New Goal Recognition Algorithms Using Attack Graphs

Reuth Mirsky(✉), Ya'ar Shalom(✉), Ahmad Majadly(✉), Kobi Gal(✉), Rami Puzis(✉), and Ariel Felner(✉)

Ben-Gurion University of the Negev, Beersheba, Israel
{dekelr,yaarsh,ahmadmaj,kobig,puzis,felner}@bgu.ac.il

Abstract. Goal recognition is the task of inferring the goal of an actor given its observed actions. Attack graphs are a common representation of assets, vulnerabilities, and exploits used for analysis of potential intrusions in computer networks. This paper introduces new goal recognition algorithms on attack graphs. The main challenges involving goal recognition in cyber security include dealing with noisy and partial observations as well as the need for fast, near-real-time performance. To this end we propose improvements to existing planning-based algorithms for goal recognition, reducing their time complexity and allowing them to handle noisy observations. We also introduce two new metric-based algorithms for goal recognition. Experimental results show that the metric based algorithms improve performance when compared to the planning based algorithms, in terms of accuracy and runtime, thus enabling goal recognition to be carried out in near-real-time. These algorithms can potentially improve both risk management and alert correlation mechanisms for intrusion detection.

1 Introduction

Attack Graphs combine vulnerabilities, exploits, assets, and connectivity among the nodes in a computer network into a singe concise model that encompasses all attack scenarios where an attacker can reach its goal [4,30]. Attack graphs have been used in past to asses the security risk of a computer network [33], improve resilience of the network by patching vulnerabilities that are the most important for the overall network security, optimize sensors placement [29], correlate alerts for efficient intrusion detection [37], penetration testing [39] and more.

One of the most challenging and yet important problems in security operations management is reconstructing the attack scenario to understand how did the attacker breach the network and recognizing the adversary's goals to anticipate his future actions [2,22,34]. Goal recognition is a key AI problem that deals with reasoning about an actor's goals according to a sequence of observed actions [45]. This paper studies the goal recognition problem for time-critical security settings where up to date insights about the possible attack goals and yet unobserved attacker's actions should be presented to the analyst by the recognizer.

S. Dolev et al. (Eds.): CSCML 2019, LNCS 11527, pp. 260–278, 2019.
https://doi.org/10.1007/978-3-030-20951-3_23

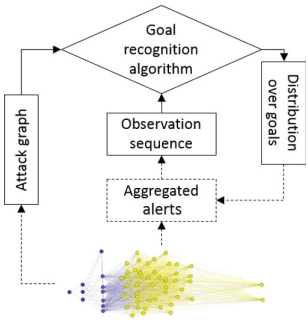

Fig. 1. Components of the proposed approach.

Consider an attacker that aims to take control of a target node in a communication network. Following the initial penetration an attacker establishes presence in the victim computer network and perform a series of actions consisting of lateral movements, privilege escalations, etc. up until reaching the target node. These actions represent its attack plan. An Intrusion Detection System (IDS) monitors the nodes in the network and detects malicious behavior of potential attackers in with certain accuracy. The alerts produced by IDS are mapped to their respective hosts. Preliminary inspection of logs and sensory information may infer the tools and exploits used in the course of attack execution. Given these (noisy) observations the goal recognition task is to infer the most probable goal (target node) of the attacker as well as its plan for achieving this goal.

There are two challenges facing goal recognition in this setting. First, the recognition must be done in near real time in order to be able to counter the attack as soon as possible. Thus, we seek algorithms with short running times. However, existing plan recognizers rely on making calls to costly planning algorithms during runtime. Second, the incorrect observations, including false positive/false negative alerts, hinder the goal recognition task using standard algorithms, thus the proposed goal recognition algorithm must be robust to noisy and partial observations.

While several works have tried to reason about probabilistic goal recognition for cyber security, they were evaluated theoretically [13,15] or using toy problems [6,25], all using various relaxations that do not apply to real time settings. In addition, while there exist planning algorithms for goal recognition [32,35,42,45], they require running off-the-shelf planners online as part of the recognition task of determining the most likely goals. The planner, which is the computational bottleneck of these approaches, is called at least once for each goal so as to be able to compare the observed actions to the optimal plan to that goal. This paper enhances existing approaches as well as introduces new approaches for goal recognition, which reduce the number of times the planner is called, making them significantly more efficient, without hindering their empirical performance. The main advantage of our new approaches is that they reduce and even eliminate the need to call planners online.

Figure 1 summarizes our proposed framework, working in a pipeline with existing intrusion detection approaches that correlate and aggregate alerts [1, 8]. A PDDL representation of the attack graph (Fig. 1 left) is generated from the network (Fig. 1 bottom). Alerts generated by an IDS deployed within the network are compiled into observations (Fig. 1 middle). The inputs for the goal recognition algorithm (Fig. 1 top) are the attack graphs and the alerts. The goal recognition algorithm outputs a distribution over the goals of the potential attacker (Fig. 1 right), which can be potentially used by the IDS to refine future alerts. The components presented in this work are represented with solid lines, where other components are represented using dashed lines.

This paper makes the following contributions:

1. We enhance the benchmark planning-based algorithm for goal recognition [35] to real time settings by reducing its time complexity and allowing it to better handle noisy and missing observations. This is described in Sect. 4;
2. In Sect. 5 we introduce a new family of goal recognition algorithms that completely avoid running planners in the recognition phase. Importantly, our algorithms execute such planners in a preprocessing phase and their output can be shared by many online queries on the same network. We describe novel metrics to compute the distance between the actor's observed actions and the optimal plan to the goal (calculated in the preprocessing phase);
3. Finally (in Sect. 6) we experimented with of all these algorithms on attack graphs representing a real world network and analyze the pros and cons of each of the approaches. Our metric-based algorithms always run faster than the planning-based approaches, even on noisy observations. In addition, they provide equal or better predictions except for input with missing observations, where the planning-based approaches are superior.

2 Goal Recognition

We follow a recent line of research known as *goal recognition as planning* [35]. The input is a planning theory, usually described in PDDL, and a set of possible goals. The output is one of the goals or a distribution over all possible goals. This opened a line of algorithms that execute planners to find plans to the goal with or without the observations [12,23,36,41,42,44]. Some works added landmark-based heuristics [31] and cost propagation [10] to the planning algorithms.

We next briefly mention some basic concepts in this research direction, as presented by Ramírez and Geffner [35], simplified for brevity. In a goal recognition problem, there is an observer and an actor, and the observer needs to infer the goal of the actor. The input to the observer is a theory description and a sequence of observations.

Definition 1. *A* ***Theory Description*** *(D) is a tuple* $D = \langle S, A, G, I, C \rangle$*, where* S *is a set of states,* A *is a set of actions representing transitions from state to state,* $G \subseteq S$ *is the goals the actor can achieve and* $I \in S$ *is the initial state.* C *is a cost function mapping any action in* A *to a real number.*

Each state in S is represented by a set of predicates. All predicates in the state are assumed to hold true in the environment, and predicates that do not appear in the state are assumed to be false. Each action a in A has preconditions and effects, describing what should be true in the state before executing a and after its execution, respectively. They are both represented as a set of predicates. The actor is assumed to plan by choosing a goal and then carrying out a plan for reaching this goal.

Definition 2. *A plan for achieving a goal $g \in G$ is a sequence of actions $a_1, \ldots, a_n$ such that:*

- *The execution starts at I (the predicates in the precondition of the first action are all in I).*
- *For each $a_i \in A$, the state before the execution contains all of the predicates in a_i's precondition.*
- *After executing all actions in the sequence, all the predicates in g are true.*

Given a theory with n possible goals, $g_1, \ldots, g_n$ and an observation sequence $O = o_1, \ldots, o_t$, a solution to the goal recognition problem is a distribution over the goals such that $p(g_i \mid O)$ is the likelihood that the actor is pursuing goal g_i given O.

In order to suit a real world scenario we make the following assumptions. First, we allow noisy observation sequences by assuming that it contains false alerts, i.e., the input may falsely contain some observed actions. Second, we assume that there are some missing observations, i.e., some actions of the actor are not reported in the input sequence.

2.1 Other Goal Recognition Works

While we use the *goal recognition as planning approach*, we note that there are different approaches to represent a theory in goal recognition, including policy-based, library-based and others [3,45].

For example, YAPPR [14] takes as input a plan library and can output both a distribution over the goals of the actor and predictions about future actions. DOPLAR [20] extended YAPPR using probabilistic reasoning to reach better performance, at the cost of completeness.

These works require to model the settings using a plan library, which is difficult to elicit and susceptible to faults. Bui [7] uses particle filtering to provide approximate solutions to goal recognition problems. These works all rely on a model of the plans or strategies that the actor can execute.

While adversarial goal recognition was investigated in the past [13,21,22,25], none of the works mentioned above have combined a PDDL description that can represent an attack graph in order to recognize the goals of an attacker. Masters and Sardina [24] recently presented an intersection of deception with goal recognition in the context of path-planning. We next dive deeper into the attack graph representation.

3 Attack Graphs

3.1 Definitions and Background

An attack graph is a description of a network that comprises hosts, their vulnerabilities, and their connectivity to one another [5]. Formally,

Definition 3. *An attack graph A is an undirected graph $A = \langle V, E \rangle$ where:*

- *V is a set of vertices, such that each $v \in V$ represents a single host in the network. Each host has a single operating system (OS) and it contains a list of installed software. Each piece of such software can hold a vulnerability, which can be exploited by an attacker to gain control over the host.*
- *E is a set of edges, representing the connectivity of the hosts.*

A potential attacker traverses over the network by reaching a host and gaining control over it. After gaining control over a host, all of its neighbors become accessible to the attacker. The attacker then may choose to traverse one of these edges and try to take control over the corresponding neighbor. Gaining control requires to exploit a specific vulnerability, which depends on a combination of the OS and software available on that host. In the real world, vulnerabilities might be different from one computer to another even if they share similar OS and software, and there is no way to detect the specific vulnerabilities in advance.

Swiler et al. [43] presented an automatic tool for generating an attack graph representation of a computer network. This work has been extended to handle networks at a larger scale [27]. Later, Noel and Jajodia [29] presented a method for optimizing the placement of intrusion detection system (IDS) sensors and prioritizing IDS alerts using attack graph analysis.

Recent studies continued improving the attack graph generation and analysis approaches toward automated pentesting [9,16,18,40].

Hoffmann [18] discusses the suitability of the "CyberSecurity" benchmark at the International Planning Competition (IPC) and analyzes the importance of factoring uncertainty when it comes to understanding the behavior of potential hacking actors. However, analysis of an attack graph has only focused on mapping of vulnerabilities and pentesting, rather than on realtime intrusion detection.

A different line of research does utilize the attack graphs for prioritizing alerts generated by IDS. These works use attack graphs for alert correlations [28,37,46,47]. Modern attacks are getting more complex and the number of alerts emerging from the system increases significantly. Reasoning about temporal order and causality of alerts, allows detecting false negative and false positive alerts more efficiently [28,37]. This line of research has done a great deal in refining the alerts, but did not reason about the ultimate goal of the attacker.

In this work, we feed the alerts as observations into a goal recognizer with the attack graph as the underlying theory description. Thus, the attack graph is used directly for online goal recognition. Under these settings, the attacker might wish to obfuscate the attack by executing irrelevant actions, or the alert aggregation might produce false positive alerts which are not part of a valid attack.

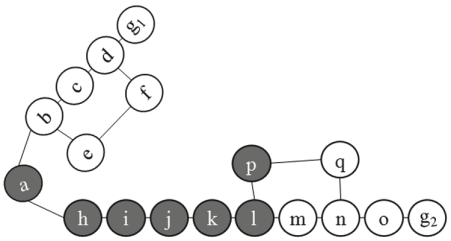

Fig. 2. Simple network example.

3.2 Attack Graphs as PDDL

The theory we use is based on the work of Shmaryahu [38], and was compiled to match the requirements of our goal recognition algorithms. The theory has 4 types of variables, $host, os, sw, vuln$, representing hosts, operating systems, software and vulnerabilities respectively.

These types are used to define 7 parameterized predicates:

1. (LNK, h_1, h_2) - true if h_1 and h_2 are linked by an edge on the attack graph.
2. $(CNTRL, h)$ - true if the attacker controls h.
3. (SW, h, s) - true if the software s is installed on the host h.
4. (OS, h, o) - true if host h runs operating system o.
5. $(VLNR, v, h)$ - true if vulnerability v exists in host h.
6. $(MATCH, o, s, v)$ - true if vulnerability v exists in software s under operating system o.
7. $(ACCESS, h)$ - true if host h is accessible by the attacker.

In addition, the PDDL theory defines the actions the attacker can perform:

- $GetAccess(h_1, h_2)$, which has the preconditions that h_1 and h_2 are linked and that the attacker is controlling h_1. The effect is that host h_2 is now accessible to the attacker.
- $RunSW(h, sw, os, v)$ is the action used to run some software sw on the system os of host h, that might exploit a vulnerability v. This combination might be a valid software usage, or an exploit.

Using these actions, reaching a goal is a process of traversing to a host, then running some software on it in order to gain control over it. This is done repeatedly, until taking control on a connected path to the goal host.

Throughout the paper we will use the following running example from Fig. 2. This example includes a network with 18 nodes, each node represents a host in the network, and edges represent connectivity between nodes. An attacker commences at host a, with the goal of attacking node g_1 or g_2. Observations (hosts that are controlled by the attacker) are shown as dark nodes. The cost of connecting (*GetAccess* action) and gaining control over a node (*RunSW* action) is 1. Note that there is no notion of backtracking because there is no need to undo a connection. Once control has been established for a given node, it can always be used to connect to hosts at adjacent nodes.

4 Planning-Based Algorithms for GR

All of the methods we present rely on calling a planning algorithm to compute the cost of potential plans that the actor may be pursuing. Running the planner is the computational bottleneck of all these approaches, especially in realtime systems as the one that is the focus of this paper. The main advantage of our new approaches is that they reduce and even eliminate the need to call such planners online. We distinguish between calls to the planner that are done *offline*, that is in a preprocessing phase before observations have been received and *online*, during recognition.

We first describe two variants of the benchmark planning-based algorithm by Ramírez and Geffner [35] (denoted R&G). Then, we will introduce our further enhancement to this approach. In the next section we will introduce a new metric-based approach for GR.

Table 1. Number of calls to planner by each method

	offline	online
R&G		$2\|G\|$
M&S	$\|G\|$	$\|G\|$
R&G+	$\|G\|$	$\|G\|$
PED	$\|G\|$	
APC	$\|G\|$	

4.1 The R&G Approach

Ramírez and Geffner [35] introduced the first planning-based algorithm for goal recognition (R&G).

Given a theory D and an observation sequence O, R&G introduces a modified theory that takes into account O:

Definition 4. *A **Modified Theory** (D') given D and O is a tuple $D' = \langle S, A', G, I, C \rangle$, where for each action $a \in A$, there is a parallel action in A' with a possible extra predicate in its effects list: p_a when a is the first observation in O and $p_b \rightarrow p_a$ when b is the action that immediately precedes a in O.*

In a transformed theory and given a plan π, the fluent p_a is true after executing π if and only if π satisfies all observations in O until a. Using D', R&G use an off-the-shelf classical planner in order to calculate the following two quantities:

1. The minimal cost of achieving the goal g_i such that the plan satisfies all observations in O. This cost is denoted $C_i(O)$;

2. The minimal cost of achieving the goal g_i such that it does not satisfy the observation sequence O. This cost is denoted $C_i(\neg O)$.

The term $L(g_i \mid O)$ measures the proximity (in terms of cost) between a plan that is directly based on the observations and the optimal plan for reaching g_i and is defined to be:

$$L(g_i \mid O) = C_i(O) - C_i(\neg O) \tag{1}$$

Intuitively, as the difference grows larger, this increases the likelihood of goal g_i given O. Given a set of observations $O = \{o_1, \ldots, o_t\}$, the R&G algorithm computes the terms $C_i(O)$ and $C_i(\neg O)$ by calling the planner once for each goal g_i given O during the recognition phase. For full details we refer to their paper. The total number of calls to the planner is $2|G|$ (see Table 1).

The score given for goal g_i given the observation sequence O is defined as:

$$p(g_i \mid O) \cong \frac{1}{e^{\beta \cdot L(g_i|O)} + 1} \tag{2}$$

Where $0 \leq \beta \leq 1$ is used to soften the impact of observations that deviate from the optimal plan. After $p(g_i \mid O)$ is calculated for all $g_i \in G$, they are normalized to provide a valid probability distribution.

Consider the running example from Fig. 2 and the observation sequence $O = \{a, h, i, j, k, l, p\}$. The optimal plan for g_1 is $P_1 = \{a, b, c, d, g_1\}$. P_1 is also the optimal plan that does not fully follow O (it does not visit any node in O except a). Similarly, the optimal plan for g_2 is $P_2 = \{a, h, i, j, k, l, m, n, o, g_2\}$. P_2 is also the optimal plan that does not fully follow O (it does not visit node p). We assume a stardard cost of 1 for gaining control over each of the nodes. Thus we have that $C_1(P_1) = 5$ and $C_2(P_2) = 10$. Assuming $\beta = 1$ (for the sake of the example), we get the following values:

$$L(g_1 \mid O) = C_1(O \cup \{b, c, d, g_1\}) - C_1(P_1) = 6$$
$$L(g_2 \mid O) = C_2(O \cup \{q, n, o, g_2\}) - C_2(P_2) = 1$$
$$p(g_1 \mid O) \cong \frac{1}{e^6 + 1} = 0.002, \quad p(g_2 \mid O) \cong \frac{1}{e^1 + 1} = 0.269$$

After normalization we get the goal probabilities $p(g_1 \mid O) = 0.007$ and $p(g_2 \mid O) = 0.993$.

4.2 The M&S Approach

Masters and Sardina [23] (M&S) suggested replacing the $C_i(\neg O)$ term with C_i, the minimal cost of achieving the goal g_i without reference to the observations. This term can be computed by calling the planner offline without referencing the observations. The resulting likelihood formula is as follows:

$$L(g_i \mid O) = C_i(O) - C_i \tag{3}$$

For example, in Fig. 2, C_2 is simply the cost of the optimal plan $C_2(P_2)$. The number of calls to the planner in this approach includes $|G|$ offline calls, for

computing C_i for each goal, and $|G|$ online calls, for computing $C_i(O)$ for each goal (see Table 1). M&S showed that using this approach significantly improves computation time with similar performance to R&G.

In our example, given the same observation sequence O, we get the same probability measures for the goal likelihoods of g_1 and g_2.

4.3 Improvement 1: Realtime Reasoning and Generalization (R&G+)

We can use the insight from M&S to improve the R&G approach for the case in which there is a single optimal plan. Let P_i be the optimal plan that is associated with C_i and computed offline. During the recognition process, we distinguish between two cases.

1. If $O \subseteq P_i$, that is, the observations in O are part of the optimal plan, then by definition of $C_i(O)$, we get that $C_i(O) = C_i$.
2. Otherwise, there is at least one observation in O that is not in the optimal plan. By definition of $C_i(\neg O)$ we get that $C_i(\neg O) = C_i$.

Note that it is always the case that C_i is a lower bound for both $C_i(O)$ and $C_i(\neg O)$.

We use the above insight to compute $L(g_i \mid O)$ using Eq. 1. We make a single offline call to the planner to compute C_i and a single online call to compute $C_i(O)$ or $C_i(\neg O)$ as needed (see Table 1). Determining whether $O \subseteq P_i$ can be done in linear time.

In our running example, suppose $O = \{a, h, i, j, k, l\}$. M&S will output the same plan P_2 twice, once for calculating C_2 (offline) and once for calculating $C_2(O)$ (online). Our new improvement will get that P_2 is in fact a plan that contains all observations from O, and will choose to calculate $C_2(\neg O)$, thus will be able to produce the same probability distribution as R&G, with the improved online runtime of M&S.

4.4 Improvement 2: Sunk Cost (SC)

In this approach we vary the R&G algorithm to consider the cost already attributed to the actor from executing the actions in O. Consider the following two scenarios based on the example in Fig. 2. In the first scenario, the observation sequence O includes the first six steps of a plan $\{a, h, i, j, k, l\}$, and a seventh observation $\{p\}$ that is not a part of the optimal plan (shown in the figure). Such an action may possibly be the cause of a faulty alert or due to a strategic actor (in Sect. 6 we consider the effect of noise on our algorithms). Here, we get that $C_2(O) = 11$ and $C_2(\neg O) = 10$ and the likelihood $L(g_2 \mid O)$ of goal g_2 is 1 (Eq. 1).

In the second scenario, an observation sequence O' includes $\{a\}$, the first out of the ten steps of the optimal plan for g_2, and then a non-related action $\{b\}$. We get that $C_2(O') = 6$, $C_2(\neg O') = 5$ and the likelihood $L(g_2 \mid O') = 1$. In both scenarios, the observation sequence includes a single non-related action, and the

score of goal g_2 is the same $p(g_2 \mid O) = p(g_2 \mid O') = 0.27$. However, if an actor has already put the effort and executed six out of ten steps, it is more plausible that g_2 is the goal, compared to the case when only one action was executed.

We can augment the original R&G formalism to reason about the cost that is already incurred by the actor to carry out the actions in the observation sequence. To this end we vary the β parameter, which was a constant value in the original R&G approach, as follows:

$$\beta = \frac{1}{min(C_i(\neg O), C_i(O))}. \tag{4}$$

Using this measure to compute the likelihood in Eq. 1 reduces the effect of noisy actions in a way that considers their proportion to the optimal plan and the observation sequence. Given this modification, we get a revised score over the goals by which goal g_2 is more likely under O than O'. ($p(g_2 \mid O) = 0.56, p(g_2 \mid O') = 0.44$).

Because all of the above approaches used Eq. 1 to compute the goal probability, we can modify the β parameter in any of them. In the empirical results we use R&G+ with the sunk-cost β.

5 Metric-Based Algorithms for GR

All of the three methods of the R&G family described above must run a planner at least once online, during the recognition phase. We propose a new paradigm for goal recognition that is based on distance metrics between the optimal plan (that is computed offline) and the observation sequence. It does not require online planner executions and so its online performance is much faster. Furthermore, as we show in the Empirical Section, in many cases their recognition is better than the R&G family. We begin with a naive distance metric.

5.1 Plan Edit Distance (PED)

The term *edit distance* usually refers to distance between sequences or sets by counting the number of edit actions needed to transform one sequence to the other. Here, we define an edit distance metric between action sequences which can be partial or complete plans.

The edit distance between two action sequences A and A' is defined as the number of actions that separate them.

$$D_{edit}(A, A') = |A \Delta A'| \tag{5}$$

Given an observation sequence O and a goal g_i we define a distance metric that only depends on the number of actions.

$$D(g_i, O) = D_{edit}(P_i, O) \tag{6}$$

where P_i and O are the action sequences in the optimal plan for achieving g_i (computed offline) and an observation sequence O, respectively. This metric prefers plans which have executed more steps that are part of the optimal plan. It is naive, as it does not reason about the order by which the actions have taken place, or the actual cost of executing the different actions. However, as we show in the Empirical Section, in some cases this metric can be effective.

The score of goal g_i (given O) is defined as:

$$p(g_i \mid O) \cong \frac{1}{D(P_i, O) + 1} \tag{7}$$

We normalize to get a probability distribution. We label this algorithm by Plan Edit Distance (PED).

To summarize, we need to call the planner $|G|$ times offline to compute the optimal plan P_i. Computing $D(g_i, O)$ can thus be done in linear time online.

Using this metric for the same sequence in the running example, $O = \{a, h, i, j, k, l, p\}$, we get that $D(O, P_1) = 10, D(O, P_2) = 5$ and the goal probabilities are $p(g_1 \mid O) = 0.352$ and $p(g_2 \mid O) = 0.647$.

5.2 Alternative Plan Cost

A more informed distance metric between the observations and the optimal plan is based on the Alternative Plan Cost (APC) of Felner et al. [11]. Their approach finds a minimal mapping from the states visited when executing O and the states visited when executing the optimal plan for a given goal g_i.

Let $S = (s_1, \ldots, s_n)$ and $S' = (s'_1, \ldots, s'_m)$ be two sequences of states. Out of the several mappings suggested by Felner et al. [11], we chose a Time Dimension Mapping $M(S, S')$ from the states in S to the states in S'. A time dimension mapping is monotonic, meaning that given a mapping M with $M(s_i) = s'_j$ and $M(s_{i+l}) = s'_k$ (for some $l > 0$), it must hold that $j < k$. This property guarantees consistency over the order in which the states are visited in S and S'.

Given some distance measure between individual states termed D_{base}, we choose the minimal monotonic mapping M that minimizes the average distance between S and S':

$$D_{\rightarrow}(S, S') = \min_{M: S \rightarrow S'} \frac{\sum_{i=1}^{|S|} D_{base}(s_i, M(s_i))}{|S|} \tag{8}$$

The APC approach can capture more complex relationships than PED, depending on the D_{base} metric that is selected to measure distance between states. Given an observation sequence O and a goal g_i, let S to be the sequence of states visited while executing O and S' be the sequence of stated visited while executing P_i (the optimal plan to g_i). The distance between O and P_i is defined as:

$$D_{\rightarrow}(g_i \mid O) = D_{\rightarrow}(S, S') \tag{9}$$

Consider the running example from Fig. 2. Reaching each node x is associated with a state s_x. The set of states in S is $\{s_a, s_h, s_i, s_j, s_k, s_l, s_p\}$ such that

each $s_i \in S$ corresponds to observation $o_i \in O$. The set of states in S' is $\{s_a, s_h, s_i, s_j, s_k, s_l, s_m, s_n, s_o, s_{g_2}\}$ such that each $s_i \in S'$ corresponds to action p_i in the optimal plan P_2. The distance D_{base} between individual states is defined as the cost of the shortest path between one state to the other. The minimal time dimension mapping from O to P_2 maps any state that also exists in the optimal plan to itself, and the state s_p that deviates from the optimal plan is mapped to s_m.

$$\forall s \in \{s_a, s_h, s_i, s_j, s_k, s_l\}, \quad M(s) = s$$
$$M(s_p) = s_m$$

We have that $D_{\rightarrow}(g_i \mid O) = 2/7$, because $D_{base}(s_p, s_m) = 2$ and 0 for all other s, and $|O| = 7$.

One issue with the definition $D_{\rightarrow}$ is that it does not reason about the differences in the size of the optimal plan and the observation sequence. In the above example, if the goal of the actor was m, we get the same mapping $D_{\rightarrow}$ as for goal g_2. To account for this issue we also need to consider the reverse mapping $D_{\leftarrow}(g_i|O)$ from S' to S. The APC distance metric between O and g_i averages both measures:

$$APC(g_i \mid O) = (D_{\rightarrow}(g_i \mid O) + D_{\leftarrow}(g_i \mid O))/2 \tag{10}$$

Each of the two mappings maintains the monotonicity property. Finally, the score of a goal g_i given an observation sequence O is

$$p(g_i \mid O) \cong \frac{1}{APC(g_i \mid O) + 1} \tag{11}$$

Again, this is normalized to get a probability distribution.

The minimal time dimension mapping from P_2 to O maps any state in the optimal plan that exists in the observation sequence to itself, otherwise it is mapped to s_p (which preserves monotonicity).

$$\forall s \in \{s_a, s_h, s_i, s_j, s_k, s_l\}, \quad M(s) = s$$
$$\forall s \in \{s_m, s_n, s_o, s_{g_2}\}, \quad M(s) = s_p$$

Thus, we get that the mapping cost from O to P_2 is $2/7$ and the mapping cost from P_2 to O is $10/10$ (because the distance between p and o, for example, is 3 – one need to reach m, n, o in order to reach o, since l was already reached or alternatively reach q, n, o and $|P_2| = 10$), and together $APC(g_2 \mid O) = 2/7 + 10/10 = 1.29$.

For this method, we need $|G|$ offline calls to the planner to compute the optimal plan for each goal. Also we can build the D_{base} metric offline. The online component of this approach is to find the mapping between the observation sequence and the optimal plan, which takes $\mathcal{O}(|O| \times |P_i|)$ for each goal $g_i \in G$ (Fig. 3).

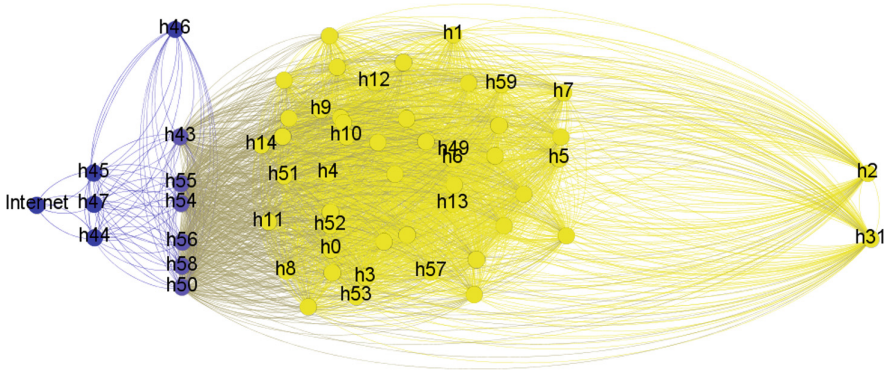

Fig. 3. An illustration of the network used in this work

6 Empirical Evaluation

In this section we provide an empirical comparison between the different goal recognition algorithms. The specific attack graph we use in our evaluations contains 60 hosts. The network architecture as well as the operating systems and software ran by each host was aggregated directly from a real network, that of a large research university.

The vulnerabilities data was randomly generated such that each host has 15 possible vulnerabilities. The initial state in the network is labeled "Internet" and is the entry to all connections in the system. We fixed nodes h_2 and h_{31} as two possible goals for the attacker. The theory was encoded in PDDL as described in Sect. 3.2.

All of the algorithms were implemented in Python, calling the FF planner [17] when required. The algorithms are R&G, M&S, R&G+SC (R&G+ with our varied β), Plan Edit Distance (denoted PED). For the Alternative Plan Cost (denoted APC) algorithm, we used the landmark-based distance suggested by Hoffmann et al. [19].

We simulated attacks targeting host #31 using optimal plans to the goals generated by the FF planner [17]. First, we deleted actions from these plans. Then, we added noisy data to simulate false positive alerts by adding some actions that are not part of the plan.

6.1 Prediction Quality

First, we examined whether the goal with the highest probability predicted is indeed the actual goal. Figure 4 shows the probability distribution on our two goals. The left blue bar is the probability predicted for the real goal (#31) while the right orange bar is for the other goal (#2). As seen in this figure, all of the algorithms gave goal #31 the highest probability, meaning that they managed to capture the true goal. APC gave the highest probability, while PED was close behind it.

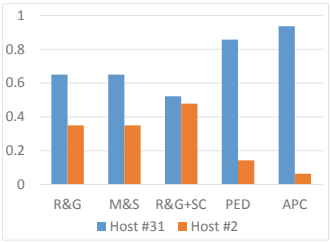

Fig. 4. Probability distribution for multiple goals

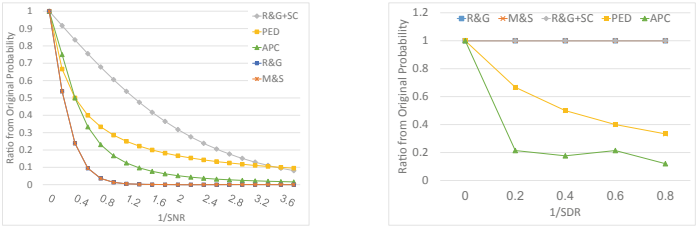

Fig. 5. Probability decline with noisy observations on left (False Positives) and missing observations on right (False Negatives)

The planning-based algorithms gave the true goal the same score (before normalization), but R&G+SC gave the other goal a higher score, thus after the normalization, the true goal's probability declined. This highlights the drawback in giving more influence to the β parameter, which has a dampening effect – while it is beneficial to reduce the decline in probability given noisy observations, it also give high probability to less likely plans.

6.2 Noisy Observations

Next, we looked at different false information that can interrupt the recognition process. When trying to perform an intrusion, an attacker is likely to try and obfuscate the attack. These efforts can lead to False Positive alerts.

Figure 5 (left) shows the probability decline for all recognition algorithms as more observations are noisy. The x-axis shows the Signal to Noise Ratio (SNR) which represents the number of false observations that were injected to the original plan in comparison to the length of the original plan, and the y-axis represents the relative decline in probability as more noise is introduced. As seen from this figure, R&G and M&S give the exact same values and are the less robust to noise and the probability declines fast when introducing noisy observations. R&G+SC is the best to deal with noise due to the plan-length normalization of the cost. PED and APC both perform better than R&G and M&S. Thus, all our new three algorithms outperformed the previous ones along this measure.

Table 2. Runtime comparison

	Offline	Online
R&G	0	1.7237
M&S	0.8210	0.8345
R&G+SC	0.6451	0.6578
PED	0.7251	0.0002
APC	686.7375	0.3246

6.3 Missing Observations

The second property examined is the algorithms' sensitivity to actions that are missing from the observations. This can be either because some of the actions of the attacker were not recognized or because the attacker managed to hide its actions from the alert system, or due to the fact that the attack is still ongoing and not all of the attack steps have yet been executed. Figure 5 (right) shows the probability decline for all recognition algorithms as more actions are missing. The x-axis is Signal Drop Ratio (SDR), representing relative proportion of the number of actions that were removed from the original plan, and the y-axis represents the relative decline (the original probability was normalized to 1) in probability of the goal as more data is missing. The planning-based algorithms give the same results and are more robust to missing observations, as they complete the needed missing actions anyway as part of their solution.

By contrast, both PED and APC reason about the proportion of the plan that was executed. This results in a decline in the probability returned by these algorithms.

6.4 Running Times

Finally, a desired attribute of a goal recognizer in an intrusion detection system is the ability to make a recognition in real-time. Table 2 shows the online and offline runtimes (measured in seconds) of each algorithm. They reflect the differences in computation between the algorithms: R&G and R&G+SC require to run the planner once more per goal, to compute an alternative plan given the observations. PED is the lightest and requires to compute the difference between the actions in the optimal plan and the observation sequence. APC requires to find the landmarks achieved in the observation sequence (a traversal over the list of landmarks) and then to compute the edit distance of two sequences of predicates. This makes it the slowest method during offline planning. PED is the fastest and APC is about an order of magnitude faster than R&G and SC, as it does not require to run a planner during recognition. Thus, both new distance-based algorithms are much faster but have an internal tradeoff. PED is faster but is less robust to noise and its prediction quality (See Fig. 4). Each algorithm has also a different cost in its preprocessing: all algorithms require to find an optimal

plan for each goal. Additionally, APC requires to extract the set of landmarks for each goal, a process that takes more than an order of magnitude longer than the other algorithms, but is performed offline.

7 Summary, Discussion and Conclusions

This paper presented goal recognition algorithms on attack graphs for intrusion detection. It utilizes state-of-the-art goal recognition algorithms using planning, proposed a new algorithm to this family that is more adjusted to the challenges of attack graphs, and then proposed a new approach for goal recognition and offers two algorithms to tackle specific challenges that are interesting in the context of an intrusion detection system. All of the algorithms are evaluated on a real-world network and simulated attacks.

The empirical results show that all algorithms manage to deal with missing observations and false positives, while having the highest probability always assigned to the correct goal. However, there is a clear tradeoff between the algorithms in terms of robustness to noise and time.

The first group of presented algorithms (R&G, M&S and R&G+SC) handle missing observations by extrapolating the observation sequence. This means that they are robust to missing observations, but this comes at the cost of runtime. The PED and APC algorithms do not require to run a planner on the observation sequence, thus in real-time the running costs are smaller. The latter does require the extraction of all possible landmarks, but it can be performed offline. They are even better than the previous planning-based algorithms in their online runtime and the prediction quality.

One limitation of the study is that it assumes a deterministic planner that returns a single optimal plan. In many cases k (optimal) plans exist. To find k-optimal or k-best plans one needs to execute a planner k times. This becomes costly in terms of time [42], which is less desirable for real-time intrusion detection. Furthermore, finding k plans may not be cost-effective in our case as k-best plans to the same goal have many overlaps and are very similar, as opposed to plans to different goals which have much fewer overlaps if any [26].

An interesting research direction is to find new algorithms for real-time goal recognition that take advantage of the network structure given by the attack graph. The authors believe that this structure can be used to enhance the distance metric between states, and later be used with the alternative plan cost calculation presented in Felner et al. [11].

References

1. Al-Mamory, S., Zhang, H.: A survey on IDS alerts processing techniques. In: The 6th WSEAS International Conference on Information Security and Privacy (2007)
2. Ang, S., Chan, H., Jiang, A.X., Yeoh, W.: Game-theoretic goal recognition models with applications to security domains. In: Rass, S., An, B., Kiekintveld, C., Fang, F., Schauer, S. (eds.) Decision and Game Theory for Security. LNCS, vol. 10575, pp. 256–272. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68711-7_14

3. Avrahami-Zilberbrand, D., Kaminka, G.: Fast and complete symbolic plan recognition. In: International Joint Conference on Artificial Intelligence (2005)
4. Azer, M.A., El-Kassas, S.M., El-Soudani, M.S.: Security in ad hoc networks: from vulnerability to risk management. In: 2009 Third International Conference on Emerging Security Information, Systems and Technologies, SECURWARE 2009, pp. 203–209. IEEE (2009)
5. Backes, M., Hoffmann, J., Künnemann, R., Speicher, P., Steinmetz, M.: Simulated penetration testing and mitigation analysis. arXiv preprint arXiv:1705.05088 (2017)
6. Bisson, F., Kabanza, F., Benaskeur, A.R., Irandoust, H.: Provoking opponents to facilitate the recognition of their intentions. In: AAAI (2011)
7. Bui, H.: A general model for online probabilistic plan recognition. In: International Joint Conference on Artificial Intelligence, vol. 3, pp. 1309–1315 (2003)
8. Chyssler, T., Burschka, S., Semling, M., Lingvall, T., Burbeck, K.: Alarm reduction and correlation in intrusion detection systems. In: DIMVA, pp. 9–24 (2004)
9. Durkota, K., Lisỳ, V., Bosanskỳ, B., Kiekintveld, C.: Optimal network security hardening using attack graph games. In: International Joint Conference on Artificial Intelligence, pp. 526–532 (2015)
10. E-Martin, Y., R-Moreno, M., Smith, D.: A fast goal recognition technique based on interaction estimates. In: Twenty-Fourth International Joint Conference on Artificial Intelligence (2015)
11. Felner, A., Stern, R., Rosenschein, J., Pomeransky, A.: Searching for close alternative plans. AAMAS **14**, 211–237 (2007). https://doi.org/10.1007/s10458-006-9006-1
12. Freedman, R., Zilberstein, S.: Integration of planning with recognition for responsive interaction using classical planners. In: AAAI, pp. 4581–4588 (2017)
13. Geib, C., Goldman, R.: Plan recognition in intrusion detection systems. In: 2001 Proceedings of the DARPA Information Survivability Conference and Exposition II, DISCEX 2001, vol. 1, pp. 46–55. IEEE (2001)
14. Geib, C., Maraist, J., Goldman, R.: A new probabilistic plan recognition algorithm based on string rewriting. In: ICAPS, pp. 91–98 (2008)
15. Goldman, R., Friedman, S., Rye, J.: Plan recognition for network analysis: preliminary report. In: AAAI Workshops on PAIR (2018)
16. Gonda, T., Shani, G., Puzis, R., Shapira, B.: Ranking vulnerability fixes using planning graph analysis. In: IWAISe: First International Workshop on Artificial Intelligence in Security, p. 41 (2017)
17. Hoffmann, J.: FF: the fast-forward planning system. AI Mag. **22**(3), 57 (2001)
18. Hoffmann, J.: Simulated penetration testing: from "Dijkstra" to "Turing Test++". In: ICAPS, pp. 364–372 (2015)
19. Hoffmann, J., Porteous, J., Sebastia, L.: Ordered landmarks in planning. J. Artif. Intell. Res. **22**, 215–278 (2004)
20. Kabanza, F., Filion, J., Benaskeur, A.R., Irandoust, H.: Controlling the hypothesis space in probabilistic plan recognition. In: International Joint Conference on Artificial Intelligence, pp. 2306–2312 (2013)
21. Le Guillarme, N., Mouaddib, A., Gatepaille, S., Bellenger, A.: Adversarial intention recognition as inverse game-theoretic planning for threat assessment. In: ICTAI, pp. 698–705. IEEE (2016)
22. Lisỳ, V., Píbil, R., Stiborek, J., Bošanskỳ, B., Pěchouček, M.: Game-theoretic approach to adversarial plan recognition. In: ECAI, pp. 546–551. IOS Press (2012)
23. Masters, P., Sardina, S.: Cost-based goal recognition for path-planning. In: AAMAS, pp. 750–758 (2017)

24. Masters, P., Sardina, S.: Deceptive path-planning. In: International Joint Conference on Artificial Intelligence 2017, pp. 4368–4375. AAAI Press (2017)
25. Mirsky, R., Gal, Y., Tolpin, D.: Session analysis using plan recognition. In: Workshop on User Interfaces and Scheduling and Planning (UISP) (2017)
26. Mirsky, R., Stern, R., Gal, Y., Kalech, M.: Plan recognition design. In: AAAI, pp. 4971–4972 (2017)
27. Noel, S., Jajodia, S.: Managing attack graph complexity through visual hierarchical aggregation. In: Workshop on Visualization and Data Mining for Computer Security, pp. 109–118. ACM (2004)
28. Noel, S., Robertson, E., Jajodia, S.: Correlating intrusion events and building attack scenarios through attack graph distances. In: Computer Security Applications Conference (2004)
29. Noel, S., Jajodia, S.: Optimal IDS sensor placement and alert prioritization using attack graphs. J. Netw. Syst. Manag. **16**(3), 259–275 (2008)
30. Ou, X., Govindavajhala, S.: MulVAL: a logic-based network security analyzer. In: 14th USENIX Security Symposium. Citeseer (2005)
31. Pereira, R., Oren, N., Meneguzzi, F.: Landmark-based heuristics for goal recognition. In: AAAI (2017)
32. Pereira, R., Oren, N., Meneguzzi, F.: Plan optimality monitoring using landmarks and planning heuristics. In: PAIR Workshop in AAAI (2017)
33. Poolsappasit, N., Dewri, R., Ray, I.: Dynamic security risk management using Bayesian attack graphs. IEEE Trans. Dependable Secur. Comput. **9**, 61–74 (2012)
34. Qin, X., Lee, W.: Attack plan recognition and prediction using causal networks. In: 2004 20th Annual Computer Security Applications Conference, pp. 370–379. IEEE (2004)
35. Ramírez, M., Geffner, H.: Plan recognition as planning. In: AAAI (2009)
36. Ramírez, M., Geffner, H.: Probabilistic plan recognition using off-the-shelf classical planners. In: AAAI (2010)
37. Roschke, S., Cheng, F., Meinel, C.: A new alert correlation algorithm based on attack graph. In: Herrero, Á., Corchado, E. (eds.) CISIS 2011. LNCS, vol. 6694, pp. 58–67. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21323-6_8
38. Shmaryahu, D.: Constructing plan trees for simulated penetration testing. In: ICAPS (2016)
39. Shmaryahu, D., Shani, G., Hoffmann, J., Steinmetz, M.: Partially observable contingent planning for penetration testing. In: IWAISe: First International Workshop on Artificial Intelligence in Security, p. 33 (2017)
40. Shmaryahu, D., Shani, G., Hoffmann, J., Steinmetz, M.: Simulated penetration testing as contingent planning. In: ICAPS (2018)
41. Shvo, M., Sohrabi, S., Mcllraith, S.: An AI planning-based approach to the multi-agent plan recognition problem. In: PAIR Workshop in AAAI (2017)
42. Sohrabi, S., Riabov, A., Udrea, O.: Plan recognition as planning revisited. In: International Joint Conference on Artificial Intelligence, pp. 3258–3264 (2016)
43. Swiler, L., Phillips, C., Ellis, D., Chakerian, S.: Computer-attack graph generation tool. In: DISCEX, p. 1307. IEEE (2001)
44. Vered, M., Kaminka, G.: Heuristic online goal recognition in continuous domains. In: International Joint Conference on Artificial Intelligence, pp. 4447–4454 (2017)

45. Vered, M., Pereira, R., Magnaguagno, M., Kaminka, G., Meneguzzi, F.: Towards online goal recognition combining goal mirroring and landmarks. In: AAMAS (2018)
46. Wang, L., Liu, A., Jajodia, S.: Using attack graphs for correlating, hypothesizing, and predicting intrusion alerts. Comput. Commun. **29**(15), 2917–2933 (2006)
47. Zhang, S., Li, J., Chen, X., Fan, L.: Building network attack graph for alert causal correlation. Comput. Secur. **27**(5–6), 188–196 (2008)