例数量平均为 2.9 个, 特别是 DRT-15 (测试剖面的更新幅度为 0.15), 平均数量达到了 6.7; 相对地, RT 生成失效度大于 40 的测试用例数量为 0. 这说明, 采用 DRT-PP 方法通过动态调整测试剖面, 确实能够逼近高失效度的测试用例. 最后, 如果测试用例的失效度为 0, 则表明该测试用例无解, 而生成这种无效测试用例是对测试资源的一种浪费, 因此我们还检查了各方法生成失效度等于 0 的测试用例个数. 从表中可以看出, 对于路径规划算法的测试来说, 生成有效测试用例并不是那么容易, 几乎所有测试方法都生成了数量较多的无效测试用例. 尽管如此, DRT-PP 平均生成的无效测试用例个数 159.8 个, 概率约为 16%, 即大部分情况下生成的测试用例是有效的. 相比之下, RT 生成无效测试用例的平均个数达到了 636.3, 这表明路径规划算法测试的无效输入空间远大于有效测试空间, 若采用纯随机测试方法, 则大概率生成无效测试用例. 而 DRT-PP 方法则能够通过调整测试剖面, 在很大程度上回避无效测试用例的生成.

表 1　RQ1、RQ3 不同方法测试 RRT 路径规划算法的有效性

方法	S^{avg}	S^{max}	>40	=0
RT	6.5	37.7	0.0	636.3
ART3	6.8	39.8	0.6	631.1
ART5	6.7	39.3	0.4	630.9
DRT-05	14.2	39.3	0.5	163.4
DRT-10	**17.7**	41.3	3.1	157.6
DRT-15	17.2	**43.6**	**6.7**	**150.8**
DRT-20	16.4	40.2	1.1	167.5
DRT平均	16.4	41.1	2.9	159.8

ART 作为本文提出的另一种基线方法, 其测试效果与 RT 相比略有提升. 从表 1 中可以看出, 在 ART3 和 ART5 这两个参数配置下, ART 的表现相差不大. 与 RT 相比, ART3 和 ART5 平均失效度 S^{avg} 从 6.5 分别提升至 6.8 和 6.7, 最大失效度 S^{max} 从 37.7 分别提升至 39.8 和 39.3, 失效度大于 40 的个数从 0.0 提升至 0.6 和 0.4, 而无效的测试场景仅仅从 636.3 减少至 631.5 和 630.9. 然而从整体上看, DRT-PP 方法对测试效果的提升会更加明显. 具体来说, 所有配置下的 DRT-PP 在平均失效度 S^{avg} 以及生成有效测试用例的个数上都明显好于 ART: DRT-PP 在 S^{avg} 以及“=0”上的平均值为 16.4 和 159.8, 而 ART3 为 6.8 (ART5 为 6.7) 和 631.5 (ART5 为 630.9). 此外, 在最大失效度 S^{max} 以及失效度大于 40 的个数上, 除了 DRT-05 由于测试剖面更新幅度为过小的原因导致其在 S^{max} 以及“>40”上略逊于 ART (其与 ART 基本持平), DRT-PP 整体好于 ART: DRT-PP 在 S^{max} 以及“>40”上的平均值为 41.1 和 2.9, 而 ART3 为 39.8 (ART5 为 39.3) 和 0.6 (ART5 为 0.4).

表 2 分别展示了不同配置下的 DRT-PP 算法与 RT 针对平均失效度 S^{avg} 和最大失效度 S^{max} 的统计分析结果. 具体来说, 对每种测试方法我们都执行 10 次测试流程, 并将每种测试方法得到的 10 组 S^{avg} 和 S^{max} 通过 t-检验进行两两比较. 表 2 中 DRT-05、DRT-10、DRT-15、DRT-20 这 4 列分别展示了当更新幅度为 0.05、0.10、0.15、0.20 时, DRT-PP 与 RT 的假设检验结果, 每个单元格括号中的 3 个值分别为统计量 T 值、p 值以及是否拒绝原假设 (即二者之间存在大小关系): 若显著程度为√, 则说明 DRT-PP 优于 RT; 若显著程度为×, 则说明 DRT-PP 劣于 RT; 若显著程度为≡, 则说明结果不显著. 从表中可以看出, 对于 S^{avg} 来说, 显著程度均为√, 且 p 值均为 0.00, 这表明在平均失效度方面 DRT-PP 明显好于 RT. 对于 S^{max} 来说, 除了 DRT-05 外, 其他 3 种配置下的 DRT-PP 方法均明显好于 RT; DRT-05 在 T 值方面也好于 RT, 但比较结果不是很显著. 表 3 与表 4 分别展示了不同配置下的 DRT-PP 算法与 ART3 和 ART5 针对平均失效度 S^{avg} 和最大失效度 S^{max} 的统计分析结果, 其中各行的含义与表 2 相同. 从这两个表中可以看出, 对于 S^{avg} 来说, 显著程度均为√, 且 p 值均为 0.00, 这表明在平均失效度方面 DRT-PP 明显好于 ART. 对于 S^{max} 来说, DRT-15 明显好于 ART3 和 ART5; DRT-20 和 DRT10 在 T 值方面均好于 ART3 和

ART5, 但除了 DRT10 与 ART5 的比较结果外, 其余结果不显著; DRT-15 略逊于 ART, 但比较结果不是很显著. 将来我们会用更多实验数据对其进行比较.

表 2 RQ1 不同 DRT-PP 配置与 RT 的对比的假设检验结果

准则	DRT-05	DRT-10	DRT-15	DRT-20
S^{avg}	(40.36, 0.00, √)	(15.81, 0.00, √)	(14.61, 0.00, √)	(22.11, 0.00, √)
S^{max}	(1.84, 0.09, ≡)	(4.52, 0.00, √)	(3.91, 0.00, √)	(3.37, 0.00, √)

表 3 RQ1 不同 DRT-PP 配置与 ART3 的对比的假设检验结果

准则	DRT-05	DRT-10	DRT-15	DRT-20
S^{avg}	(39.26, 0.00, √)	(15.45, 0.00, √)	(14.26, 0.00, √)	(21.55, 0.00, √)
S^{max}	(−0.52, 0.61, ≡)	(1.39, 0.18, ≡)	(2.31, 0.04, √)	(0.39, 0.70, ≡)

表 4 RQ1 不同 DRT-PP 配置与 ART5 的对比的假设检验结果

准则	DRT-05	DRT-10	DRT-15	DRT-20
S^{avg}	(39.51, 0.00, √)	(15.54, 0.00, √)	(14.35, 0.00, √)	(21.69, 0.00, √)
S^{max}	(−0.11, 0.92, ≡)	(2.25, 0.04, √)	(2.77, 0.02, √)	(1.05, 0.31, ≡)

图 6 展示了 DRT-PP (包含 DRT-05、DRT-10、DRT-15、DRT-20)、ART (包含 ART3、ART5) 与 RT 在执行过程中生成测试用例的失效度变化过程, 其中, 横坐标表示生成并执行的测试用例数量, 纵坐标表示失效度 $S(t)$. 为了便于观测, 我们将曲线进行了平滑处理 (浅色曲线为实际曲线). 从图中可以看出, 由于 RT 采用了纯随机策略, 其生成测试用例的失效度始终在一个比较低的值附近摆动; 而 ART 尽管试图通过增大随机采样的分散程度来提升测试用例的多样性, 但由于路径规划场景空间非常复杂, 因此无论是 ART3 还是 ART5, ART 失效度的变化都与 RT 类似, 始终在较低的值周围摆动; DRT-PP 算法得到的失效度尽管也有剧烈摆动, 但由于其考虑了路径规划的领域知识, 设计了基于测试剖面更新的寻优策略, 随着测试的进行, 其失效度平均值会逐渐增大.

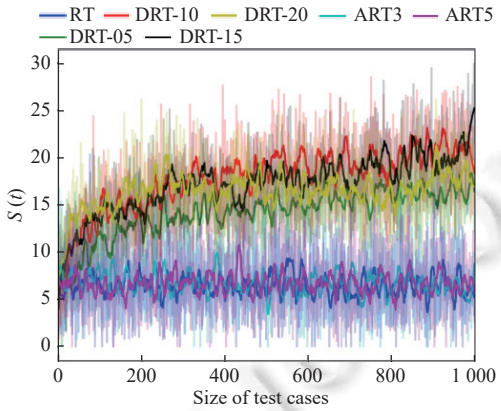

图 6 DRT-PP 与 RT 和 ART 在测试过程中失效度的变化

RQ1 结论: 实验结果表明, 动态随机测试 (DRT-PP) 方法在面向智能体路径规划算法的测试中优于随机测试 (RT) 以及自适应随机测试 (包括 ART3 和 ART5). 具体来说, DRT-PP 能够生成具有较高平均失效度的测试用例集, 能够发现更多具有高失效度的测试用例, 并且能够生成较少的无效测试用例. 此外, 随着测试的进行, DRT-PP 的测试剖面确实能够逐渐趋近于生成能够暴露潜在失效的测试用例.

3.3.2 多样性分析

表 5 展示了利用公式 (27) 对 RT、ART 和 DRT-PP 生成测试用例的多样性的评估结果. 可以看出, 若不计算

无效测试用例, 执行 RT 和 ART (包括 ART3 和 ART5) 后生成测试用例的多样性程度均为 14; 而 DRT-PP 生成测试用例的平均多样性程度在 8.0–10.0 之间, 平均值为 8.8. 可见, 由于 DRT-PP 提升了寻优性, 其生成测试用例的多样性程度相较于纯随机的 RT 有所下降, 但减小幅度有限. 表 6 展示了当考虑所有无效测试用例后, 即将无效测试用例的多样性记为 0 后, 各测试方法的多样性分析结果. 可以看出, 由于 RT 生成了大量无效测试用例, 因此在考虑了这些测试用例的负面影响后, 整体的多样性程度会大大降低: 从 14.0 降至 1.8; ART 尽管旨在提升测试用例的多样性, 但由于路径规划场景的复杂性, 仅仅通过测试用例间的距离进行筛选而不考虑测试剖面仍然有很大的可能生成无效的测试用例, 进而使整体的多样性程度大大降低, 这一现象在 ART3 和 ART5 上均有体现: 二者的多样性程度分别从 14.0 降至 1.9 和 1.6. 而 DRT-PP 通过随机搜索的方式在保证测试用例有效性的同时, 在一定程度上兼顾了测试用例的多样性, 因此, 即使考虑无效测试用例, DRT-PP 生成测试用例的多样性仍保持在 5.0–7.0 之间 (平均值为 6.2).

表 5 RQ2、RQ3 不同的测试方法生成测试用例的多样性 *Div* 比较 (排除无效测试用例)

方法	1	2	3	4	5	6	7	8	9	10	Avg.
RT	**14.0**	**14.0**	**14.0**	**14.0**	**14.0**	**14.0**	**14.0**	**14.0**	**14.0**	**14.0**	**14.0**
ART3	**14.0**	**14.0**	**14.0**	**14.0**	**14.0**	**14.0**	**14.0**	**14.0**	**14.0**	**14.0**	**14.0**
ART5	**14.0**	**14.0**	**14.0**	**14.0**	**14.0**	**14.0**	**14.0**	**14.0**	**14.0**	**14.0**	**14.0**
DRT-05	11.0	9.5	9.2	10.4	10.2	8.9	9.7	10.3	9.7	9.6	**9.9**
DRT-10	8.2	9.0	6.6	8.6	8.5	6.2	7.9	8.2	7.6	9.6	8.0
DRT-15	7.7	9.8	9.0	8.8	7.9	7.4	9.3	8.5	8.5	8.2	8.5
DRT-20	9.8	6.3	7.1	8.8	9.5	9.0	8.9	9.5	9.5	9.5	8.8
DRT-Avg	9.2	8.6	8.0	9.1	9.0	7.8	9.0	9.1	8.8	9.2	8.8

表 6 RQ2、RQ3 不同的测试方法生成测试用例的多样性 Div^{all} (包括无效测试用例)

方法	1	2	3	4	5	6	7	8	9	10	Avg.
RT	1.6	2.1	2.1	1.8	1.9	2.0	1.9	1.7	1.9	1.7	1.8
ART3	1.9	1.8	2.1	1.8	2.2	1.8	1.7	1.8	2.1	1.9	1.9
ART5	1.6	2.1	2.0	2.0	2.1	1.8	1.7	2.0	1.9	1.9	1.9
DRT-05	**8.8**	6.6	5.8	**7.8**	6.3	6.3	**6.7**	**7.3**	7.2	6.5	**6.9**
DRT-10	5.7	7.0	4.4	5.7	5.9	4.2	5.6	6.0	5.7	7.1	5.7
DRT-15	5.8	**7.3**	**5.9**	6.9	5.0	5.5	6.5	6.9	6.1	5.5	6.1
DRT-20	7.1	4.0	4.2	5.8	**6.7**	**6.5**	6.3	6.6	**7.3**	**6.8**	6.1
DRT-Avg	6.8	6.2	5.1	6.6	6.0	5.6	6.3	6.7	6.6	6.5	6.2

RQ2 结论: 实验结果表明, 基于纯随机采样的测试方法生成测试用例的多样性程度较高, 但不能弥补其生成大量无效测试用例带来的负面影响. 而 DRT-PP 方法由于提升了寻优性, 生成测试用例的多样性程度有所降低. 然而, DRT-PP 兼顾了测试用例的有效性和多样性, 因此其生成有效测试用例的多样性反而得到了提升.

3.3.3 DRT-PP 参数分析

表 1 同样比较了各测试剖面更新幅度下 DRT-PP 的测试效果. 从表中可以看出, 在各种参数下, DRT-PP 的整体测试效果相差不大. 针对平均失效度 S^{avg}, DRT-10 的表现最好 (17.7), DRT-15 其次 (17.2), DRT-05 表现相对较差 (14.2); 对于其他指标, 包括最大失效度 S^{max}、失效度大于 40 的测试用例数量、无效测试用例数量, 都是 DRT-15 表现更好; 这 3 个指标表现相对较差的 DRT-PP 配置分别是 DRT-05 (S^{max} = 39.3)、DRT-05 (平均失效度大于 40 的测试用例个数为 0.5)、DRT-20 (平均无效测试用例个数为 167.5). 为了进一步观察, 我们分别针对平均失效度 S^{avg} 和最大失效度 S^{max} 对 DRT-PP 的各个参数配置进行假设检验分析, t-检验结果如表 7 和表 8 所示. 表中的行和列分别为不同的 DRT-PP 配置参数: 行为方法 1, 列为方法 2. 与表 2 类似, 每个单元格的数据仍然依次为 T 值、p 值、显著程度. 显著性为√意味着方法 1 优于方法 2; 反之, 显著性为×意味着方法 2 优于方法 1. 从这两个表可以

看出, 在平均失效度 S^{avg} 方面, DRT-10、DRT-15、DRT-20 均优于 DRT-05; 在最大失效度 S^{max} 方面, DRT-15 优于 DRT-05 和 DRT-20.

表 7 RQ3 不同测试剖面更新幅度下 DRT-PP 的假设检验对比结果 (平均失效度 S^{avg})

方法	DRT-05	DRT-10	DRT-15	DRT-20
DRT-05	—	(−4.93, 0.00, ×)	(−4.04, 0.00, ×)	(−4.85, 0.00, ×)
DRT-10	(4.93, 0.00, √)	—	(0.56, 0.58, ≡)	(1.56, 0.14, ≡)
DRT-15	(4.04, 0.00, √)	(−0.56, 0.58, ≡)	—	(0.86, 0.40, ≡)
DRT-20	(4.85, 0.00, √)	(−1.56, 0.14, ≡)	(−0.86, 0.40, ≡)	—

表 8 RQ3 不同测试剖面更新幅度下 DRT-PP 的假设检验对比结果 (考虑最大失效度 S^{max})

方法	DRT-05	DRT-10	DRT-15	DRT-20
DRT-05	—	(−1.97, 0.06, ≡)	(−2.67, 0.02, ×)	(−0.96, 0.35, ≡)
DRT-10	(1.97, 0.06, ≡)	—	(−1.47, 0.17, ≡)	(1.13, 0.27, ≡)
DRT-15	(2.67, 0.02, √)	(1.47, 0.17, ≡)	—	(2.16, 0.05, √)
DRT-20	(0.96, 0.35, =)	(−1.13, 0.27, ≡)	(−2.16, 0.05, ≡)	—

总的来说, DRT-15, 即当测试剖面的更新幅度为 0.15 时, 测试效果较好, 这说明当我们将更新幅度设为 0.15 时, 既保证了寻优性, 也能够避免陷入局部最优. 相比之下, DRT-05 和 DRT-20 表现较差. 首先, 路径规划算法的输入空间复杂, 而测试剖面在调整过程中也会伴有较大的噪声, 这一点从图 6 中也可以观测到, 而 DRT-05 由于测试剖面调整幅度较小, 更新效果有可能被噪声淹没, 因此其发现失效度较大测试用例的概率就会少一些. 相对的, DRT-20 由于测试剖面调整幅度过大, 使得测试剖面收敛过快, 进而导致测试过程陷入局部最优, 或产生更多的无效测试用例.

表 5 和表 6 同样展示了各种 DRT-PP 配置下生成测试用例的多样性. 可以看出 DRT-05 虽然在寻优性上略逊于其他配置, 但由于其测试剖面调整程度小, 生成的测试用例具有较高的多样性. 实验结果表明, 无论是否考虑无效测试用例, DRT-05 生成测试用例的多样性程度都是最高的 (分别为 9.9 与 6.9).

RQ3 结论: 实验结果表明, 在不同配置下, DRT-PP 的测试效果相差不大. 对于寻优性来说, DRT-15 (即测试剖面更新幅度为 0.15) 的表现稍好, 而 DRT-05 和 DRT20 (即测试剖面更新幅度分别为 0.05 和 0.20) 的表现稍差. 考虑测试用例的多样性, DRT-05 的表现最好. 在实际应用时, 测试人员可以根据经验选取合适的测试剖面更新幅度. 在今后的研究中, 我们也会对 DRT-PP 的各种配置参数做进一步考察.

3.3.4 实例研究

在 RQ1、RQ2 和 RQ3 中, 我们研究了 DRT-PP 在测试 RRT 路径规划算法时的有效性. 本节我们通过实例评估, 对测试过程中发现的失效度最大的测试用例进行观察, 并讨论其在暴露算法潜在问题上的价值. 图 7 描述了测试过程中生成的失效度最大测试用例的威胁分布情况以及 RRT 算法在执行该测试用例时的生成路径. 该测试用例为 DRT-PP 在测试剖面更新幅度为 0.15 时生成的第 897 个测试用例, 这里我们将其记为 t_{897}. 执行 t_{897} 得到不同的路径, 经计算其失效度 $S(t_{897})$ 值达到了 54.

该测试用例的威胁分布如图 7 中的红色圆圈所示, 可以看出 DRT-PP 生成的测试用例 t_{897} 是一个由 168 个威胁构成了一个“多陷阱”场景, 而路径规划的起始点和终止点分别位于左上角和右下角. 直觉上, 该场景的一个较为合理的路径如图中绿色曲线所示: 智能体避开陷阱, 从场景的中间穿过, 到达终点. 现在我们将该测试用例用 RRT 算法运行 10 次, 并记录每次执行后的生成路径, 如图中蓝色线条所示. 在该例子中, 我们能够明显看到, 生成的路径之间存在很大的差异, 很多路径都走入了不同的“陷阱”中, 从而导致路径长度过长. 因此, 可以认为在该例子中, 经典的 RRT 算法表现出较为明显的不稳定性. 确实, 由于 RRT 基于构建随机树来生成路径, 一旦随机树的某个分支进入了“陷阱”并且优先走到距离目标点较近的位置, 那么这个分支将很可能被当作最终路径并无法被修正, 从而导致较长生成路径. 因此, RRT 算法在这种多陷阱场景中, 有可能存在性能风险.

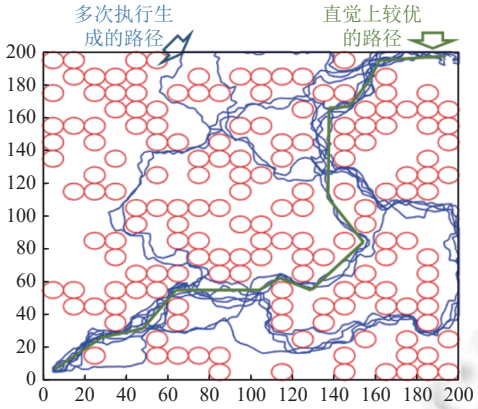

图 7 测试过程中 DRT-PP 生成的失效度最大的测试用例及其生成路径

4 风险分析

根据文献 [54] 中对实验风险的论述, 对本研究结论的有效性构成威胁的因素包括 4 个方面: 构念有效性 (construct validity)、结论有效性 (conclusion validity)、内部有效性 (internal validity) 以及外部有效性 (external validity). 本节从这 4 个方面来分析实验结论的有效性.

(1) 构念有效性. 由于路径规划算法的确认测试本身存在测试预期问题, 因此如何描述算法的性能需求并以此为依据制定测试结果的评判准则就是一大挑战. 本文借助蜕变测试思想, 基于在相同测试用例下多次执行被测路径规划算法所得到的路径差别不应很大这一启发式规则, 设计了度量准则 $S(t)$, 并将其作为失效度. 然而不同的实际任务对 $S(t)$ 值的需求可能不同, 因此我们很难将其与性能失效建立确切的联系. 为此, 本文通过 DRT-PP 方法试图找到尽可能大的 $S(t)$. 此外, 我们在实验中分别对平均失效度、最大失效度以及失效度大于 40 的测试用例数量进行了考察, 直觉上如果在 200×200 的场景中同一测试用例的所生成的路径步长相差 40, 则该测试用例对分析算法稳定性及寻优性是有价值的. 今后我们会对规划时间等其他性能需求做进一步研究, 并制定相关的算法性能评价准则. 此外, DRT-PP 方法还依赖于测试剖面的更新幅度 ε 为此我们在实验中通过 RQ3 分析了不同 ε 对测试结果的影响.

(2) 结论有效性. 由于基于随机采样的 RRT 路径规划算法是非确定性的, 而本文提出的 DRT-PP 以及作为基线方法的随机测试 RT 与自适应随机测试 ART 都带有随机性, 因此这些不确定性有可能会影响实验结果的有效性. 首先, 本文的测试方法本身就是面向 RRT 随机采样所带来的稳定性问题; 其次, 针对 RT、ART 和 DRT-PP 的不确定性, 我们对相关方法的各个配置都进行了多次重复实验, 并采用假设检验的方式对实验结果进行了分析. 今后的研究中我们还会持续收集更多的数据并持续细化分析结果.

(3) 内部有效性. 本文实现了路径规划问题的基本场景、RRT 算法, 并在此基础上实现了 RT、ART 和 DRT-PP, 我们对代码进行了反复检验和测试, 以确认其正确性. 例如, 当出现无法达到目标点的情况时, 我们会对规划场景进行反复检查, 以确认该现象由无效的测试用例所导致, 并排除了代码错误等问题.

(4) 外部有效性. 本文将目前路径规划系统通用的 RRT 算法作为被测对象, RRT 目前被广泛应用于各种路径规划问题中 (如机械臂、智能物流等), 并被当作很多实际规划策略的基础. 并且, RRT 基于随机的采样, 其不依赖于特定领域知识, 因此具备较强的普适性. 为了进一步检查外部有效性, 我们检查了 DRT-PP 在 RRT 算法不同参数下的表现. 具体来说, 我们将 RRT 算法中生成随机树的最大孩子节点数设为 3 (记为 RRT3), 并在此配置下分别运行 DRT-PP 和 RT (这里我们选取 DRT-15 配置). 结果显示, 各测试方法在 RRT 和 RRT3 上的表现差距不大. 当被测算法由 RRT 替换为 RRT3 时, 对于 RT 来说, 平均失效度 (S^{avg})、最大失效度 (S^{max})、失效度大于 40 的个数 (>40) 分别从 6.5、37.7、0.0 变为 7.0、39.7、0.4, 表现略有提升; 而生成无效测试用例的个数二者持平 (均为

636.3). 与 RRT 相比, DRT 在 RRT3 上的平均失效度 (S^{avg})、最大失效度 (S^{max})、失效度大于 40 的个数 (>40)、无效测试用例个数分别从 17.2、43.6、6.7、150.8 变为 18.3、42.1、17.1、159.1, 除了最大失效度略有下降外, 其余指标均有所提升. 可见, 当 RRT 的生成树最大孩子节点为 3 (即三叉树) 时, RRT 算法性能有所下降, 使得无论运行 RT 还是 DRT, 都更容易发现具有较大失效度的测试用例. 此外, RT 和 DRT 的对比结果并没有明显变化, 即 DRT 仍然明显好于 RT.

此外, 本文采用 200×200 的基本场景来规划路径规划算法, 该设计旨在对任务空间进行离散处理, 实际使用时离散化的精度可以根据具体需求而定, 而本文所采用的步长及离散化精度与目前路径规划算法的设计工作是一致的 (通常选取场景边界的 1/10–1/20 作为步长)[4,16,55]. 最后, 本文针对二维空间进行测试, 二维空间假设是大部分关于路径规划算法研究的基本假设. 由第 2.4 节的讨论可知, 无论空间的维度如何, 通过 DRT 算法生成测试用例的代价都要远小于执行路径规划算法的代价, 因此其执行代价可以在某种程度上忽略. 在今后的研究中, 我们会对更多算法 (如 RRT*、PRM 等) 进行测试. 此外, 我们还会进一步考察任务空间大小以及规划维度对 DRT 测试效率的影响.

5 总结与展望

路径规划算法旨在规划智能体的行为轨迹, 使其在不碰到威胁障碍物的情况下安全且高效地从起始点到达目标点. 随着物理信息系统 (CPS) 的发展, 这些算法在工业界和日常生活中得到广泛应用, 其失效也会带来巨大损失. 然而, 由于任务场景中威胁分布的多样性, 如何生成测试用例以触发这些算法的性能风险就成为一大挑战. 为了解决上述挑战, 本文借鉴自适应随机测试思想, 提出了面向智能体路径规划算法的动态随机测试策略 (DRT-PP). 具体来说, 我们通过对输入空间的离散化以及引入子区域的威胁概率来构建测试剖面, 并在测试过程中通过实时提取和分析测试信息对测试剖面进行动态调整, 使得我们在得到多样的测试用例的同时, 能够触发更高的失效度, 从而提升测试效率. 实验结果表明, 与传统随机测试 (RT) 以及本文提出的 ART 测试相比, DRT-PP 能够更早地发现多样且有价值的测试用例; DRT-PP 生成测试用例的平均失效度约为 RT 的 2.5 倍、ART 的 2.4 倍.

在未来工作中, 我们将对 DRT-PP 进行更深入的探究, 包括更多的测试剖面构建方法以及测试剖面调整幅度的自适应更新等. 此外, 我们还将展开专门的实验研究, 对更多路径规划算法 (如 RRT*、PRM) 进行考察, 并提出更多的功能及性能需求, 以全面评估 DRT-PP 的适用性.

References:

[1] Omeiza D, Webb H, Jirotka M, Kunze L. Explanations in autonomous driving: A survey. IEEE Trans. on Intelligent Transportation Systems, 2022, 23(8): 10142–10162. [doi: 10.1109/TITS.2021.3122865]

[2] Zhang XY, Arcaini P, Ishikawa F. An incremental approach for understanding collision avoidance of an industrial path planner. IEEE Trans. on Dependable and Secure Computing, 2023, 20(4): 2713–2730. [doi: 10.1109/TDSC.2022.3159773]

[3] Mohsan SAH, Othman NQH, Li YL, Alsharif MH, Khan MA. Unmanned aerial vehicles (UAVs): Practical aspects, applications, open challenges, security issues, and future trends. Intelligent Service Robotics, 2023, 16(1): 109–137. [doi: 10.1007/s11370-022-00452-4]

[4] LaValle SM. Planning Algorithms. Cambridge: Cambridge University Press, 2006.

[5] Li JY, Tinka A, Kiesel S, Durham JW, Kumar TKS, Koenig S. Lifelong multi-agent path finding in large-scale warehouses. In: Proc. of the 35th AAAI Conf. on Artificial Intelligence. Virtually: AAAI, 2021. 11272–11281. [doi: 10.1609/aaai.v35i13.17344]

[6] Colas F, Mahesh S, Pomerleau F, Liu M, Siegwart R. 3D path planning and execution for search and rescue ground robots. In: Proc. of the 2013 IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems. Tokyo: IEEE, 2013. 722–727. [doi: 10.1109/IROS.2013.6696431]

[7] Fan QJ, Wang FX, Shen XQ, Luo DL. Path planning for a reconnaissance UAV in uncertain environment. In: Proc. of the 12th IEEE Int'l Conf. on Control and Automation (ICCA). Kathmandu: IEEE, 2016. 248–252. [doi: 10.1109/ICCA.2016.7505284]

[8] Cabreira TM, Brisolara LB, Jr Ferreira PR. Survey on coverage path planning with unmanned aerial vehicles. Drones, 2019, 3(1): 4. [doi: 10.3390/drones3010004]

[9] Wu L, Xi ZY, Zheng Z, Li XL. Application of metamorphic testing on UAV path planning software. Journal of Systems and Software, 2023, 204: 111769. [doi: 10.1016/j.jss.2023.111769]

[10] Li R, Liu H, Lou GN, Zheng X, Liu X, Chen TY. Metamorphic testing on multi-module UAV systems. In: Proc. of the 36th IEEE/ACM

Int'l Conf. on Automated Software Engineering (ASE). Melbourne: IEEE, 2021. 1171–1173. [doi: 10.1109/ASE51524.2021.9678841]

[11] Schmidt T, Pretschner A. StellaUAV: A tool for testing the safe behavior of UAVs with scenario-based testing (tools and artifact track). In: Proc. of the 33rd IEEE Int'l Symp. on Software Reliability Engineering (ISSRE). Charlotte: IEEE, 2022. 37–48. [doi: 10.1109/ISSRE55969.2022.00015]

[12] Liu Y, Zheng Z, Qin FY. Homotopy based optimal configuration space reduction for anytime robotic motion planning. Chinese Journal of Aeronautics, 2021, 34(1): 364–379. [doi: 10.1016/j.cja.2020.09.036]

[13] Zhang JT, Zheng Z, Yin BB, Qiu K, Liu Y. Testing graph searching based path planning algorithms by metamorphic testing. In: Proc. of the 24th IEEE Pacific Rim Int'l Symp. on Dependable Computing. Kyoto: IEEE, 2019. 158–167. [doi: 10.1109/PRDC47002.2019.00046]

[14] Arcuri A, Iqbal MZ, Briand L. Random testing: Theoretical results and practical implications. IEEE Trans. on Software Engineering, 2012, 38(2): 258–277. [doi: 10.1109/TSE.2011.121]

[15] Gammell JD, Barfoot TD, Srinivasa SS. Batch informed trees (BIT*): Informed asymptotically optimal anytime search. The Int'l Journal of Robotics Research, 2020, 39(5): 543–567. [doi: 10.1177/0278364919890396]

[16] Karaman S, Walter MR, Perez A, Frazzoli E, Teller S. Anytime motion planning using the RRT*. In: Proc. of the 2011 IEEE Int'l Conf. on Robotics and Automation. Shanghai: IEEE, 2011. 1478–1483. [doi: 10.1109/ICRA.2011.5980479]

[17] Solovey K, Janson L, Schmerling E, Frazzoli E, Pavone M. Revisiting the asymptotic optimality of RRT. In: Proc. of the 2020 IEEE Int'l Conf. on Robotics and Automation (ICRA). Paris: IEEE, 2020. 2189–2195. [doi: 10.1109/ICRA40945.2020.9196553]

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

[19] Zhang XY, Liu Y, Arcaini P, Jiang MY, Zheng Z. MET-MAPF: A metamorphic testing approach for multi-agent path finding algorithms. ACM Trans. on Software Engineering and Methodology, 2024, 33(8): 1–37. [doi: 10.1145/3669663]

[20] Zhang XD, Cai Y. Building critical testing scenarios for autonomous driving from real accidents. In: Proc. of the 32nd ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. Seattle: ACM, 2023. 462–474. [doi: 10.1145/3597926.3598070]

[21] Singh S, Padhi R. Automatic path planning and control design for autonomous landing of UAVs using dynamic inversion. In: Proc. of the 2009 American Control Conf. St. Louis: IEEE, 2009. 2409–2414. [doi: 10.1109/ACC.2009.5160444]

[22] Okumura K, Machida M, Défago X, Tamura Y. Priority inheritance with backtracking for iterative multi-agent path finding. Artificial Intelligence, 2022, 310: 103752. [doi: 10.1016/j.artint.2022.103752]

[23] Lin HI, Hsieh MF. Robotic arm path planning based on three-dimensional artificial potential field. In: Proc. of the 18th Int'l Conf. on Control, Automation and Systems (ICCAS). Pyeongchang: IEEE, 2018. 740–745.

[24] Zhang B, Tang L, Roemer M. Probabilistic weather forecasting analysis for unmanned aerial vehicle path planning. Journal of Guidance, Control, and Dynamics, 2014, 37(1): 309–312. [doi: 10.2514/1.61651]

[25] Arcaini P, Zhang XY, Ishikawa F. Targeting patterns of driving characteristics in testing autonomous driving systems. In: Proc. of the 14th IEEE Conf. on Software Testing, Verification and Validation (ICST). Porto de Galinhas: IEEE, 2021. 295–305. [doi: 10.1109/ICST49551.2021.00042]

[26] Wang CB, Wang L, Qin J, Wu ZZ, Duan LH, Li ZQ, Cao MQ, Ou XC, Su X, Li WG, Lu ZJ, Li MJ, Wang YL, Long JJ, Huang ML, Li YH, Wang QH. Path planning of automated guided vehicles based on improved A-Star algorithm. In: Proc. of the 2015 IEEE Int'l Conf. on Information and Automation. Lijiang: IEEE, 2015. 2071–2076. [doi: 10.1109/ICInfA.2015.7279630]

[27] Hu Y, Harabor D, Qin L, Yin QJ. Regarding goal bounding and jump point search. Journal of Artificial Intelligence Research, 2021, 70: 631–681. [doi: 10.1613/jair.1.12255]

[28] Reif JH. Complexity of the mover's problem and generalizations. In: Proc. of the 20th Annual Symp. on Foundations of Computer Science. San Juan: IEEE, 1979. 421–427. [doi: 10.1109/SFCS.1979.10]

[29] Hu YR, Yang SX. A knowledge based genetic algorithm for path planning of a mobile robot. In: Proc. of the 2004 IEEE Int'l Conf. on Robotics and Automation, 2004. New Orleans: IEEE, 2004. 4350–4355. [doi: 10.1109/ROBOT.2004.1302402]

[30] Huang C, Zhou XB, Ran XJ, Wang JM, Chen HY, Deng W. Adaptive cylinder vector particle swarm optimization with differential evolution for UAV path planning. Engineering Applications of Artificial Intelligence, 2023, 121: 105942. [doi: 10.1016/j.engappai.2023.105942]

[31] Taylor B, Choi A. Fuzzy ant colony algorithm for terrain following optimization. In: Proc. of the 2014 IEEE Int'l Conf. on Systems, Man, and Cybernetics (SMC). San Diego: IEEE, 2014. 3834–3839. [doi: 10.1109/SMC.2014.6974528]

[32] Penrose M. Random Geometric Graphs. Oxford: Oxford University Press, 2003.

[33] Chandler B, Goodrich MA. Online RRT* and online FMT*: Rapid replanning with dynamic cost. In: Proc. of the 2017 IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems (IROS). Vancouver: IEEE, 2017. 6313–6318. [doi: 10.1109/IROS.2017.8206535]

[34] Orso A, Rothermel G. Software testing: A research travelogue (2000–2014). In: Future of Software Engineering Proc. Hyderabad: ACM, 2014. 117–132. [doi: 10.1145/2593882.2593885]

[35] Chen TY, Kuo FC, Liu H, Poon PL, Towey D, Tse TH, Zhou ZQ. Metamorphic testing: A review of challenges and opportunities. ACM Computing Surveys (CSUR), 2018, 51(1): 4. [doi: 10.1145/3143561]

[36] Jia Y, Harman M. An analysis and survey of the development of mutation testing. IEEE Trans. on Software Engineering, 2011, 37(5): 649–678. [doi: 10.1109/TSE.2010.62]

[37] Chen TY, Merkel R. An upper bound on software testing effectiveness. ACM Trans. on Software Engineering and Methodology (TOSEM), 2008, 17(3): 16. [doi: 10.1145/1363102.1363107]

[38] Chen TY, Leung H, Mak IK. Adaptive random testing. In: Advances in Computer Science—ASIAN 2004. Chiang Mai: Springer, 2005. 320–329. [doi: 10.1007/978-3-540-30502-6_23]

[39] Kuo FC, Chen TY, Liu H, Chan WK. Enhancing adaptive random testing in high dimensional input domains. In: Proc. of the 2007 ACM Symp. on Applied Computing. Seoul: ACM, 2007. 1467–1472. [doi: 0.1145/1244002.1244316]

[40] Mao CY, Chen TY, Kuo FC. Out of sight, out of mind: A distance-aware forgetting strategy for adaptive random testing. Science China Information Sciences, 2017, 60(9): 092106. [doi: 10.1007/s11432-016-0087-0]

[41] Cai KY, Chen TY, Li YC, Ning WY, Yu YT. Adaptive testing of software components. In: Proc. of the 2005 ACM Symp. on Applied Computing. Santa Fe: ACM, 2005. 1463–1469. [doi: 10.1145/1066677.1067011]

[42] Sun CA, Dai HP, Wang G, Towey D, Chen TY, Cai KY. Dynamic random testing of web services: A methodology and evaluation. IEEE Trans. on Services Computing, 2022, 15(2): 736–751. [doi: 10.1109/TSC.2019.2960496]

[43] Arcaini P, Zhang XY, Ishikawa F. Less is more: Simplification of test scenarios for autonomous driving system testing. In: Proc. of the 2022 IEEE Conf. on Software Testing, Verification and Validation (ICST). Valencia: IEEE, 2022. 279–290. [doi: 10.1109/ICST53961.2022.00037]

[44] Luo YX, Zhang XY, Arcaini P, Jin Z, Zhao HY, Ishikawa F, Wu RX, Xie T. Targeting requirements violations of autonomous driving systems by dynamic evolutionary search. In: Proc. of the 36th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). Melbourne: IEEE, 2021. 279–291. [doi: 10.1109/ASE51524.2021.9678883]

[45] Gambi A, Nguyen V, Ahmed J, Fraser G. Generating critical driving scenarios from accident sketches. In: Proc. of the 2022 IEEE Int'l Conf. on Artificial Intelligence Testing (AITest). Newark: IEEE, 2022. 95–102. [doi: 10.1109/AITest55621.2022.00022]

[46] Zhu Y, Xu ZG, Zhao XM, Wang RM, Qu XB. TsGAN-based automatic generation algorithm of lane-change cut-in test scenarios on expressways for autonomous vehicles. Journal of South China University of Technology (Natural Science Edition), 2024, 52(8): 76–88 (in Chinese with English abstract). [doi: 10.12141/j.issn.1000-565X.230229]

[47] Xia CY, Huang S, Zheng CY, Zhang QR, Wang Y, Wei YH. Modeling and verification method of intersection test scenario for automated driving. Ruan Jian Xue Bao/Journal of Software, 2023, 34(7): 3002–3021 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6855.htm [doi: 10.13328/j.cnki.jos.006855]

[48] Liu Y, Zhang XY. Adaptive random testing for multiagent path finding systems. IEEE Trans. on Reliability, 2022, 71(1): 295–308. [doi: 10.1109/TR.2022.3146323]

[49] Fan YR, Xia X, Lo D, Hassan AE, Wang Y, Li SP. A differential testing approach for evaluating abstract syntax tree mapping algorithms. In: Proc. of the 43rd IEEE/ACM Int'l Conf. on Software Engineering (ICSE). Madrid: IEEE, 2021. 1174–1185. [doi: 10.1109/ICSE43902.2021.00108]

[50] Zheng YY, Dou WS, Wang YC, Qin Z, Tang L, Gao Y, Wang D, Wang W, Wei J. Finding bugs in gremlin-based graph database systems via randomized differential testing. In: Proc. of the 31st ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. Virtual: ACM, 2022. 302–313. [doi: 10.1145/3533767.3534409]

[51] Barr ET, Harman M, McMinn P, Shahbaz M, Yoo S. The oracle problem in software testing: A survey. IEEE Trans. on Software Engineering, 2015, 41(5): 507–525. [doi: 10.1109/TSE.2014.2372785]

[52] Laurent T, Arcaini P, Zhang XY, Ishikawa F. Metamorphic testing of an autonomous delivery robots scheduler. In: Proc. of the 2024 IEEE Conf. on Software Testing, Verification and Validation (ICST). Toronto: IEEE, 2024. 361–372. [doi: 10.1109/ICST60714.2024.00040]

[53] Feldt R, Poulding S, Clark D, Yoo S. Test set diameter: Quantifying the diversity of sets of test cases. In: Proc. of the 2016 IEEE Int'l Conf. on Software Testing, Verification and Validation (ICST). Chicago: IEEE, 2016. 223–233. [doi: 10.1109/ICST.2016.33]

[54] Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B, Wesslén A. Experimentation in Software Engineering. 2nd ed., Berlin: Springer, 2024. [doi: 10.1007/978-3-662-69306-3]

[55] Liu Y, Zhu XM, Zhang XY, Xiao JN, Yu XH. RGG-PSO+: Random geometric graphs based particle swarm optimization method for

UAV path planning. Int'l Journal of Computational Intelligence Systems, 2024, 17(1): 127. [doi: 10.1007/s44196-024-00511-x]

附中文参考文献:

[46] 朱宇, 徐志刚, 赵祥模, 王润民, 曲小波. 基于 TsGAN 的自动驾驶汽车高速公路变道切入测试场景自动生成算法. 华南理工大学学报 (自然科学版), 2024, 52(8): 76–88. [doi: 10.12141/j.issn.1000-565X.230229]

[47] 夏春艳, 黄松, 郑长友, 张清睿, 王宇, 魏瑀皓. 自动驾驶交叉路口测试场景建模及验证方法. 软件学报, 2023, 34(7): 3002–3021. http://www.jos.org.cn/1000-9825/6855.htm [doi: 10.13328/j.cnki.jos.006855]

张逍怡(1988－), 男, 博士, 副教授, 主要研究领域为软件测试, 故障定位, 智能体路径规划.

郑征(1980－), 男, 博士, 教授, 博士生导师, CCF 专业会员, 主要研究领域为复杂软件可靠性, 失效机理分析, 高安全软件的适航管理.

李幸(2000－), 女, 硕士生, 主要研究领域为软件测试.

孙昌爱(1974－), 男, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为软件测试, 故障定位, 服务计算.

刘洋(1990－), 男, 博士, 讲师, CCF 专业会员, 主要研究领域为面向多智能体服务系统设计与优化及其可靠性测试技术.