基于深度强化学习的网络攻击路径规划方法

高文龙, 周天阳, 赵子恒, 朱俊虎

信息工程大学 郑州 中国 450001

摘要 攻击路径规划对实现自动化渗透测试具有重要意义, 在现实环境中攻击者很难获取全面准确的网络及配置信息, 面向未知渗透测试环境下的攻击路径规划, 提出了基于深度强化学习的攻击路径规划方法。首先, 对渗透测试问题的状态空间和动作空间进行形式化描述, 引入信息收集动作增强对环境的感知能力。然后, 智能体通过与环境的自主交互进行学习, 寻找最大化长期收益的最优策略, 从而指导攻击者进行路径规划。当前深度强化学习算法应用于攻击路径规划存在适应性不强和收敛困难等问题, 限制了其处理复杂渗透测试环境的能力。智能体在训练初期通过盲目探索得到的动作序列在维度迅速增长时质量会急剧下降, 有时很难完成目标, 而且低质量的动作序列大量积累会导致算法不收敛甚至神经元死亡。针对此问题, 本文提出的深度强化学习算法在 DDQN 算法的基础上增加了路径启发信息和深度优先渗透的动作选择策略。路径启发信息充分利用历史经验, 在训练初期对智能体的学习过程加以引导, 深度优先渗透的动作选择策略在一定程度上对动作空间进行了剪枝, 加速智能体的学习过程。最后, 通过与其他深度强化学习算法在相同实验条件下的对比, 验证了本文算法收敛速度更快, 运行时间缩短 30%以上。

关键词 深度强化学习; 路径启发信息; 深度优先渗透的动作选择策略; 攻击路径规划

中图法分类号 TP393.0 DOI 号 10.19363/J.cnki.cn10-1380/tn.2022.09.06

Network Attack Path Planning Method based on Deep Reinforcement Learning

GAO Wenlong, ZHOU Tianyang, ZHAO Ziheng, ZHU Junhu

Information Engineering University, Zhengzhou 450001, China

Abstract Attack path planning is of great significance to the realization of automated penetration testing. In a real environment, it is difficult for an attacker to obtain comprehensive and accurate network and configuration information. For network attack path planning in unknown penetration test environment, a path planning method based on deep reinforcement learning is proposed. First, the state space and action space of the penetration test problem are formally described, and the information collection action is introduced to enhance the perception of the environment. Then, the agent learns through autonomous interaction with the environment to find the optimal strategy for maximizing long-term benefits, so as to guide the attacker to plan the path. The current deep reinforcement learning algorithm applied to attack path planning has problems such as poor adaptability and difficulty in convergence, which limit its ability to handle complex penetration testing environments. The quality of the action sequence obtained by the agent through blind exploration in the initial training stage will drop sharply when the dimensionality increases rapidly, and sometimes it is difficult to complete the goal. Moreover, the large accumulation of low-quality action sequences will cause the algorithm to fail to converge and even neuron death. In response to this problem, the deep reinforcement learning algorithm proposed in this paper adds path-heuristic information and the action selection strategy of depth-first penetration on the basis of the DDQN algorithm. Path-heuristic information makes full use of historical experience to guide the learning process of the agent in the early stage of training. The action selection strategy of depth-first penetration prunes the action space to a certain extent, accelerating the learning process of the agent. Finally, through comparison with other deep reinforcement learning algorithms under the same experimental conditions, it is verified that the algorithm in this paper converges faster and the running time is shortened by more than 30%.

Key words deep reinforcement learning; path-heuristic information; the action selection strategy of depth-first penetration; attack path planning

通讯作者: 周天阳, 博士, 副教授, Email: aipteamzhouty@aliyun.com

收稿日期: 2021-09-23; 修改日期: 2021-11-22; 定稿日期: 2022-07-19

1 引言

随着信息技术的发展, 网络系统日益复杂化、多样化, 网络风险也在不断上升。传统的安全工具如防火墙和杀毒软件等难以发现和应对潜在脆弱点及其组合带来的安全风险, 渗透测试通过模拟攻击者在真实环境下的攻击意图和行为, 对整个测试系统进行全面的利用和评估, 帮助发现潜在的攻击链条, 逐渐成为评估网络安全的重要手段。渗透测试是一种专业化程度很高的活动, 需要测试人员具备多种安全知识和技能, 这些经验需要反复实践积累。同时, 渗透测试往往会持续较长时间, 几天或数周, 甚至需要对网络系统进行反复测试和检查维护。如果完全依赖人工实施渗透测试, 一是专家经验不易学习获取, 二是需要消耗大量时间精力进行反复测试。实施自动化渗透测试可以摆脱对专家经验的依赖, 节省时间和人力成本, 提高渗透测试效率[1-2]。攻击路径自动规划是实现自动化渗透测试的关键所在, 目前成为研究热点。

攻击路径规划是指从攻击者角度出发, 通过选择有效的动作序列从而到达既定的目标状态。攻击路径规划技术已有很多相关研究, 根据可解决的环境模型是否为完全可观察、确定、静态等, 大致分为确定性规划技术和不确定性规划技术[3]。确定性规划技术的基本思路是首先将渗透测试问题转化为规划域描述如规划领域定义语言(Planning Domain Definition Language, PDDL), 然后借助智能规划算法求得可行规划序列。经典的智能规划算法通过前向或后向搜索寻找规划解, 包括规划图技术[4]、偏序规划技术[5]、分层任务网络[6]等, 在搜索的过程中借助各种剪枝策略缩减搜索空间。此外, 组合优化算法也广泛应用于路径规划领域, 如模拟退火算法[7]、遗传算法[8]、粒子群算法[9]、蚁群算法[10]等, 这些算法通过迭代寻优的方式进行搜索, 可以在设定优化目标的情况下得到高质量规划解。上述规划算法通常用于求解经典规划问题, 需要设计好精确的搜索空间和算法模型。而攻击者在现实情况下很难获得完全准确的环境信息, 因此研究非确定性规划技术具有重要意义。非确定性规划技术主要有两种解决思路: 一是通过引入概率模拟环境不确定性, 然后结合确定性规划算法进行求解。Sarraute 等人[11]设计了一种规划模型, 将动作结果的不确定性建模为概率, 首先借助攻击树计算复合概率和预期时间, 然后使用 Dijkstra 算法进行求解得到规划路径。HU Tairan 等人[12]使用攻击成功概率表示环境的不确定程度, 基于启发式搜索提出了 APU-D*算法, 在规划失败的时候通过增量重规划对规划解进行快速调整。虽然在一定程度上解决了动作效果不确定性带来的挑战, 但攻击者仍然需要获取完整的拓扑连接等环境信息。二是基于马尔可夫决策过程(Markov Decision Process, MDP)或部分可观察马尔可夫决策过程(Partially Observable Markov Decision Process, POMDP)模型对渗透测试过程中的不确定性进行描述。MDP 模型以最大化长期受益为目标, 将行动表示为状态之间的转移, 可以更方便刻画动作效果的不确定性。POMDP 模型在 MDP 的基础上增加了观察空间和观察函数, 刻画不确定环境的理论更加完备, 而精确求解却面临着巨大挑战, 学术界一般使用近似方法进行求解, 可解决的问题规模仍然非常有限[13]。

基于 MDP 模型的强化学习技术[14]通过与环境的交互进行学习, 最终生成动作策略, 可以在任何给定状态下采取最大化长期收益的动作。强化学习可以帮助攻击者在环境知识不完整的条件下进行路径规划, 同时在计算上比起 POMDP 模型更具有可行性, 具有广阔的应用前景。Schwartz 等人[15]设计了一个轻量级的网络攻击模拟器(Network Attack Simulator, NSA), 将渗透测试框架表示为 MDP 模型, 基于此使用标准的强化学习算法如 Q-leaning 寻找最优攻击路径, 但是作者只使用了标准算法在小规模网络环境下进行了测试, 模拟器的精细程度及算法可扩展性有待提高。后续有学者在此基础上对强化学习算法进行了改进, 周仕承等人[16]提出了一种改进的强化学习算法 Noisy-Double-Dueling DQN, 通过融合优先经验回放、双重 Q 网络、竞争网络机制和噪声扰动机制提升了 DQN 算法在路径规划问题上的收敛速度; Nguyen H V 等人[17]提出了一种双智能体(Agent)架构的 A2C 算法, 两个 Agent 分别负责网络结构发现和主机服务探测利用, 可以解决大规模网络环境的攻击规划问题, 但是未详细阐明两个 Agent 的交互细节。这种融合了多种网络结构的改进方法在一定程度提高了算法性能。Zennaro F M 等人[18]采用新的思路求解强化学习问题, 针对每一种具体的渗透测试仿真环境, 分别提供不同形式的先验知识如延迟加载等, 帮助智能体学习更复杂的问题。文中仿真环境的设计非常典型精细, 不足之处在于实验环境仅有两台主机, 没有在网络中进行攻击规划。

当前深度强化学习算法应用于攻击路径规划存在适应性不强和收敛困难等问题, 限制了深度强化学习算法处理复杂渗透测试环境的能力。Agent 在训

练初期通过盲目探索得到的动作序列在维度迅速增长时质量会急剧下降, 有时很难完成目标。长期低质量路径的积累会导致算法不收敛甚至神经元死亡。针对此类问题, 本文提出了 DH_DDQN 算法, 在 Double_DQN 算法基础上增加路径启发信息和基于深度优先渗透的动作选择策略。路径启发信息充分利用历史经验, 在训练初期对 Agent 的学习过程加以引导, 基于深度优先渗透的动作选择策略在一定程度上对动作空间进行了剪枝。最后, 在不同规模的渗透测试环境下对算法进行了测试, 验证了算法的有效性。

2 强化学习

强化学习根据设计者的意图来训练 Agent, 通过与环境的交互学习策略, 发现能带来最大累积奖励的动作序列。不需要预先给定数据集, Agent 根据执行某一动作后环境的反馈进行学习, 通过不断试错积累经验。强化学习常用的理论框架是 MDP, MDP 模型通过四元组 $<S,A,R,T>$ 进行刻画, 其中, S 表示有限状态集; A 表示可执行的动作集; 奖励函数 R 表示在状态 s 执行某一动作 a 后获得的奖赏或惩罚 $r(s'|s,a)$, 定义为 $R:S\times A\to R$; 状态转移函数 T 表示在状态 s 执行某一动作 a 后转移至下一状态 s' 的概率 $p(s'|s,a)$。MDP 的目标是依赖最大化长期累积奖赏 $J_\pi(s_0)$ 获得最优策略 π^*。在公式(1)和(2)中, γ 为折扣因子; s_0 表示初始状态; 策略 π 表示根据当前状态 s 选择动作 a, 即 $\pi(s)=a$; L 表示从 s_0 开始依据 π 决策直至达到目标状态的步数。

$$J_\pi(s_0)=E[\sum_{l=0}^{L-i}\gamma^l r_l\,|\,s_0,\pi] \tag{1}$$

$$\pi^*=\arg\max_\pi J_\pi(s_0) \tag{2}$$

目前基于值函数和策略梯度的求解方法是解决强化学习问题的核心基础方法。网络攻击路径规划领域通常是离散空间, 本文考虑使用基于值函数的求解算法。基于值函数的求解方法主要借助值函数对策略进行评估, $V^\pi(s)$ 表示在策略 π 的基础上状态 s 的预期奖赏, 称为状态值函数。如果状态 s 可选动作有多个, $Q^\pi(s,a)$ 表示在策略 π 基础上对状态 s 采取动作 a 的预期奖赏, 称为状态动作值函数。二者之间的关系为 $V^*(s)=\max_a Q^*(s,a)$, 可根据贝尔曼最优方程对最优状态值函数 $V^*(s)$ 进行求解。

$$V^*(s)=\max_a E_{s'\sim p(s'|s,a)}[r(s'|s,a)+\gamma V^*(s')] \tag{3}$$

$$Q^*(s,a)=E_{s'\sim S}[r+\gamma\max_{a'}Q(s',a')\,|\,s,a] \tag{4}$$

在环境模型确定已知的情况下可通过动态规划算法进行求解, 但在实际应用中很难满足, Q-Leaning 算法是一种时序差分学习算法, 通过 Q 表格中的 Q 值直接估计最优状态动作值函数 $Q^*(s,a)$, Q 值通过预期估值与实际估值的差进行更新, 计算过程为

$$Q(s,a)\leftarrow Q(s,a)+\alpha(r(s'|s,a)+\gamma\max_{a'}Q(s',a')-Q(s,a)) \tag{5}$$

经过多轮迭代训练后得到 Q 表, 在测试阶段 Agent 通过查询 Q 表中当前状态最大 Q 值对应的动作, 依次进行规划。随着问题规模的增加, 存储和维护 Q 表的代价太大, 因此考虑使用线性函数来近似表示状态的 Q 值, 但是强化学习没有数据集的概念, 直接把决策过程当做数据集的话会导致数据之间相关性太强, 算法不收敛。Mnih 等人[19]提出了深度 Q 网络(DQN)算法, 通过在训练过程使用经验回放机制降低了样本之间的关联性, 然后通过当前值网络和目标值网络之间的差值对神经网络进行学习, 在解决一些即时战略游戏等复杂问题上取得了显著进展。其中, 在非目标状态下目标 Q 值 y_j 的计算见公式(6), 然后通常采用目标 Q 值与当前网络的估计 Q 值的平方差作为损失函数进行训练。

$$y_j=r_j+\gamma\max_a Q_{target}(s',a\,|\,\theta_{target}) \tag{6}$$

DQN 算法容易出现过估计问题, 即估计 Q 值往往比真实的值要大, 于是出现了 Double_DQN 算法(DDQN)[20], DDQN 使用两个值函数, 一个用来选择动作, 一个用来评估当前状态的价值, 两个值函数的参数不一样。估计 Q 值的选取沿用之前的策略, 不同之处在于计算目标 Q 值的时候, 动作 a 不是直接从目标网络选择最大值, 而是在当前网络找出最大 Q 值的动作, 然后放入目标网络取计算目标 Q 值, 即公式(6)中的 a 不再是基于 θ_{target} 进行选择, 而是基于 $\theta_{current}$ 选择, 除此之外其他部分与 DQN 一致。算法流程如下所示。

算法 1 DDQN 算法

输入: 状态空间 S, 动作空间 A, 参数集

输出: 神经网络参数 w,b

1 初始化神经网络参数

2 $i\_episode=0$, $MAX\_EPISODES$ DO

3 初始化环境, 获取当前状态 s

4 输入 s，根据 Q 网络的输出 Q 值，使用 ε 贪婪法选择动作 a

5 在环境中执行 a，获得下一步状态 s'，立即奖励 r，是否完成 $done$

6 将 $(s,a,s',r,done)$ 存入经验池

7 $s = s'$

8 当经验池存储条目超过一定值，开始采样，计算目标 Q 值

9 计算损失函数，通过反向传播更新神经网络参数 w,b

10 若运行步数达到更新频率，更新目标网络参数 w',b'

11 如果 $done$ 为真或者超过每轮最大运行步数，则完成一轮迭代，否则转 4

3 基于深度强化学习的网络攻击路径规划算法

3.1 渗透测试网络模型

本文将渗透测试问题表示为强化学习模型，也就是四元组$\{S,A,R,T\}$。S 定义为网络的状态空间，A 表示攻击者在某一状态下可能采取的行动，R 表示攻击者采取特定行动后获得的奖励，T 为状态转移函数，即在主机状态为 s 时采取动作 a 后转移至下一主机状态 s' 的概率。状态 s 反映了主机的所有信息，包括主机标识、操作系统字段、服务字段和控制字段。更进一步地，主机标识由子网和 IP 地址唯一确定；操作系统字段表示主机的操作系统，本文假设一台主机只运行某一种操作系统，所以操作系统字段中的元素具有互斥性；服务字段表示主机上运行的服务，可以有多种，攻击者针对这些服务进行漏洞利用以控制主机；控制字段记录当前主机的额外信息，包括是否被攻陷、是否可访问、是否被发现、主机的资产价值、可被访问的级别(user 或 root)。对上述信息通过 one-hot 方式进行编码，主机标识字段的维度等于子网数和各子网最大主机数之和，操作系统字段的维度等于网络内所有操作系统种类数，服务字段的维度等于各主机运行服务的最大值，控制字段的维度是固定的，每一变量为布尔型或数值型。

动作空间表明了攻击者可能采取的攻击行为，包括信息收集动作和漏洞利用动作。在真实的渗透环境下，攻击者难以在一开始就获取准确全面的网络环境信息，因此引入信息收集动作增加对环境的感知能力。现实中的信息收集可以是主动方式收集，如扫描；也可以是被动方式收集，如日志分析或流量分析。本文屏蔽具体实现细节，认为在网络连通且防火墙允许的条件下，信息收集动作可以成功返回目标的相关信息，如存在或是不存在。信息收集动作分为子网信息收集和主机信息收集，其中，子网信息收集用来发现新的子网及其中的主机。主机信息收集是为了获取某一主机的配置信息，进一步可分为操作系统信息收集和服务信息收集，通过操作系统信息收集，可对状态空间中的操作系统字段进行更新；通过服务信息收集，可确定主机运行了哪些服务，进而帮助攻击者选择最佳漏洞利用动作进行渗透利用。信息收集动作的执行成本设为某一固定值。漏洞利用动作是攻击者针对主机某一存在漏洞的服务，选择合适的参数和漏洞利用程序进行攻击。诸如开源漏洞框架 metasploit(简称 MSF)等已集成了大量成熟的漏洞利用程序，因此本文对漏洞利用动作进行抽象表示，忽略了选择参数和漏洞利用程序的过程，认为针对特定服务执行相应的动作后可以以一定概率攻下主机。进一步地，将漏洞利用动作分为远程攻击利用和本地提权利用，漏洞利用动作具有攻击成本 c 和攻击成功率 p 两个属性，攻击成本 c 由服务漏洞自身的高危程度 s、漏洞利用过程的耗时 t 共同决定；攻击成功率 p 用以刻画动作效果的不确定性。其中，服务漏洞自身的高危程度借助通用漏洞评分系统(Common Vulnerability Scoring System, CVSS)进行评价，CVSS 得分反映了漏洞的严重性；漏洞利用过程的耗时可结合 MSF 框架中渗透攻击模块命令及参数复杂程度进行估计；攻击成功率可以根据 core impact 等测试平台通过大量模拟实验获得。

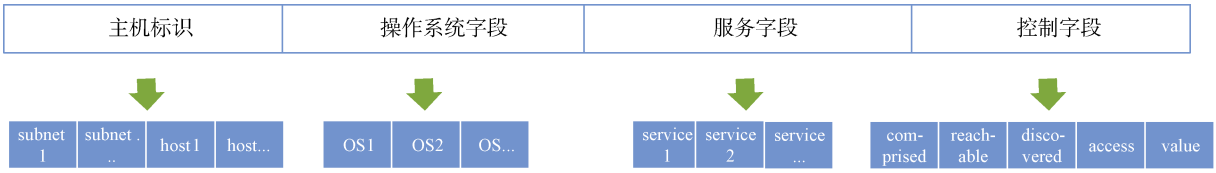


图 1 状态空间向量化表示

Figure 1 Vectorized representation of state space

以漏洞高危程度的最大值最小值为界, 把攻击耗时 t 进行标准化保证量纲统一, 针对服务漏洞 i 的攻击动作成本 c_i 的计算见公式(7)

$$c_i = k(10 - s_i) + (1-k)\frac{t_i\text{-}\min(T)}{\max(T)-\min(T)}(\max(S)-\min(S)) \tag{7}$$

其中, s_i 是漏洞 i 的 CVSS 评分, t_i 是针对漏洞 i 的攻击用时, S 和 T 分别表示所有漏洞的评分及攻击用时的集合, $k \in (0,1)$ 为调节因子, 当 k 取 0.5 时, $c_i \in (0,10)$。我们的目标是寻找 c_i 尽可能小的动作, 公式表明某一漏洞越高危, 越值得被推荐使用, c_i 越小; 某一漏洞的利用程序越简单, 参数越少, c_i 越小。对动作空间同样借助 one-hot 编码进行向量化表示, 假设网络中存在 M 个子网、一共有 N 台主机, 主机最多运行 K 个服务, 所有信息收集动作的个数为 $M+N+N*K$, 所有漏洞利用动作的个数为 $2*N*K$。

奖励函数的设计参考文献[15], 假设 Agent 在状态 s 下采取动作 a 后转移至状态 s', 如果 s' 对应的主机不是目标状态, 则 *value* 值为 0, 动作所获得的立即奖励为负; 如果 s' 对应的状态是目标状态, 则立即奖励等于被入侵主机的资产价值与动作执行代价之差。目标状态的 *value* 设置为某一较大的正数, 其余主机的 *value* 设为 0。

$$R(s,a) = value(s') - c_a \tag{8}$$

3.2 算法流程

DDQN 算法初始阶段采用盲目搜索的方式进行搜索, 导致目标状态的奖励传播太慢。探索和利用问题是强化学习过程需要考虑的重要因素, 解决探索利用平衡问题常采用的方法是 ε 贪婪策略。在每次选择动作时, 首先产生一个随机数 $\varsigma \in (0,1)$, 与 ε 进行比较后决定选择动作的方式, ε 的作用是调整探索和利用的比例。如果 ε 的值过小, Agent 更侧重利用已有的经验, 容易陷入局部最优; 反之 Agent 更侧重探索未知区域, 收敛速度过慢。

$$\pi(s) = \begin{cases} random(A), \varsigma \leqslant \varepsilon \\ a \sim \arg\max Q(s,a), \varsigma > \varepsilon \end{cases} \tag{9}$$

3.2.1 路径启发信息设计

经典的强化学习算法并不能高效利用搜索过程中的知识, 擅长学习却不擅长搜索, 而启发式搜索擅长搜索却不擅长学习, 因此本文通过在强化学习过程中加入路径启发信息, 指导后续的学习过程, 从而提高深度强化学习算法的效率。开始训练前设置路径启发表(Path Heuristic List, PHL), 每执行一个情节(episode)后, Agent 获得了从起始状态到目标状态的一条动作执行轨迹, 然后计算每一状态至目标状态的路径长度, 即从当前状态至目标状态的状态数。在获取某一状态的路径长度后, 直接对 PHL 表进行更新, 使 PHL 表中当前状态的启发值保持为最小值。PHL 中每一状态的值对应为(h_a, $H(s,a)$), 其中, $H(s,a)$ 表示在状态 s 采取动作 a 后猜测其到达目标状态的最小距离, 计算过程为

$$H(s,a) = \| s' - goal \| \tag{10}$$

s' 是在状态 s 采取动作 a 后转移到的状态。在本文中, $H(s,a)$ 表示为 s' 到目标 *goal* 的路径状态数。当给定了状态的启发值后, 启发式动作 h_a 定义为

$$h_a(s,H) = \arg\min_a H(s,a) \tag{11}$$

接下来是路径启发表的应用, 对 ε 贪婪策略进行扩展, 采用动态 ε 选择。将动作选择方式分为三种, 借助 PHL 表选择动作的方式记做 $\varepsilon_{1\text{-}1}$, 随机动作选择方式记做 $\varepsilon_{1\text{-}2}$, 通过深度神经网络产生的最大 Q 值选择方式记做 ε_2。当已运行步数 *steps* 小于最大探索阶段步数 *EXPLORE_STEPS* 时, $\varepsilon_1 = 1 - steps * \frac{1 - FINAL\_\varepsilon}{EXPLORE\_STEPS}$, $\varepsilon_2 = 1 - \varepsilon_1$, 此时把 ε_1 进一步细分为 $\varepsilon_{1\text{-}1}$ 和 $\varepsilon_{1\text{-}2}$, 若 *steps* < *EXPLORE_STEPS*/4, $\varepsilon_{1\text{-}1} = 0$, $\varepsilon_{1\text{-}2} = \varepsilon_1$; 否则 $\varepsilon_{1\text{-}1} = \varepsilon_1$, $\varepsilon_{1\text{-}2} = 0$。当已运行步数 *steps* 超过最大探索阶段步数 *EXPLORE_STEPS* 时, 令 $\varepsilon_{1\text{-}1} = 0$, $\varepsilon_{1\text{-}2} = FINAL\_\varepsilon$, $\varepsilon_2 = 1 - FINAL\_\varepsilon$。这样设计的目的是, 在训练步数达到一定值之前, 主要借助 ε_1 进行动作选择, 而且会随着训练进行逐渐递减。因为刚开始 PHL 表尚未完全构建, 所以在 PHL 表完善之前, 发挥主要作用的是 $\varepsilon_{1\text{-}2}$; 当 PHL 表构建完成后, 发挥主要作用的是 $\varepsilon_{1\text{-}1}$, ε_2 也在不断增加; 当超过最大探索阶段步数后, 基本只依靠 ε_2 进行选择, 保留极小概率 *FINAL_ε* 的随机选择方式。

$$\pi(s) = \begin{cases} h_a(s,H), & \varsigma \leqslant \varepsilon_{1\text{-}1} \\ random(A), \varepsilon_{1\text{-}1} < \varsigma \leqslant \varepsilon_{1\text{-}1} + \varepsilon_{1\text{-}2} \\ a \sim \arg\max_a Q(s,a), \varsigma > \varepsilon_{1\text{-}1} + \varepsilon_{1\text{-}2} \end{cases} \tag{12}$$

3.2.2 基于深度优先渗透的动作选择策略设计

在没有任何经验指引情况下, 强化学习 Agent 将信息收集动作和利用动作一视同仁, 反复尝试所有的状态空间和动作空间, 在网络包含大量子网大量主机的情况下, 状态空间的过快增长限制了算法的可扩展性。通过观察每个情节中生成的动作执行轨

迹可以发现, Agent 尝试了大量渗透效果相似的状态, 比如对非目标主机所在的子网进行子网内主机的信息收集和漏洞利用, 然而从实际渗透测试角度考虑, 攻击者在抵达目标主机之前, 最主要的目的是进行不同子网的跨网段攻击, 以更快抵达目标主机。

本文将除目标主机之外的子网内状态空间进行逻辑意义上的状态聚合, 认为控制子网内少数主机与控制大量主机对于实现攻击目标主机而言效果一样, 提出了基于深度优先渗透的动作选择策略。首先, 在攻击者进入目标主机所在子网之前, 当成功地对一个子网内某台主机进行本地提权后, 下一步优先使用信息收集动作以探测发现新的子网和主机。如果信息收集动作失败, 则采用完整的动作空间进行信息收集和漏洞利用尝试, 成功对另一台主机进行本地提权动作后继续优先渗透下一子网, 直至成功控制目标主机。

以一个典型的渗透测试网络环境为例进行说明, 攻击者位于外网, 目标主机是位于业务网的 FTP 服务器。非军事冲突区(Demilitarized Zone, DMZ)的 Web 服务器对外网提供访问服务, 攻击者在初始时刻只能访问 DMZ 区的 Web 服务器或 Redis 服务器。办公室内网和业务网管理区可以访问 DMZ 区, 办公室内网可访问业务网管理区。办公室内网和业务网管理区只能通过 Web 服务器某一特定漏洞去访问, 比如服务端请求伪造漏洞(Server-Side Request Forgery, SSRF)。业务网只能通过业务网管理区访问, 不能与任何其他网络互相访问。软防火墙不允许 Redis 服务器访问其他子网。每台主机均运行了一些具有脆弱性的服务。理想状态下最佳的渗透路径为: 攻击机→DMZ 区 Web 服务器→信息收集发现存在业务网管理区→业务网管理区→VPN 服务器→信息收集发现业务网→业务网 FTP。但是在训练过程中 Agent 可能会在入侵 DMZ 区 Web 服务器后, 对后台的文件服务器等进行信息收集和利用, 或者在 Web 服务器日志中发现存在办公室内网, 通过反向代理方式入侵办公室内网。这对攻击目标主机而言没有实际价值, 反而增加了成本。采用本文的深度优先渗透策略, 如果攻击者刚开始入侵了 Web 服务器并成功执行了本地提权动作, 则通过信息收集动作可以发现新的子网, 搭建反向代理后直接对下一个子网进行渗透; 如果攻击者刚开始入侵的是 Redis 服务器, 执行信息收集动作会返回失败, 此时采用完整的动作空间, 对 Web 服务器进行漏洞利用。可以看出基于深度优先渗透的攻击策略可以减少每个情节中动作执行的个数, 更快到达目标主机, 提高强化学习的速度和准确率。

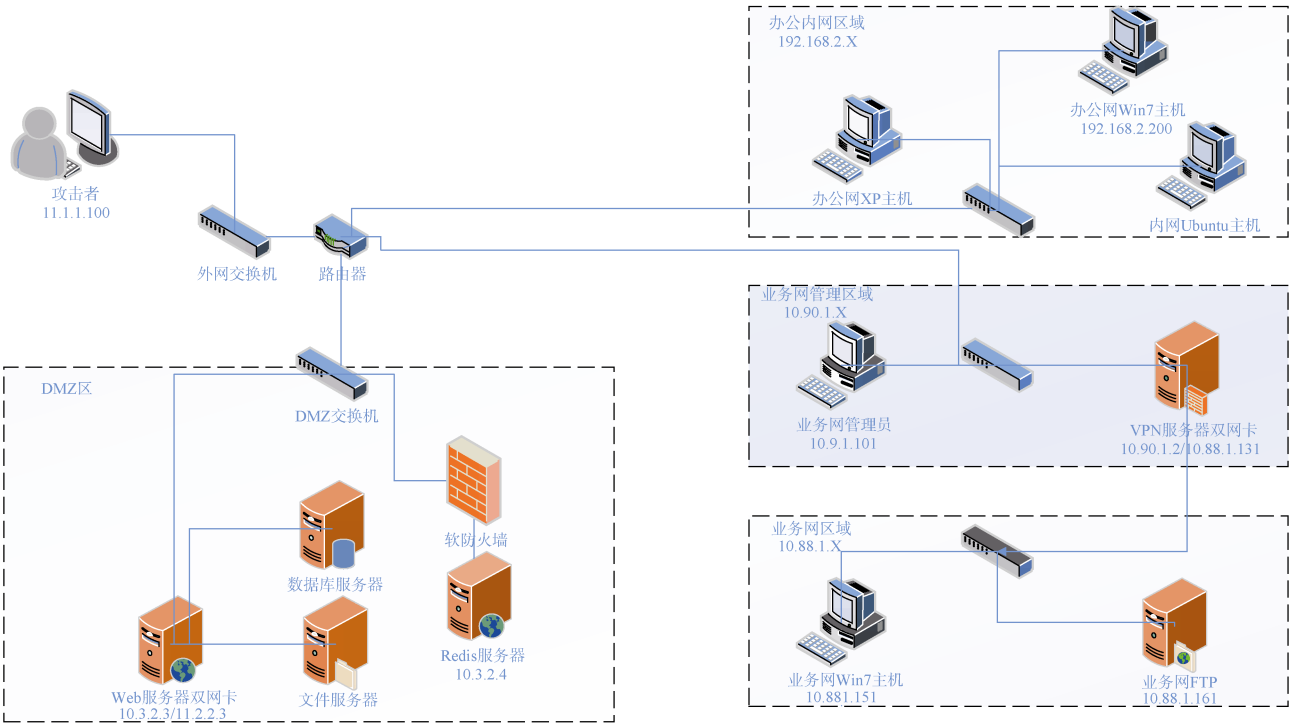


图 2　典型的渗透测试网络环境

Figure 2　Typical penetration testing network environment

3.2.3 DH_DDQN 算法流程

算法 2 DH_DDQN 算法流程

```
1  FUNCTION update_phl(trace_sa)
2    FOR i = 0, len(trace_sa) – 1 DO
3      (s,a), h = trace_sa[i], len(trace_sa) – i – 1
4      IF s not in PHL.keys() THEN
5        append s to PHL, set h,a = h,a
6      ELIF h < PHL[s].h THEN
7        update PHL[s].h, PHL[s].a = h,a
8      END IF
9    END FOR
10 FUNCTION get_taboo_a(a)
11   initial taboo_a
12   add actions within the same subnet to taboo_a
13   add remote exploit actions and privilege elevation actions within network to taboo_a
14   RETURN taboo_a
15 FUNCTION main()
16   initial neural networks, PHL, steps_done = 0
17   FOR episode = 1, MAX_EPISODE DO
18     s = env.reset(), initial trace_sa, taboo_a, step_num = 0
19     WHILE not done AND step_num < MAX_EP_STEPS DO
20       a = get_egreedy_action(s, ε = get_epsilon())
21       IF a in taboo_a THEN
22         select a which not in taboo_a and update a
23       END IF
24       save(s,a) to trace_sa
25       s_next, r, done, info = env.step(a)
26       IF a is privilege action AND success THEN
27         taboo_a = get_taboo_a(a)
28       ELSE
29         reset taboo_a
30       END IF
31       save (s,a,s_next,r,done) to MEMORY
32       IF steps_done > START_STEPS THEN
33         sample and update θ, θ' at a certain frequency
34       END IF
35       steps_done += 1, step_num += 1, s = s_next
36   END WHILE
37   IF episode < MAX_PHL_STEPS AND done THEN
38     update_phl(trace_sa)
39     END IF
40   END FOR
```

对上述算法进行简要说明。首先对神经网络和路径启发表 PHL 进行初始化，行 19~36 表示训练一个情节后，如果成功到达目标状态，则对 PHL 表进行更新，更新次数设置为 *MAX_PHL_STEPS*。如果当前状态尚未出现在 PHL 表中，则新增元素，设置对应的动作和启发值；否则更新启发值信息使其保持为最小值，具体步骤见行 2~9。行 26~29 表明如果当前动作成功对目标主机进行提权利用后，产生禁用动作表 taboo_a，将本子网内的动作以及整个网络的漏洞利用动作添加至 taboo_a 表。如果下一步信息收集失败的话，把禁用动作表置空，继续采用原来的动作空间进行选择。

4 实验设计与结果分析

本文使用文献[15]中的网络模拟器设计渗透测试环境，通过 yaml 文件进行描述。对算法运行过程中总奖励和运行时间的变化进行展示，验证算法性能。

4.1 实验及参数设计

实验所使用的的网络拓扑结构如图 3 所示，攻击者在初始时刻仅能访问子网 1，关于各子网间是否具有网络连通关系可以通过配置防火墙规则实现。例如，在图 3 中，设置子网 1 可以访问子网 3 不能访问子网 2，子网 2 仅能访问子网 3 不能访问子网 1，子网 3 可以访问子网 1 也可访问子网 2。攻击者在初始时刻并不知道子网 2 和子网 3 的存在，状态空间中对应的状态值为 0，当在子网 1 通过信息收集动作发现子网 3 后，更改相应的状态值为 1，在此基础上可以对子网 3 的主机进行漏洞利用，修改对应的状态值，以此类推。

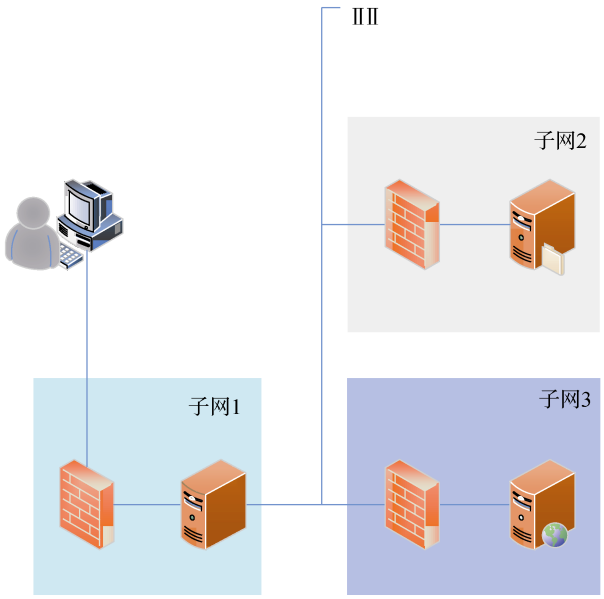


图 3 网络结构示意

Figure 3 Schematic diagram of network structure

首先设计三种不同规模的网络环境, 测试本文算法与其他算法的性能。每个网络环境的子网数、子网每主机数、以及主机漏洞数逐渐增加, 同时网络结构的复杂程度也增加, 攻击者需要更多的攻击步骤才能到达目标状态。目标主机从最后两个子网中挑选 2 台, 资产价值均设为 10。实验场景信息见表 1。

表 1　实验场景信息

Table 1　Experimental scene information

	子网数	各子网主机数	主机漏洞总数
场景 1	3	[1,2,2]	2
场景 2	4	[1,2,5,2]	5
场景 3	5	[2,3,5,6,4]	8

实验硬件配置包括 Intel i9-7980XE CPU、128G 内存, 操作系统为 windows 10, 算法程序通过 python 语言编写。程序超参数的含义见表 2, 部分超参数需要根据问题规模的扩大进行适当调整, 这类超参数的设定对实验结果有较大影响, 表 3 给出了 3 种实验场景下的超参数取值。其余超参数通过参阅相关文献和进行大量实验进行设定, 对实验结果的影响不大, 保持在适当范围即可。

4.2　实验结果展示与分析

Dueling_DQN[21]是采用了竞争网络结构的 DQN 算法, 将 Q 网络分解为价值函数网络和优势函数网络, 在 gym(OpenAI 的开源测试平台)的一些简单测试任务中取得了比 DQN 算法更优的效果。引入竞争网络结构的 DDQN 算法称为 Dueling_DDQN 算法。将加入路径启发信息的 DDQN 算法记做 H_DDQN, 首先将本文算法 DH_DDQN 与 DDQN、Dueling_DDQN 和 H_DDQN 算法进行对比, 在保持超参数及环境设定一致的情况下进行实验, 运行结果见图 4、图 6 和图 8, 分别表示 3 种实验场景下 4 种算法在每个情节所获得的总奖励, 四种算法最终都收敛于最佳环境奖励值。图 5、图 7 和图 9 分别展示了 3 种场景下算法运行过程中的时间变化。

表 2　算法超参数的设定及含义

Table 2　Setting and meaning of algorithm hyperparameters

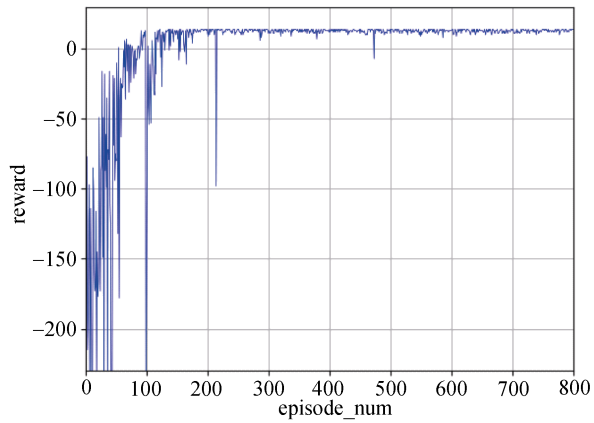
参数名	值	含义
MAX_STEPS	100000	最大运行步数
γ	0.9	折扣因子
LR	0.001	神经网络学习率
MAX_PHL_STEPS	100	*PHL* 表的更新次数
HIDDEN_SIZE	[128,128]	隐藏层神经元层数及个数
BATCH_SIZE	32	抽样规格
REPLAY_SIZE	10000	记忆存储规格
EXPLORATION_STEPS	10000	探索阶段步数
TARGET_UPDATE_STEPS	1000	目标网络更新频率
START_STEPS	2000	神经网络开始学习的步数
MAX_EP_STEPS	300	每情节的最大运行步数

表 3　不同实验场景下部分超参数的设定

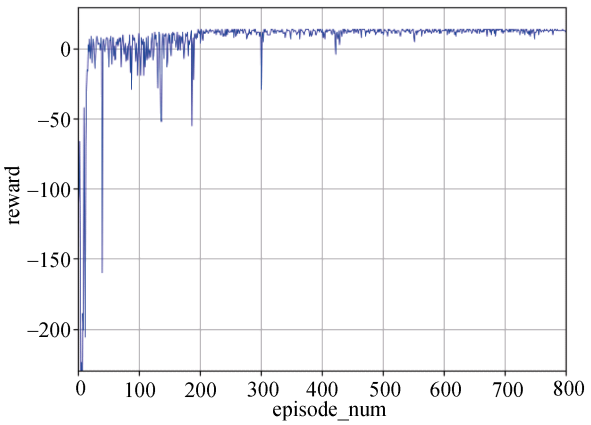
Table 3　Settings of some hyperparameters in different experimental scenarios

	MAX_STEPS	*MAX_EP_STEPS*	*EXPLORATION_STEPS*	*START_STEPS*	*REPLAY_SIZE*
场景 1	100000	300	10000	2000	10000
场景 2	100000	1000	50000	5000	10000
场景 3	150000	3000	80000	5000	20000

从场景 1 的实验结果看出, 在实验规模较小的情形下, 四种算法的初次收敛次数相近, 而 H_DDQN 算法的学习过程更为迅速。从时间曲线看, DDQN 和 Dueling_DDQN 算法的运行时间接近, H_DDQN 和 DH_DDQN 算法的运行时间接近, 均大



DDQN算法实验结果



H_DDQN算法实验结果

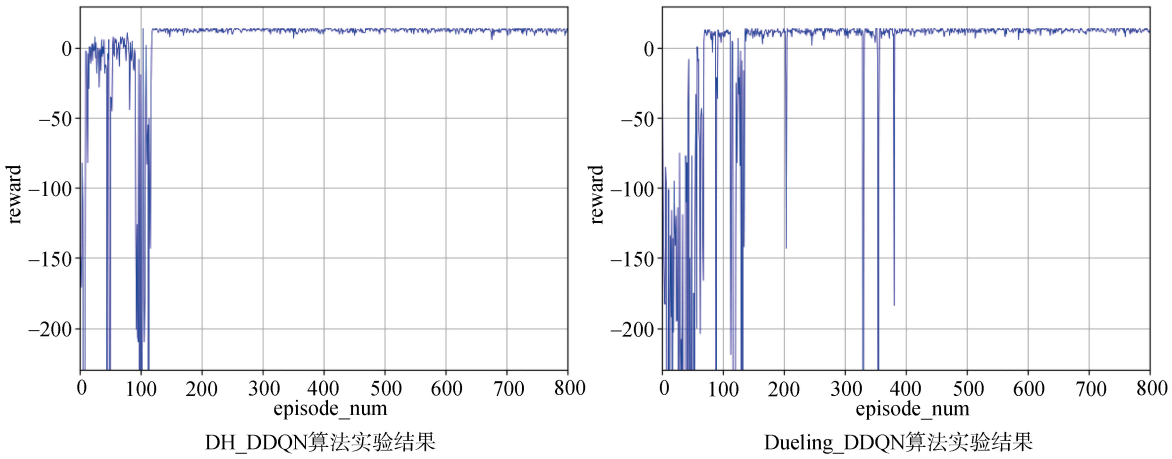


图 4　场景 1 实验结果

Figure 4　Experimental results in scenario 1

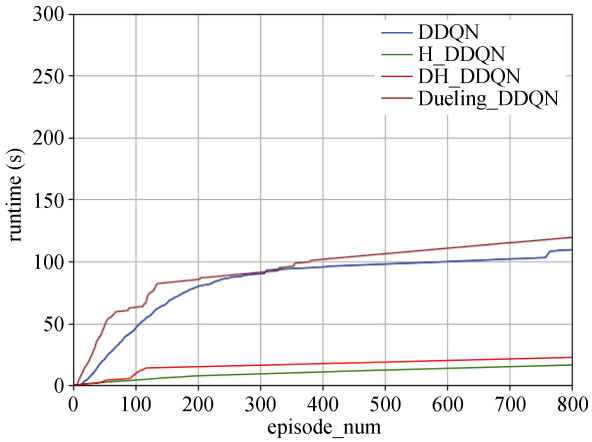


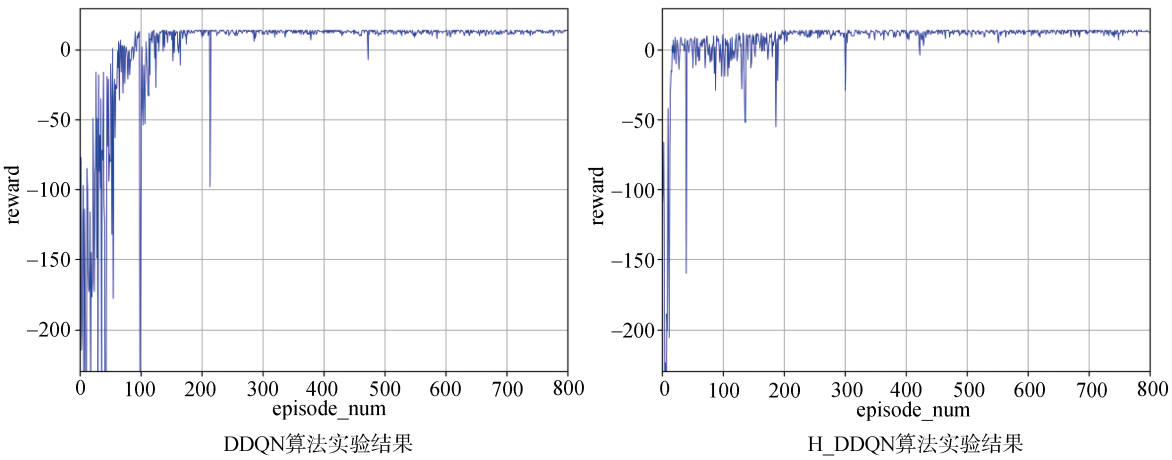
图 5　四种算法训练过程的运行时间

Figure 5　Running time of the four algorithm training processes

幅优于前两种算法, 说明本文算法可以提高 Agent 的学习速度。由于网络结构相对简单, DH_DDQN 算法的深度优先渗透策略并未发挥太大作用, 反而增加了一些计算量, 所以在运行时间上比 H_DDQN 略多。

从场景 2 的实验结果可以看出, 网络规模扩展后, DDQN 和 Dueling_DDQN 算法的学习过程更加复杂漫长, H_DDQN 和 DH_DDQN 算法能够在更少的训练次数中达到收敛。结合运行时间曲线看, Dueling_DDQN 算法在状态空间较大的测试环境中, 运行时间增长过快, 收敛慢; DH_DDQN 算法的运行时间最快, 达到收敛所需要的的时间在 20s 左右。

从场景 3 的运行结果可以看出, H_DDQN 和 DH_DDQN 算法相比于其余两种算法学习过程更高效, 而且随着问题规模的扩展, DH_DDQN 算法相较于其他算法的提升更明显。从 H_DDQN 算法运行情况可以看到两次明显的波动, 第一次波动发生在刚加入路径启发信息时, 随着训练的进行, 借助路径启发信息进行动作选择的概率越来越低, 最后趋于 0, 此时产生第二次明显的波动, 路径启发信息作用的时间段在这两次波动之间。在训练过程中逐渐降低直至撤销路径启发选择动作的概率的原因在于, 虽然加入启发式动作选择策略后算法可以很快收敛,



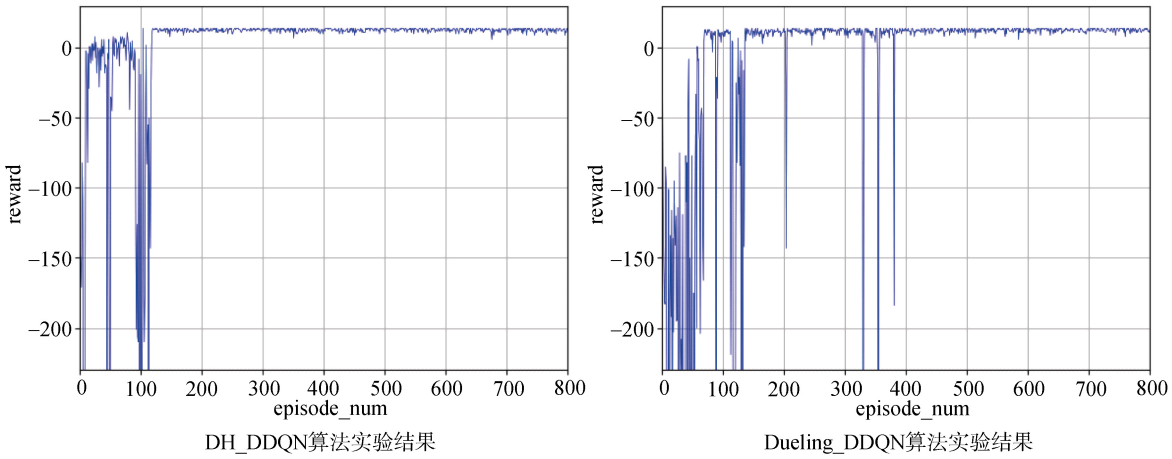


图 6　场景 2 实验结果

Figure 6　Experimental results in scenario 2

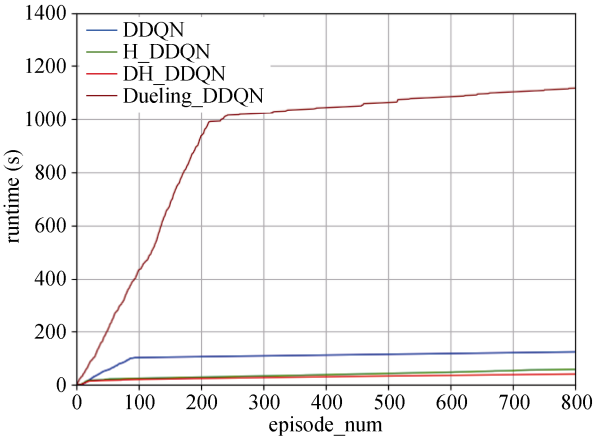
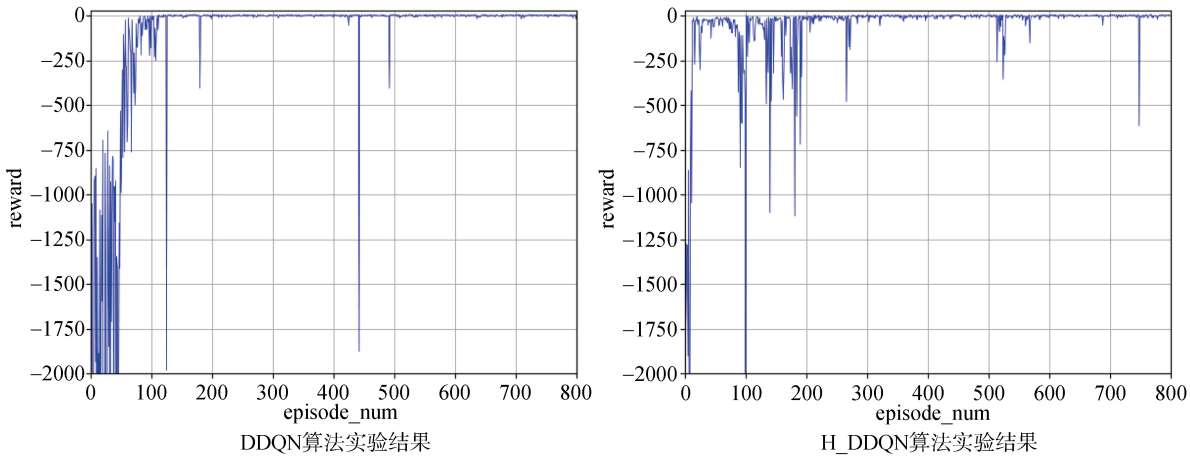


图 7　四种算法训练过程的运行时间

Figure 7　Running time of the four algorithm training processes

但是无法确定算法收敛是构造路径启发表带来的结果还是神经网络参数已训练完毕带来的结果。从实验结果看, 训练次数在 20~70 之间的“收敛”是路径启发信息本身造成的, 此时神经网络参数没有达到最优, 这才导致撤销路径启发信息这种动作选择方式后, 网络再一次出现较大波动, 这是神经网络自学习的过程, 迭代 200 次以后的收敛才是神经网络训练完成的结果。DH_DDQN 算法在 H_DDQN 算法基础上加入深度优先渗透策略, 中间的波动是 3 种动作选择方式切换造成的, 神经网络经过训练后收敛更快, 而且收敛后的策略更加稳定。

接下来设计更复杂的实验场景以验证 DH_DDQN 算法的可扩展性。具体场景信息见表 4, 实验结果见图 10, 算法近似收敛时的迭代次数及运行时间见图 11。场景 4、5、6 在保持网络拓扑一致的情况下增加主机漏洞总数, 使用本文算法进行路径规划。场景 4、7、8 在保持主机漏洞总数的情况下增加子网个数及主机个数, 使用本文算法进行路径规划。关于各实验场景所使用的算法超参数见表 5。可以看出, 在网络及主机数量不变的情况下, 随着主机漏洞数目的增加, 算法达到近似收敛时需要更多的迭代次数, 时间增长相对平稳; 增加子网个数及



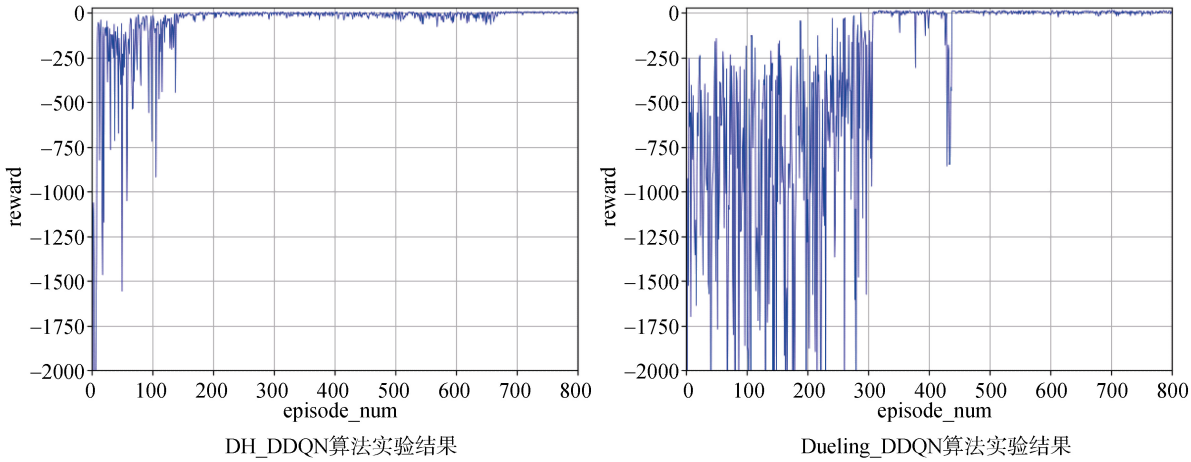


图 8 场景 3 实验结果

Figure 8 Experimental results in scenario 3

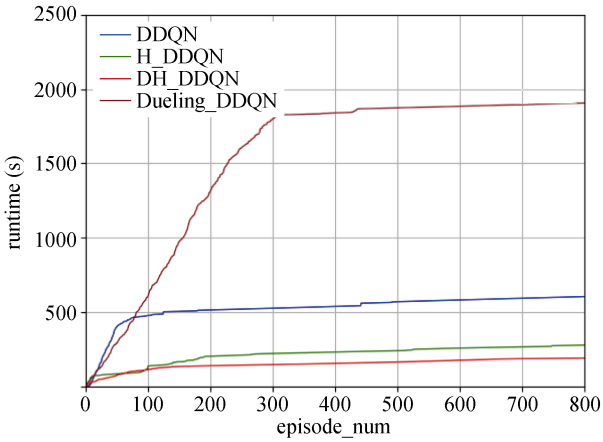


图 9 四种算法训练过程的运行时间

Figure 9 Running time of the four algorithm training processes

子网内主机个数时，由于 Agent 在每个情节需要尝试动作数迅速增长，所以算法收敛时所需要的时间也迅速增长。在 100 台主机以内的网络环境中，本文算法能够在有限时间范围求得规划解。

5 结束语

本文提出的 DH_DDQN 算法融合了路径启发信息引导和深度优先渗透的动作选择策略，可以有效提升算法训练效率，通过实验证明在同等实验条件和参数设置条件下可以解决更复杂的攻击路径规划问题。DH_DDQN 算法的改进措施相较于原算法在计算复杂度和存储复杂度方面有一定的影响。首先在路径启发信息的构建过程中，增加的计算量等于每个情节的路径长度与最大更新次数 *MAX_PHL_STEPS* 之积，在每个情节中路径长度的最大值为 *MAX_EP_STEPS*，所以在最差情况下，增加计算次数为 *MAX_PHL_STEPS* × *MAX_EP_STEPS*，增加的存储量为 *MAX_PHL_STEPS* × *MAX_EP_STEPS*。实际上，*MAX_PHL_STEPS* 的值不必设置过大，更多操作是更新元素而不是新增元素，在奖励稀疏的情况下增加的存储量远小于理论值。其次是深度优先渗透策略的设计，深度优先渗透策略发挥作用的时间

表 4 实验场景信息

Table 4 Experimental scene information

	子网数	各子网主机数	主机漏洞总数
场景 4	5	[2,3,5,6,4]	8
场景 5	5	[2,3,5,6,4]	16
场景 6	5	[2,3,5,6,4]	24
场景 7	8	[5,5,5,10,5,10,5,5]	8
场景 8	10	[5,10,20,5,10,10,5,20,10,5]	8

表 5 不同实验场景下部分超参数的设定

Table 5 Settings of some hyperparameters in different experimental scenarios

	MAX_STEPS	*MAX_EP_STEPS*	*EXPLORATION_STEPS*	*START_STEPS*	*REPLAY_SIZE*
场景 4	150000	3000	80000	5000	20000
场景 5	150000	3500	80000	6000	20000
场景 6	150000	4000	80000	8000	30000
场景 7	200000	6000	100000	10000	30000
场景 8	400000	10000	200000	15000	50000

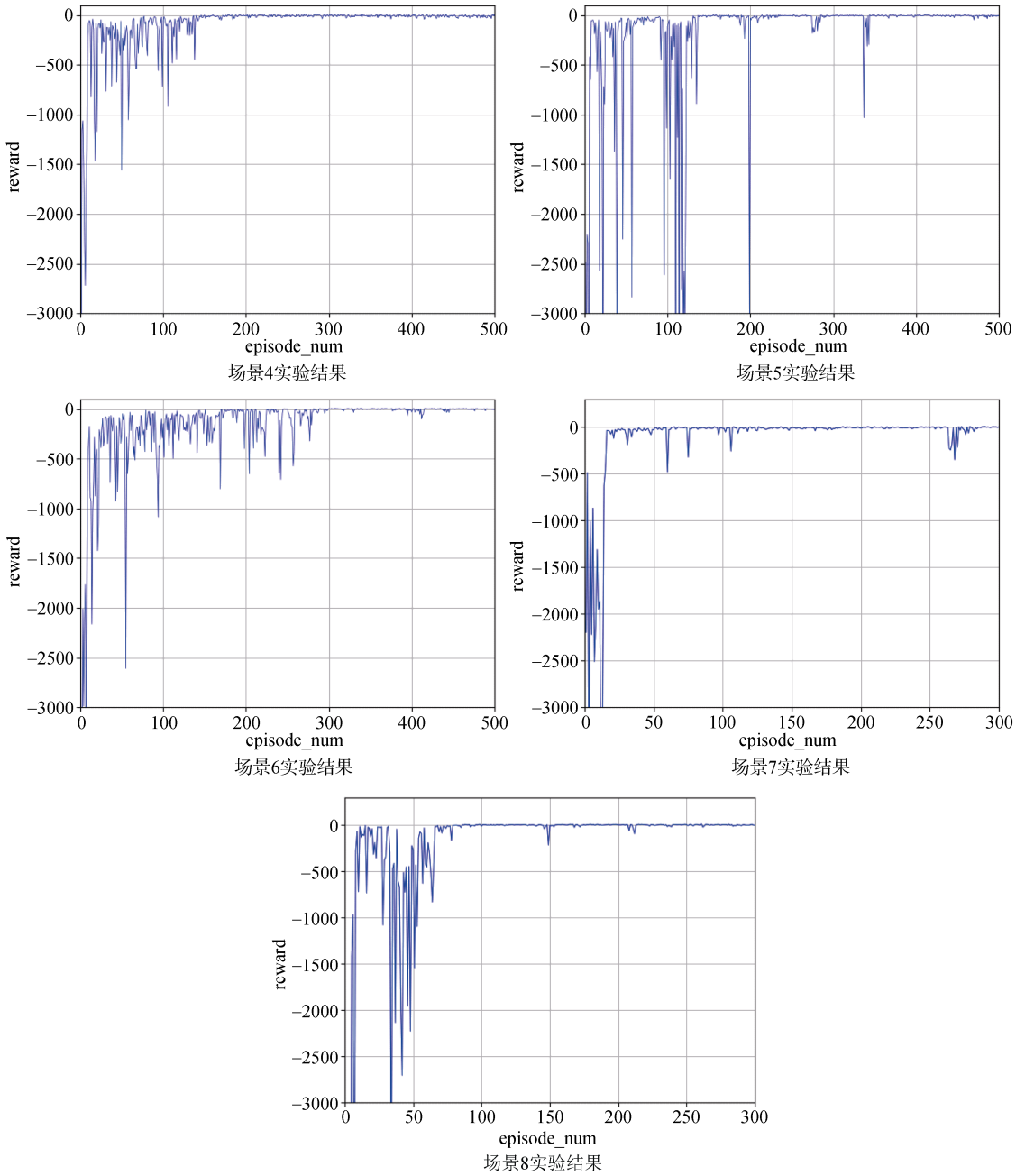


图 10　本文算法在不同规模实验场景下的运行结果

Figure 10　The results of the algorithm in this paper in different scale experimental scenarios

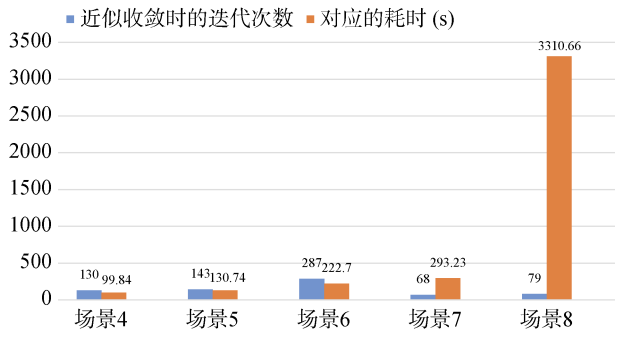


图 11　本文算法在不同场景下的迭代次数及运行时间

Figure 11　The number of iterations and running time of the algorithm in this paper in different scenarios

段在算法收敛前，当神经网络训练完成后，几乎不会再进入算法 2 第 21、22 步。假设达到收敛时的迭代次数为 N，算法 2 第 22 步尝试更换动作的次数的最大值为 M，则这一过程增加的计算量为 O($N \times M \times MAX\_EP\_STEPS$)，此外第 27 步构建 taboo_a 表的过程，相当于对动作空间的一次遍历，这一过程增加的计算量为 O($MAX\_STEPS \times \|A\|$)，综合来看基于深度优先渗透策略增加的计算量为 O($MAX\_STEPS \times \max\{M, \|A\|\}$)，增加的存储空间的最大值是 taboo_a 表的长度，即 O($\|A\|$)。

本文在 DDQN 算法基础上的改进方法简洁有

效。路径启发信息值作用于训练前期的一段时间, 而后概率降为 0, 目的是为了提高训练初期的探索路径质量, 同时基于深度优先渗透策略在收敛后也几乎不会发挥作用。训练结束后深度神经网络习得最佳策略, 可以用于攻击者在任一状态下的攻击规划。与其它算法对比, 本文算法收敛速度更快, 运行时间更短。未来可以考虑以下方面的改进。一是 DH_DDQN 算法未对神经网络结构进行改进调整, 如果能够与更先进的神经网络或优化方式融合, 算法性能会有更大提升; 二是深度强化学习的训练过程较为漫长, 如何保存和转化历史策略去适应全新的环境, 也是一个比较复杂的问题。

参考文献

[1] Stefinko Y , Piskozub A , Banakh R . Manual and automated penetration testing. Benefits and drawbacks. Modern tendency[C]. *2016 13th International Conference on Modern Problems of Radio Engineering*. Telecommunications and Computer Science (TCSET). IEEE, 2016.

[2] M. R. Reddy and P. Yalla, "Mathematical analysis of Penetration Testing and vulnerability countermeasures," *2016 IEEE International Conference on Engineering and Technology (ICETECH)*, 2016, pp. 26-30, doi: 10.1109/ICETECH.2016.7569185.

[3] Zang Yichao, Zhou Tianyang, Zhu Junhu, et al. Research progress on domain-independent intelligent planning technology and its attack path discovery for automated penetration testing[J]. *Journal of Electronics and Information Technology*, 2020, 42(9): 2095-2107.
(臧艺超, 周天阳, 朱俊虎,等. 领域独立智能规划技术及其面向自动化渗透测试的攻击路径发现研究进展[J]. *电子与信息学报*, 2020, 42(9):2095-2107.)

[4] Blum A L, Furst M L. Fast planning through planning graph analysis[J]. *Artificial intelligence*, 1997, 90(1-2): 281-300.

[5] Barrett A, Weld D S. Partial-order planning: evaluating possible efficiency gains[J]. *Artificial Intelligence*, 1994, 67(1): 71-112.

[6] Nau D S, Au T C, Ilghami O, et al. SHOP2: An HTN planning system[J]. *Journal of artificial intelligence research*, 2003, 20: 379-404.

[7] Van Laarhoven P J M, Aarts E H L. Simulated annealing[M]//Simulated annealing: Theory and applications. Springer, Dordrecht, 1987: 7-15.

[8] Holland J H. Genetic algorithms[J]. *Scientific american*, 1992, 267(1): 66-73.

[9] Kennedy J, Eberhart R C. A discrete binary version of the particle swarm algorithm[C]. *1997 IEEE International conference on systems, man, and cybernetics*. Computational cybernetics and simulation. IEEE, 1997, 5: 4104-4108.

[10] Dorigo M, Birattari M, Stutzle T. Ant colony optimization[J]. *IEEE computational intelligence magazine*, 2006, 1(4): 28-39.

[11] Sarraute C, Richarte G, Lucángeli Obes J. An algorithm to find optimal attack paths in nondeterministic scenarios[C]. *Proceedings of the 4th ACM workshop on Security and artificial intelligence*, 2011: 71-80.

[12] Hu T, Zhou T, Zang Y, et al. APU-D* Lite: Attack Planning under Uncertainty Based on D* Lite[J]. *CMC-COMPUTERS MATERIALS & CONTINUA*, 2020, 65(2): 1795-1807.

[13] Sarraute C, Buffet O, Hoffmann J. POMDPs make better hackers: Accounting for uncertainty in penetration testing[C]. *Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012.

[14] Kaelbling L P, Littman M L, Moore A W. Reinforcement learning: A survey[J]. *Journal of artificial intelligence research*, 1996, 4: 237-285.

[15] Schwartz J, Kurniawati H. Autonomous penetration testing using reinforcement learning[J]. arXiv preprint arXiv:1905.05965, 2019.

[16] Zhou Shicheng, Liu Jingju, Zhong Xiaofeng, et al. Intelligent penetration testing path discovery based on deep reinforcement learning [J]. *Computer Science*, 2021, 48(7): 40-46.
(周仕承,刘京菊,钟晓峰, 等.基于深度强化学习的智能化渗透测试路径发现[J].*计算机科学*,2021,48(7): 40-46.)

[17] Nguyen H V, Teerakanok S, Inomata A, et al. The Proposal of Double Agent Architecture using Actor-critic Algorithm for Penetration Testing[C]. *ICISSP*, 2021: 440-449.

[18] Zennaro F M, Erdodi L. Modeling penetration testing with reinforcement learning using capture-the-flag challenges and tabular Q-learning[J]. arXiv preprint arXiv:2005.12632, 2020.

[19] Mnih V, Badia A P, Mirza M, et al. Asynchronous methods for deep reinforcement learning[C]. *International conference on machine learning*. PMLR, 2016: 1928-1937.

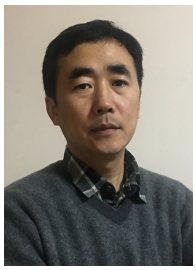
[20] Van Hasselt H, Guez A, Silver D. Deep reinforcement learning with double q-learning[C]. *The AAAI conference on artificial intelligence*, 2016, 30(1).

[21] Wang Z, Schaul T, Hessel M, et al. Dueling network architectures for deep reinforcement learning[C]. *International conference on machine learning*. PMLR, 2016: 1995-2003.

高文龙 于 2019 年在信息工程大学网络工程专业获得学士学位。现在信息工程大学网络空间安全专业攻读硕士学位。研究领域为智能规划。Email: 2396833257@qq.com

周天阳 于 2019 年在信息工程大学网络空间安全专业获得博士学位。现任信息工程大学副教授, 硕士生导师。研究领域为网络空间安全、强化学习。Email: aipteamzhouty@aliyun.com

赵子恒 于 2019 年在北京交通大学电气工程专业获得学士学位。现在信息工程大学网络空间安全专业攻读硕士学位。研究领域为信息物理系统安全。Email: 15291024@bjtu.edu.cn



朱俊虎 于 2013 年在信息工程大学计算机软件与理论专业获得博士学位。现任信息工程大学教授, 博士生导师。研究领域为网络空间安全、网络模拟与效果评估。Email: zhujunhu74@163.com