Computers, Materials & Continua CMC, vol.65, no.2, pp.1795-1807, 2020

APU-D* Lite: Attack Planning under Uncertainty Based on D* Lite

Tairan Hu[1], Tianyang Zhou[1], Yichao Zang[1,*], Qingxian Wang[1] and Hang Li[2]

Abstract: With serious cybersecurity situations and frequent network attacks, the demands for automated pentests continue to increase, and the key issue lies in attack planning. Considering the limited viewpoint of the attacker, attack planning under uncertainty is more suitable and practical for pentesting than is the traditional planning approach, but it also poses some challenges. To address the efficiency problem in uncertainty planning, we propose the APU-D* Lite algorithm in this paper. First, the pentest framework is mapped to the planning problem with the Planning Domain Definition Language (PDDL). Next, we develop the pentest information graph to organize network information and assess relevant exploitation actions, which helps to simplify the problem scale. Then, the APU-D* Lite algorithm is introduced based on the idea of incremental heuristic searching. This method plans for both hosts and actions, which meets the requirements of pentesting. With the pentest information graph as the input, the output is an alternating host and action sequence. In experiments, we use the attack success rate to represent the uncertainty level of the environment. The result shows that APU-D* Lite displays better reliability and efficiency than classical planning algorithms at different attack success rates.

Keywords: Attack planning under uncertainty, automated pentest, APU-D* Lite algorithm, incremental heuristic search.

1 Introduction

In addition to benefiting from the convenience provided by the development of state-of-the-art network services, we must address the corresponding serious security challenges. Conventional network security defense methods include firewalls, IDS, and antivirus software. These methods work in a passive way from the perspective of the defender. As a result, they are not sufficient for meeting the high demands of cybersecurity [Bertoglio and Zorzo (2017); Felderer, Büchler, Johns et al. (2016)].

[1] State Key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou, 450001, China.

[2] School of Electrical Engineering and Telecommunications UNSW, Kensington, NSW 2033, Australia.

[*] Corresponding Author: Yichao Zang. Email: aipteamzhouty@aliyun.com.

Received: 17 April 2020; Accepted: 29 April 2020.

Pentesting is an active defense method that involves analyzing network vulnerabilities and detecting potential attacks from the perspective of the attacker, and it provides a reference for developing defense strategies. However, pentesting is mainly performed manually, so the effect largely depends on the experience and skill level of the cybersecurity experts involved. With the increasing complexity of network structures and the rapid evolution of attack vectors, the requirements for pentest experts are increasing, and tasks are becoming more difficult. Automation may provide a solution to these issues. Automated pentesting not only comprehensively considers network security and rationally but also helps to reduce costs and resources, which is significant for the development and popularization of pentesting [Chen, Chen and Zhang (2014); Xiong, Yang, Zhao et al. (2017); Xu, Tao, Yang et al. (2019)].

During pentesting, the attacker probes information from the target network, analyzes the known information, decides which hosts to attack and which strategies to use, and finally attacks the target. The sequence of compromised hosts and exploit actions can be seen as an attack path from the attacker to the target. Therefore, the critical issue in automated pentesting is finding a feasible attack path, which is also called attack planning. Usually, attack planning involves mapping pentest tasks into planning problems and then using planning algorithms to solve the problems and find suitable attack paths. Most existing studies of attack planning have been based on the assumptions of a static environment and complete information. However, these factors can influence the availability of planning results because changes may occur in the network at any given time; additionally, an attacker can rarely know every detail about the target network. Additionally, most previous studies only analyzed the theoretical feasibility of attack planning and ignored the verification of planning results and availability in realistic tasks [Ramos, Lazar, Filho et al. (2017)].

In the context of the problems mentioned above, this paper proposes an uncertainty attack planning algorithm APU-D* Lite. First, we transform the pentest framework into a planning problem with Planning Domain Definition Language (PDDL). Next, we propose the pentest information graph to arrange useful information and reduce the problem scale, which functions as the input of the subsequent algorithm. Then, the APU-D* Lite algorithm is proposed based on the concept of incremental heuristic searching, and attack action planning is implemented. Finally, the sequence of node-action alternations is output to achieve attack planning in an uncertain environment.

The structure of this paper is as follows. Section 2 mainly introduces the related research in attack planning. Section 3 is the main part of this paper and presents the PDDL description of the pentest framework; additionally, the pentest information graph is proposed, and the details of the APU-D* Lite algorithm are introduced. In Section 4, a comparative experiment is performed in a typical internal network. The results and performance of APU-D* Lite and three classical planning algorithms are compared at different attack success rates, and the results verify the feasibility and effectiveness of the APU-D* Lite algorithm.

2 Related work

AI planning is one of the most important components of artificial intelligence. It involves analyzing the relevant environmental information, making an inferential judgment

according to feasible actions and resource limitations, and finally selecting the action sequence to achieve the objective. The related research originated in the late 1960s [Bozic and Wotawa (2017); Grant (2018)].

The concept of attack planning was first proposed in 2003 [Futoransky, Notarfrancesco, Richarte et al. (2003)]. A conceptual model was established from the perspective of attackers, and it defined the concepts of attacks, assets, actions, and targets in network security. In 2005, Boddy et al. [Boddy, Gohde, Haigh et al. (2005)] used AI planning in network security for the first time; in their approach, network vulnerability analysis was regarded as a planning problem to establish a Behavior Adversary Modeling System (BAMS), which was used for Course of Action (COA) generation, and the feasibility of attack planning was verified in a simple web-based file management system network. In 2009, Sarraute [Sarraute (2009)] formally proposed the idea of applying attack planning in pentesting. In 2013, Obes et al. [Obes, Sarraute and Richarte (2013)] implemented attack planning using the Metric-FF and SGP algorithms and verified the efficiency of attack planning for medium-sized networks. However, the research above regarded the studied network as a static environment and ignored its dynamic characteristics, such as software updates and host crashes. To consider uncertainty, in 2011, Sarraute et al. [Sarraute, Richarte and Obes (2011)] proposed an algorithm for attack planning based on probability planning. In this method, the uncertainty of pentesting was considered in the attack action success rate, and two primitive functions, choose and combine, were proposed for the selection of actions and strategies. Furthermore, in 2012, Sarraute et al. [Sarraute, Buffet and Hoffmann (2012); Sarraute, Buffet and Hoffmann (2013)] modeled pentesting as a partially observable Markov decision process (POMDP), which was a more accurate method than probability planning. The model accurately described the uncertainty in attack planning and considered information collection as a part of the attack action. However, this method has limitations related to the solution efficiency, which has led to restricted application. While academic research on automatic attack planning continues to expand, commercial applications involving attack planning have also been continuously explored. Core Impact [Sarraute (2013)] have been actively involved in research related to attack planning and applied the results to improve business pentesting tools.

The existing research on attack planning can mainly be divided into two classes: deterministic planning and planning under uncertainty. Deterministic planning, also known as classical planning, mainly solves planning problems in static, deterministic and completely observable environments with relatively simple domains. The algorithm systems are relatively well developed in this class, and they include A*, HSP, FF, and other. Most early studies of attack planning fall within this class. However, from the attacker's perspective, pentesting provides incomplete information acquisition, and it is difficult to determine the action effects. However, these factors are ignored in classical planning, which may fail when implementing the planning results in actual tasks. Planning under uncertainty is more suitable for dynamic, partially known environments, such as those associated with pentesting. This approach provides a way to address and analyze uncertain action effects and incomplete information. Therefore, the planning result is relatively adaptable. One method of attack planning under uncertainty involves the use of POMDP to model attack planning problems, in which uncertainty is described based on the probability distribution of the states and the best action is given based on different

observations. However, the efficiency of POMDP becomes a constraint to its application to some extent.

In 2002, Koenig and Likhachev proposed the D* Lite algorithm. D* Lite is a reverse incremental heuristic algorithm. With reverse searching, this approach records the distance from the goal node to the current node, thus making it possible to rapidly deal with unknown information and dynamic changes. In an incremental search, the agent moves forward according to the calculated shortest path, and the starting point is constantly updated based on the current node, which greatly reduces the scope and time of replanning [Koenig and Likhachev (2005)]. If the environment changes or new information is detected, the estimates of the heuristic function are updated, and the shortest path is recalculated. However, the D* Lite algorithm has problems when directly applied for attack planning. Notably, the D* Lite algorithm often uses grid maps as inputs, but the environmental information in pentests is complicated and unlike that in traditional maps. Additionally, in D* Lite algorithm, there is only one action: move to the next position. However, in a pentest, there are many vulnerabilities to consider, and the large action space can reduce algorithm efficiency. Finally, the original D* Lite method just needs to choose the position to move to; thus, the result is a sequence of nodes. However, in attack planning, it is important to choose the most suitable exploit action and host to compromise. Therefore, the result of attack planning for pentesting should be a node-action alternating sequence.

The main contribution of this paper is the development of the APU-D* Lite algorithm, which can quickly and efficiently perform attack planning under uncertain conditions. Considering the uncertain factors in pentests, the APU-D* Lite algorithm is more practical than deterministic attack planning algorithms. Additionally, based on the idea of incremental heuristic searching in the D* Lite algorithm, the APU-D* Lite algorithm performs better than existing POMDP-based algorithms in dealing with unexpected problems in the execution of planning results in a timely manner.

3 APU-D* Lite attack planning algorithm

3.1 The PDDL description

PDDL language was proposed as a standard language in the International Planning Competition (IPC) in 1998 [Pellier and Fiorino (2018)]. This language is widely used for planning problem descriptions. PDDL consists of two files: domain.pddl and problem.pddl. The domain.pddl file mainly describes background knowledge, and the problem.pddl file mainly describes the specific task information, including objects, initial states, and goals. Existing studies have shown that there is a mapping relationship between the PDDL and the pentest framework (Fig. 1) [Obes, Sarraute and Richarte (2013)].

The core of the pentest framework consists of exploit and attack modules and an attack workspace, which correspond to the PDDL description of the actions and initial conditions and goals, respectively. The exploit and attack modules are mainly composed of various vulnerability exploits and attack scripts, which can be transformed into actions in the domain.pddl file. The attack workspace is mainly the description of the network initial conditions and goals, which can be transformed to fit the task description in problem.pddl file.

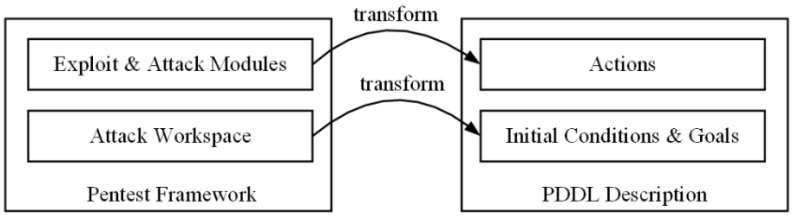

Figure 1: Pentest framework mapping to the PDDL description

```
(:action MS17-010_EternalBlue_SMB
:parameters (?srcIP - h ?dstIP - h)
:precondition (and
(host_connectivity ?srcIP ?dstIP)
(compromised ?srcIP)
(isWindows ?dstIP)
(isWindows_7 ?dstIP)
(isportopen_443 ?dstIP))
:effect (and (compromised ?dstIP)))
```

(a)

```
  ……
(:objects ……
attacker - h)
(host1 - h))
(:INIT ……
(compromised attacker)
(isWindows host1)
(isWindows_7 host1)
(isportopen_443 host1)  )
```

(b)

Figure 2: Example of a PDDL description

Furthermore, the domain description can be obtained with vulnerability libraries and pentest tools, such as Common Vulnerabilities and Exposures (CVE) and Metasploit. An attack action consists of the corresponding name, preconditions, effects, and cost. For example, the attack action of Eternal Blue vulnerability can be represented as shown in Fig. 2(a).

The task description includes initial conditions and pentest targets, which are expressed by first-order predicate logic. The initial conditions include the node states, network interconnections, and configurations of hosts, and the target can usually represent the intrusion into a host. The initial information for a network can be represented as shown in Fig. 2(b).

3.2 Pentest information graph

The pentest information graph is a directed acyclic graph that can be represented as a tuple $G =< V, E, O >$, where:

- $V = \{v_{start}, v_{goal}, v_1, v_2, \ldots\}$ is a finite set of nodes in the graph that represents all the hosts in the network, v_{start} is the node that represents the attacker when pentest starts, and v_{goal} is the goal to achieve;
- $E = \{e_{ij} \mid v_i, v_j \in V, i \neq j\}$ is a finite set of edges in the graph that represents the connected relations between hosts. The edge start point is the one that is closest to the attacker, and the distance can be referred to as a hop. The other point is the end point;
- $O = \{o_1, o_2, o_3, \ldots\}$ is a finite set of all attack actions in the graph.

A node can be represented as $v_i = \{id, rhs, g\}$, where id is the identifier, rhs indicates the minimal accumulated cost from the goal to the current node via the corresponding successor, and g indicates the accumulated cost from the goal to the current node. Additional details of rhs and g are given in Section 3.

An action can be represented as $o = \{name, pre, eff, cost\}$, where $name$ is used to identify different actions; pre represents the preconditions that must be satisfied before execution; eff is the effect of the action; and $cost$ is the price associated with performing the action.

An edge can be represented as $e_{ij} =< Ops, cost, minop >$, which starts from v_i and ends at v_j. $Ops = \{o_1, o_2, o_3, \ldots\}$ is a finite set of available attack actions for host v_j, which can be obtained by reasoning according to the host configurations and action preconditions; $cost$ is the minimal action cost among those for all Ops; and the corresponding action is $minop$.

$$e_{ij}.minop = argmin_{o \in e_{ij}.Ops}(cost(o)) \tag{1}$$

$$e_{ij}.cost = cost(e_{ij}.minop) \tag{2}$$

The steps required to build the pentest information map are as follows.

- Obtaining all attack actions: All the actions can be obtained according to domain.pddl.
- Determining nodes: All the hosts and configurations are listed in problem.pddl, which includes the IP address, operating system type, port situations, software configurations, etc.
- Determining edges: Edges are primarily formed based on the connectivity relationships between hosts. Then, the Ops, $minop$ and $cost$ attributes of edges are determined based on host configuration and available exploitation actions.

By constructing the pentest information graph, the initial information is well organized and can be used for establishing potential attack actions. In this way, irrelevant actions do not have to be considered, so the planning space is greatly reduced. The pentest information graph compactly shows known information and establishes a "map" to provide a suitable data structure for subsequent planning algorithms, thus making it ideal for practical applications.

3.3 APU-D lite algorithm*

Considering the insufficiency of the D* Lite algorithm, this paper proposes the APU-D* Lite attack planning algorithm. This algorithm includes the following improvements based on the original method.

- Add planning steps for attack actions. The corresponding result is a node-action alternating sequence.
- Add function $UpdateEdge(e, o)$ to update the edges in the pentest information graph when an attack fails.
- Design a heuristic function suitable for attack planning. Maintain $cost_f$, the global minimum cost of actions and hop, the distance from the start point to the current node, which are used to calculate the heuristic value:

$$h(v)=\begin{cases}0 & v=v_{start}\\ (v.hop-1)\cdot cost_f+e.cost & v\neq v_{start}\end{cases} \tag{3}$$

In the pentest information graph, V is the finite set of all the nodes, for which v_{start} is the start point and v_{goal} is the target. For any $v\in V$, $pre(v)$ and $succ(v)$ represent the predecessors and successors of v, respectively, and satisfies $pre(v)\subset V$ and $succ(v)\subset V$. $c(v,v')\in[0,\infty)$ represents the path cost between v and v'. Specifically, $c(v,v')=\infty$ indicates that there is no path between v and v'.

APU-D* Lite involves reverse searching. Therefore, $g(v)$ is the accumulative cost from v_{goal} to v. $rhs(v)$ indicates the minimal accumulative cost from v_{goal} to v for $v'\in succ(v)$, which can be calculated as follows.

$$rhs(v)=\begin{cases}0 & v=v_{goal}\\ min_{v'\in succ(v)}(g(v')+c(v,v')) & v\neq v_{goal}\end{cases} \tag{4}$$

The nodes in the graph can be divided into two classes according to the relation between $g(v)$ and $rhs(v)$. If $g(v)=rhs(v)$, v is locally consistent; otherwise, it is locally inconsistent. Furthermore, inconsistency falls into two classes: underconsistency and overconsistency. Underconsistency occurs when $g(v)<rhs(v)$, and overconsistency occurs when $g(v)>rhs(v)$. Underconsistency indicates that the cost of reaching the node increases, thereby suggesting that there is an obstacle in the path and replanning is needed.

The priority queue U is used to deal with the inconsistent nodes to be extended. The relation $key(v)=[key_1(v),key_2(v)]$ is used as the priority, and all the nodes in U are sorted in ascending order. For any two nodes v and v', only when $key_1(v)<key_1(v')$ or $key_1(v)=key_1(v')$ and $key_2(v)<key_2(v')$, $key(v)$ is less than or equal to $key(v')$. $key(v)$ can be calculated as follows.

$$\begin{cases}key_1(v)=min(g(v),rhs(v))+h(v)\\ key_2(v)=min(g(v),rhs(v))\end{cases} \tag{5}$$

where $h(v)$ is a heuristic function, which is shown in Eq. (3), representing the estimated cost from the start node to the current node. The heuristic function must satisfy the two conditions below.

$$\begin{cases}h(v_{start})=0\\ \forall v\in V,v'\in pre(v),h(v)\le c(v,v')+h(v')\end{cases} \tag{6}$$

Clearly, Eq. (3) satisfies the first condition. Since $cost_f$ is the global minimum action cost, $e_{v'}.cost=c(v,v')$ and $e_{v'}.cost\ge cost_f$ hold. Therefore, Eq. (3) satisfies the conditions in Eq. (6).

Because the attack action may fail during actual execution, the APU-D* Lite algorithm includes the *UpdateEdge*() function. If an action fails, the *UpdateEdge*() function is called to double the action cost instead of directly setting the cost to infinity, and the *minop* and *cost* values of that edge will be recalculated. Therefore, the cost of an action exponentially increases as the number of failures increases, and the action with the smallest value may change after a failure. With this mechanism, it is possible to avoid repeated action failure, and it is not immediately assumed that the action failed for accidental reasons.

The pseudocode of the APU-D* Lite algorithm can be seen in Algorithm 1.

Algorithm 1 APU-D* Lite

Input: Pentest Information Graph G
Output: AttackPath

```
1: function CalcKey(v)
2:     return [min(g(v), rhs(s)) + h(v_start, v); min(g(v), rhs(v))]
3: function Initialize()
4:     U = ∅
5:     for all v ∈ G.Nodes() do
6:         rhs(v) = g(v) = ∞
7:     for all e ∈ G.Edges() do
8:         Calculate e.minop and e.cost
9:     rhs(v_goal) = ∞
10:    U.insert(v_goal,CalcKey(v_goal))
11: function UpdateVertex(u)
12:    if u ≠ v_goal then
13:        rhs(u) = min_{u'∈succ(u)}(g(u') + (e_{uu'}.cost)
14:        if u ∈ U then U.remove(u)
15:        if g(u) ≠ rhs(u) then
16:            U.insert(u,CalcKey(u))
17: function UpdateEdge(e,o)
18:    o.cost = o.cost * 2
19:    e.minop = arg min_{o'∈e.Ops}(a.cost)
20:    e.cost = e.minop.cost
21: function ComputeShortestPath()
22:    while U.TopKey() < CalcKey(v_start) or rhs(v_start) ≠ g(v_start) do
23:        u = U.Pop()
24:        if g(u) > rhs(u) then
25:            g(u) = rhs(u)
26:            for all s ∈ pred(u) do UpdateVertex(s)
27:        else
28:            g(u) = ∞
29:            for all s ∈ pred(u) ∪ u do
30:                UpdateVertex(s)
31: function Main()
32:    Initialize()
33:    ComputeShortestPath()
34:    while v_start ≠ v_goal do
35:        v_next = arg min_{u'∈succ(u)}(g(u') + (e_{uu'}.cost)
36:        Execute e_{v_start v_next}.minop
37:        if Success then
38:            v_start = v_next
39:        else
40:            UpdateEdge(e_{v_start v_next},e_{v_start v_next}.minop)
41:            UpdateVertex(v_next)
42:            for all s ∈ U do
43:                U.update(s,CalcKey(s))
44:            ComputeShortestPath()
```

The APU-D* Lite algorithm first calls function *Initialize*() , which creates an empty priority queue U , sets g and rhs for all nodes to infinity, and calculates the minimum cost and optimal action for all edges. Next, *ComputerShortestPath*() is called to calculate the shortest path. Starting from v_{goal} , this function continuously processes the nodes in the priority queue U in order and updates g for each node. When the function ends, the shortest path is stored in the g sets of nodes. Then, in the *Main*() function, the attacker starts from v_{start} ; the next node is then calculated by Eq. (7), and the *minop* function of $e_{v_{start}v_{next}}$ is executed.

$$v_{next} = argmin_{u' \in succ(u)}(g(u') + (e_{uu'}.cost) \tag{7}$$

If successful, v_{start} is updated by v_{next} ; otherwise, $UpdateVertex(v_{next})$ and $UpdateEdge(e_{v_{start}v_{next}}, minop)$ are called to update the configuration of the related edge and nodes.

4 Experiment

To verify the availability and efficiency of APU-D* Lite, this paper compared APU-D* Lite with the A*, FF, HSP algorithms under different levels of uncertainty. Here, the success rate of attack action is used to represent the uncertainty level. For a typical internal network, APU-D* Lite, A*, FF, and HSP are compared based on the attack path cost, the number of extended nodes, the replanning time, and memory usage.

The attack path cost is the sum of the action costs for an attack path. The explored nodes is the number of the visited nodes during the run time. The replanning time indicates how many times the function *CalculateShortestPath*() is called, which is related to action failures and increases with decreasing action success rate. The memory usage reflects the memory consumption of the algorithm.

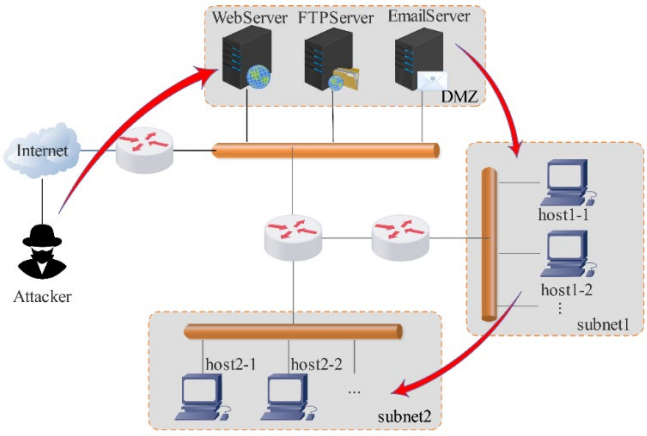

Figure 3: Network topology

The network topology of this experiment is shown in Fig. 3. The attacker is initially connected to the DMZ zone of the internal network, and the goal is to compromise host 2-2. The internal network includes the DMZ zone, subnet 1 and subnet 2. The DMZ is connected to subnet 1, and the subnet 2 network is only accessible via subnet 1. There is a possibility for an attacker to invade the internal network. First, the attacker needs to compromise one of the hosts in the DMZ and then use the compromised host as a springboard to attack hosts in the subnet 1 network. Likewise, attackers could further attack subnet 2 and eventually compromise the target host 2-2.

The configurations of each host are listed in Tab. 1.

Table 1: Host configurations

Host	Vulnerabilities
Attacker	—
MailServer	CVE-2005-2287
FTPServer	CVE-2008-2161, CVE-2009-3023, CVE-2012-6066
WebServer	CVE-2009-1151, CVE-2012-5613
Host 1-1	CVE-2010-3964, CVE-2009-0183, CVE-2007-5243
Host 1-2	CVE-2010-3964, CVE-2012-6066
Host 2-1	CVE-2012-1182, CVE-2017-7494
Host 2-2	CVE-2010-2729, CVE-2017-8895, CVE-2012-2288

In this experiment, a total of 6 different exploit success rates were set from 100% to 50%, and four algorithms were compared based on different attack action success rates. Each group of experiments involved 12,000 repeated tests, and abnormal results were excluded. The experimental results are as follows.

As shown in Fig. 4(a), since the failure of an attack action leads to an increase in the action cost, when the action success rate decreases continuously and the number of action failures increases, the total cost of the attack path also increases. In the static environment, when the action success rate is 100%, the total path cost of each algorithm is the same, indicating that the results of the APU-D* Lite algorithm are not worse than those of other classical algorithms in deterministic environments. As the success rate declines, the total path cost of each algorithm increases. The path cost of FF increases fastest, followed by those of the A* and HSP algorithms. The cost of the APU-D* Lite algorithm increases the slowest.

As shown in Fig. 4(b), the number of extended nodes increases as the success rate decreases. The number of extended nodes of FF is initially significantly larger than the numbers for the other three algorithms, and this number also grows faster than those for the other algorithms. In the deterministic environment, the A* and HSP algorithms have the fewest extended nodes, and the APU-D* Lite algorithm is centered. However, as the success rate of attack actions decreases, the number of extended nodes in the A* and HSP algorithms increases slightly, but the nodes in the APU-D* Lite algorithm remain almost unchanged. When the attack success rate is less than 70%, the number of extended nodes in the A* and HSP algorithms begins to exceed the number in APU-D* Lite.

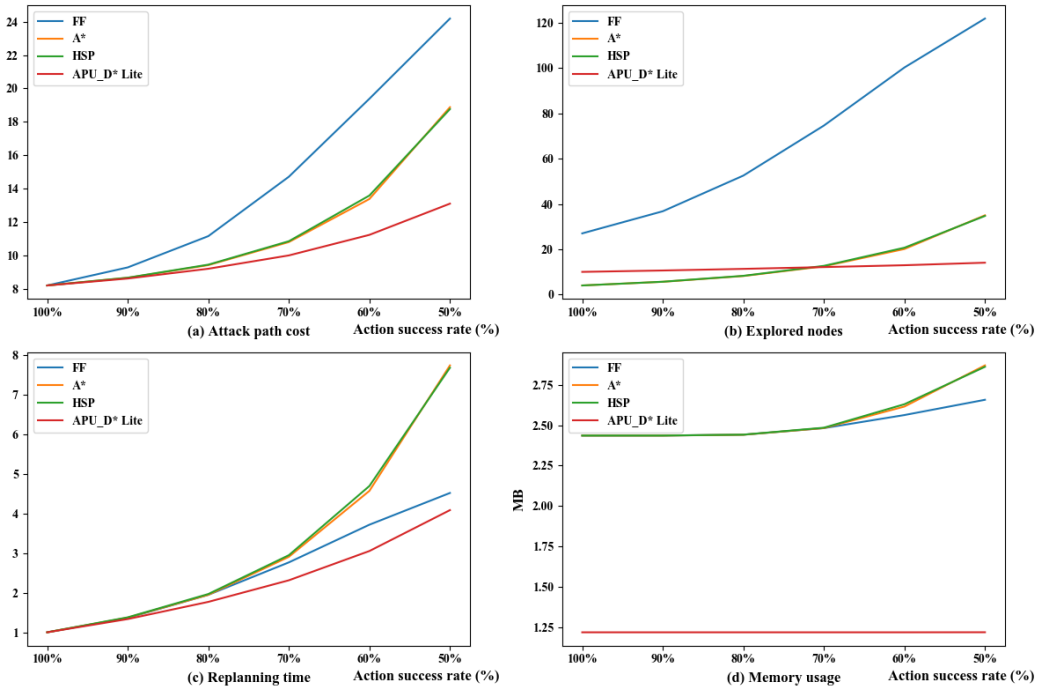

Figure 4: Comparisons of four planning algorithms

As shown in Fig. 4(c), in the deterministic environment, the replanning time for each algorithm is 1. As the success rate of attack actions decreases, the number of action failures increases, as does the replanning time for each algorithm. The A* and HSP algorithms have the fastest growth rates, and the FF algorithm and APU-D* Lite have similar growth rates. The growth rate of the FF algorithm is slightly larger than that of the APU-D* Lite algorithm. When the attack success rate drops to 50%, the performance of APU-D* Lite is similar to that of FF. The replanning times of the A* and HSP algorithms are approximately twice that of the APU-D*Lite algorithm.

As shown in Fig. 4(d), the memory usage of each algorithm is not affected by the change in the attack success rate. When the attack success rate is between 100% and 80%, the memory usage of the A*, HSP, and FF algorithms largely stays the same; when the success rate is less than 80%, the memory usage of these three algorithms starts to increase slightly. However, the memory usage of the APU-D* Lite does not notably increase as the success rate decreases. Moreover, the memory usage of the APU-D* Lite algorithm is always approximately half that of the other algorithms.

Since the APU-D* Lite algorithm adopts the idea of incremental planning, the start point is continuously updated to the current position. When an action fails, the algorithm does not need to consider the previous paths and only needs to adjust the subsequent path starting from the current location. Therefore, the APU-D* Lite algorithm does not have to start from the initial position, so the problem size tends to decrease, as does the length of the planning result. These factors allow the APU-D* Lite algorithm to experience fewer failures than the other algorithms at the same attack action failure rate, which is further reflected in the metrics of

the total path cost, the number of extended nodes and replanning time. These results illustrate that the APU-D* Lite algorithm is more efficient in uncertain environments.

The above experimental results show that in the deterministic environment, the results and efficiency of APU-D* Lite reach or exceed the levels of the other classical planning algorithms, except for the number of extended nodes, for which the proposed method is slightly inferior to the A* and HSP algorithms. Moreover, as environmental uncertainty increases, the APU-D* Lite algorithm becomes more efficient and adaptable.

5 Conclusion

Based on the characteristics of pentesting and the relevant efficiency requirements, this paper proposes the APU-D* Lite algorithm for attack planning in an uncertain environment. To express the network information, we used the PDDL for the domain description and develop a pentest information graph to organize and extract relevant information. This information functions as the input of the APU-D* Lite algorithm. The APU-D* Lite algorithm is based on an incremental heuristic search. By adding attack action planning to the traditional method, it is more practical for application. In a comparison with off-the-shelf algorithms, the availability and effectiveness of APU-D* Lite under uncertain conditions are verified.

Funding Statement: The author(s) received no specific funding for this study.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

Bertoglio, D. D.; Zorzo, A. F. (2017): Overview and open issues on penetration test. *Journal of the Brazilian Computer Society*, vol. 23, no. 1, pp. 1-16.

Boddy, M.; Gohde, J.; Haigh, T.; Harp, S. (2005): Course of action generation for cyber security using classical planning. *Proceedings of the 15th International Conference on Automated Planning and Scheduling*, pp. 12-21.

Bozic, J.; Wotawa, F. (2017): Planning the attack! Or how to use AI in security testing? *Proceedings of First International Workshop on AI in Security*, vol. 50.

Chen, X.; Chen, X. M.; Zhang, J. (2014): The dynamic path planning of UAV based on A* algorithm. *Applied Mechanics and Materials*, vol. 494, pp. 1094-1097.

Felderer, M.; Büchler, M.; Johns, M.; Brucker, A. D.; Breu, R. et al. (2016): Security testing: a survey. *Advances in Computers*, vol. 101, pp. 1-51.

Futoransky, A.; Notarfrancesco, L.; Richarte, G.; Sarraute, C. (2003): Building computer network attacks. *CoreLabs Technical Report*.

https://arxiv.org/pdf/1006.1916.pdf.

Grant, T. (2018): Speeding up planning of cyber attacks using AI techniques: state of the art. *Proceedings of 13th International Conference on Cyber Warfare & Security*, no. 3, pp. 235-244.

Koenig, S.; Likhachev, M. (2005): Fast replanning for navigation in unknown terrain. *IEEE Transactions on Robotics*, vol. 21, no. 3, pp. 354-363.

Obes, J. L.; Sarraute, C.; Richarte, G. (2013): Attack planning in the real world. *Proceedings of the AAAI Workshop on Intelligent Security*, pp. 10-17.

Pellier, D.; Fiorino, H. (2018): PDDL4J: a planning domain description library for java. *Journal of Experimental and Theoretical Artificial Intelligence*, vol. 30, no. 1, pp. 143-176.

Ramos, A.; Lazar, M.; Filho, R. H.; Rodrigues, J. J. P. C. (2017): Model-based quantitative network security metrics: a survey. *IEEE Communications Surveys and Tutorials*, vol. 19, no. 4, pp. 2704-2734.

Sarraute, C. (2009): New algorithms for attack planning. *FRHACK Conference*.

Sarraute, C. (2013): Automated attack planning. https://arxiv.org/abs/1307.7808.

Sarraute, C.; Buffet, O.; Hoffmann, J. (2012): POMDPs make better hackers: accounting for uncertainty in penetration testing. *Proceedings of Twenty-Sixth AAAI Conference on Artificial Intelligence*, pp. 1816-1824.

Sarraute, C.; Buffet, O.; Hoffmann, J. (2013): Penetration testing==POMDP solving? *Proceedings of the 3rd Workshop on Intelligent Security*, pp. 66-73.

Sarraute, C.; Richarte, G.; Obes, J. L. (2011): An algorithm to find optimal attack paths in nondeterministic scenarios. *Proceedings of the ACM Conference on Computer and Communications Security*, pp. 71-79.

Xiong, B.; Yang, K.; Zhao, J. Y.; Li, K. Q. (2017): Robust dynamic network traffic partitioning against malicious attacks. *Journal of Network and Computer Applications*, vol. 87, pp. 20-31.

Xu, W.; Tao, Y.; Yang, C.; Chen, H. (2019): MSICST: Multiple-scenario industrial control system testbed for security research. *Computers, Materials and Continua*, vol. 60, no. 2, pp. 691-705.