Adversarial Planning and Plan Recognition: Two Sides of the Same Coin

Sviatoslav Braynov
Department of Computer Science
University of Illinois at Springfield
Springfield, IL 62703
Email: sbray2@uis.edu

Abstract—Effective adversarial plan recognition requires information about how the adversary is planning his actions and vice versa, the way the adversary is planning his actions is affected by how those actions are going to be detected.

In this paper, we develop a game-theoretic model that integrates adversarial planning with adversarial plan recognition. The model considers planning and plan recognition to be in Nash equilibrium. The papers also show how plan recognition could be manipulated to the advantage of the adversary in the presence of incomplete information.

I. INTRODUCTION

Adversarial plan recognition is important for predicting intentions and future actions of attackers, recognizing unknown attacks, and planning appropriate responses. Because of its ability to detect attacks in the early stages of preparation, adversarial plan recognition plays a constantly increasing role in systems for early detection and prevention of attacks against critical infrastructures. It is also important for robotic warfare where adversarial robots can automatically plan and launch attacks that are too fast for the human decision cycle.

In general, plan recognition deals with inferring an agent’s goals and plans based on his observed actions. Goal recognition is a special case of plan recognition where only the goal is recognized. Plan recognition has been traditionally divided into two types: keyhole and intended plan recognition [2]. In keyhole plan recognition, the planning agent is indifferent to the fact that his plans are being observed and interpreted. The presence of a recognizer who is watching the activity of the planning agent does not affect the way he plans and acts.

In intended plan recognition, the planning agent is aware that he has been watched and acts cooperatively by choosing actions and plans that are easy to detect and recognize. In a sense, the planning agent and the recognizer act together.

Adversarial plan recognition was first suggested by Geib and Goldman [3] as an addition to the traditional models of keyhole and intended recognition. It has been also independently proposed by Jensen et al. [8] for predicting the opponent’s moves in robotic games. In adversarial recognition, the observed agent is hostile to the observation of his actions and attempts to thwart the recognition.

Although there has been significant recent work in adversarial plan recognition [1], [3], [4], [15], [5], [6], little thought has been given to the question of how to plan attacks in order to avoid detection. Instead, current research on plan recognition assumes that hostile agents are using planning methods similar to those used by cooperative agents. For example, by hostile agents Geib and Goldman [5] mean agents that hide some of their actions in order to conceal the traces of an attack. Such an approach creates several problems because adversarial planning is usually much more complex than hiding actions. For instance, an adversarial agent could take actions to manipulate or confuse plan recognition, or to exploit some of its weaknesses.

By neglecting adversarial planning, current research on adversarial plan recognition runs into another problem: plan recognition is generally perceived as the inverse of planning. Effective adversarial plan recognition requires information about how the adversary is planning his actions and vice versa, the way the adversary is planning his actions is affected by how those actions are going to be detected. In other words, planning and plan recognition complement each other and need to be considered together. Effective adversarial planning requires knowledge of plan recognition and effective plan recognition requires knowledge of adversarial planning.

In many applications, such as computer security and information warfare, the purpose of adversarial plan recognition is to predict possible attacks in order to generate effective counterattacks. In such applications, one needs to know not only how his plans are going to be recognized by the adversary, but also how to recognize adversary plans in order to counterplan.

In this paper, we develop a game-theoretic model that integrates adversarial planning with adversarial plan recognition. The model considers planning and plan recognition to be in Nash equilibrium. Two models are in Nash equilibrium if each model is the best response to the other. In other words, the attack planning model generates the most effective attack against the plan recognition model, and the plan recognition model maximizes the effectiveness of detecting the attack.

The paper is organized as follows. Section 2 introduces a game-theoretic model of adversarial planning and adversarial plan recognition. Section 3 describes how plan recognition could be manipulated by an adversary. Section 4 solves a planning game. Finally, Section 5 provides a general discussion.
II. GAME-THEORETIC MODEL OF ADVERSARIAL PLANNING AND ADVERSARIAL PLAN RECOGNITION

We first begin by formalizing adversarial planning and adversarial plan recognition. We propose a rather simple but general model suitable for different adversarial domains and planning/recognition problems. We assume that there are two players that pursue opposite goals. The attacker chooses one target from a set of possible targets and attempts to reach it, whereas the defender must determine which target has been chosen and defend it. We assume that the target is known only to the attacker. The defender knows the set of possible targets without being certain which target has been chosen by the attacker. In addition, the defender is limited to execute only preemptive actions, i.e., actions that can prevent attacks only when executed in advance.

The notion of attack graph captures the intuition about all possible attacks that can be launched against a set of targets.

Definition 1. An attack graph is a tuple \( AG = (S, A_A, T, s_0, T) \) where:

- \( S \) is a finite set of states,
- \( A_A(s) \) is the finite sets of actions available to the attacker in state \( s \in S \),
- \( T \subseteq S \times A_A \times S \) is the transition relation,
- \( s_0 \in S \) is the initial state,
- \( T \subseteq S \) is the set of possible targets.

Intuitively, the attacker takes actions that move the system from one state to another according to the transition relation \( T \). The attacker starts in the initial state, \( s_0 \), and attempts to reach one of the target states. By definition, the attack stops when it reaches a target state, i.e., \( A_A(s) = \emptyset \) for every target state \( s_t, s \in T \). An example of an attack graph with two target states labeled 3 and 6 is shown in Figure 1.

![Fig. 1. An attack graph](image)

Although Definition 1 is similar to the definition of attack graphs used in computer security [9], [14], [17], our model can be applied to a wide range of applications, including military operations, surveillance and intelligence gathering, economic applications, etc. For example, the attacker could be trying to penetrate a market that the defender currently dominates.

In this paper, we focus on planning domains with deterministic actions. In addition, we assume that all actions are observable and there is no uncertainty about the current state of the system.

The attacker chooses a target from the attack graph, which is known only to him. The defender knows the attack graph without being certain which target has been chosen. The attacker’s objective is to find a sequence of actions (a path in the attack graph) that leads to the target and could withstand the defender’s counteractions.

Definition 2. A deterministic plan, \( p_A \), of the attacker is a finite set of state-action pairs:

\[
p_A = \{(s, a) | a \in A_A(s), s \in S\}.
\]

The set of all attacker’s plans is denoted by \( P_A \).

The set of defender’s plans is denoted by \( P_D \) and is defined similarly.

According to Definition 2, a deterministic attack plan is a mapping from the current state of the system to an attack action. An example of an attack plan for the graph shown in Figure 1 is: \( p_A = \{ (1, a_1), (2, a_2) \} \). In the plan, the attacker executes action \( a_1 \) in the initial state, and then executes \( a_2 \) in state 2. The target is state 3.

Definition 3. A probabilistic attack plan is a probabilistic mixture of deterministic plans:

\[
[p_1 : q_1], [p_1 : q_1],..., [p_n : q_n]
\]

where deterministic plan \( p_i \) is executed with probability \( q_i \).

The concept of execution trace captures the intuition that when executed, an attack plan can take different routes in the attack graph depending on the defender’s counteractions.

Definition 4. The attacker’s execution trace is a finite ordered sequence of attacker’s actions \( a_0, a_1, a_2, ..., \) such that:

- it starts in the initial state, i.e., \( a_0 \in A_A(s_0) \),
- for every action \( a_i \) in the sequence, either \( a_i \) is the last action in the sequence, or there exists states \( s_i \) and \( s_{i+1} \), such that \( T(s_i, a_i, s_{i+1}) \).

An execution trace is complete if the last action in the trace \( a_n \) moves the system into a target state, i.e., there exist states \( s_n \) and \( s_{n+1} \), such that \( T(s_n, a_n, s_{n+1}) \) and \( s_{n+1} \in T \).

The set of attacker’s execution traces is denoted by \( \Sigma_A \).

The defender’s objective is to protect all targets in the attack graph without knowing which particular target has been chosen by the attacker. To this end, the defender first needs to determine which target is under attack and then thwart the attack using a finite set of preemptive blocking actions, \( A_D \). Each defender’s blocking action permanently blocks an attacker’s action that could eventually happen in the future. One can think of a blocking action as removing an arc from the attack graph. More formally:

Definition 5. The set of actions available to defender, \( A_D \), consists of blocking actions. A blocking action, \( a_D \), applicable to the current state \( s \) consists of:

1) choosing a state \( s' \), such that \( s' \neq s \) and \( s' \) is reachable from the current state \( s \) in the attack graph,
2) choosing an action available to the attacker at \( s' \) and removing the action from the set of actions available to
the attacker at $s'$:

$$A_A(s') = A_A(s') \setminus \{a_A\} \text{ for some action } a_A \in A_A(s')$$

Whereas the second entry in Definition 5 is intuitive and self-explanatory, the first entry needs additional justification. According to the first entry, the defender cannot block actions applicable to the current state. The reason for this is that the defender and the attacker act concurrently in the current state without knowing each other’s actions. In many cases, it would be too late to block an action that has already started or has been executed concurrently. In addition, blocking concurrent actions leads to a non-deterministic attack graph. It could also introduce infinite loops in the graph where the attacker repeatedly takes the same action and the defender always blocks it.

It is important to understand that the attacker and the defender can act concurrently at a given state: the attacker applies an attack action while the defender applies a blocking action. Although applied concurrently with the attack action, the blocking action can only block a future attack action that has not yet been attempted by the attacker.

In the example shown in Figure 1, the defender can block any action except $a_1$ and $a_3$ in the initial state $s_0$. The decision which action to block depends on the expectation of which target will be attacked and which path will be followed. If the defender decides to block action $a_2$ and the attacker executes plan $p = (1, a_1), (2, a_2)$, then the attack will be thwarted and the game will end at state 2.

The interaction between the attacker and the defender can conveniently be modeled as a two-person non-cooperative game played on the attack graph.

**Definition 6.** The non-cooperative game between the attacker and the defender is a tuple $(\text{AG}, \bar{s}, A_D, U_A, U_D, R)$, where:

- $\text{AG}$ is the initial attack graph,
- $\bar{s}, \bar{s} \in T$, is the target chosen by the attacker,
- $A_D$ is a set of blocking actions available to the defender,
- $U_A$ is the utility function of the attacker,
- $U_D$ is the utility function of the defender,
- $R$ is the plan recognition algorithm used by defender.

The target is constant during the game. In other words, the attacker chooses the target before the game starts and keeps it the same throughout the game. The game starts at the initial state and at each state, the attacker chooses an action applicable to the state. After the action has been applied, the system moves to another state according to the attack graph.

At each state, the defender must decide whether to block a future attacker’s action and which action to block. If the defender decides to apply a blocking action, then the defender’s action and the attacker’s actions are applied concurrently, with the players not knowing each other’s actions. The result of a blocking action becomes known to the attacker only at the end of its execution when the system moves to the next state. The effects of all actions are deterministic and fully observable by both players. The game ends when a target state is reached, or when there are no actions applicable to the current state.

It needs to be pointed out that the attack graph might change during the game as a result of applying blocking actions and removing arcs from the graph.

To make the situation more interesting, one might want to impose additional constraints on the defender’s actions. For example, the defender could be allowed to use no more than $k$ blocking actions, where $k$ is some predetermined constant. One could also specify the initial and the final time for applying the defender’s actions. For example, the defender could be allowed to apply actions only between the first and the $n^{th}$ step of the attack, even without knowing the length of the attack.

There are different ways to define player’s utility functions. For example, the attacker’s utility could be binary, taking a value of one only if the target has been reached. In some situations, the attacker could be partially rewarded for getting closer to the target (in surveillance and intelligence gathering scenarios, for example) or for reaching another target. In a slightly modified game, the attacker might assign different priorities to targets. The utility function of the defender could depend on target ranks or on the distance from the target. It could also be a simple binary function, taking a value of one if no target has been reached. In the game, players do not have complete knowledge of each other’s utility functions. For example, the defender does not know which target has been chosen. The attack graph, however, is common knowledge to both players.

In the paper, we constrain our attention to binary utility functions. For example, the attacker’s reward is 1 if he reaches the goal, and 0 otherwise.

In order to protect the targets, the defender needs to ascertain the attacker’s goal and recognize his plans by observing his actions. We assume that the defender uses a plan recognition algorithm to identify the target that has been chosen and to predict the attacker’s future actions.

**Definition 7.** The defender’s plan recognition is a function $R : \Sigma_A \rightarrow \Delta P_A$, where $\Sigma_A$ is the set of the attacker’s execution traces, and $\Delta P_A$ is the set of probability distributions over the set of attacker’s plans.

The plan recognition algorithm is applied to the attacker’s execution trace which is usually incomplete, and produces a probability distribution over possible attacker’s plans. A good plan recognition should always assign a non-zero probability to the actual attacker’s plan. In the ideal case, the plan recognition algorithm could produce a deterministic prediction by assigning a probability of one to the actual attacker’s plan.

The output of the plan recognition algorithm is fed into the input of the defender’s decision algorithm which produces a defense plan.

**Definition 8.** The defender’s decision algorithm is a function $D : \Delta P_A \rightarrow P_D$, where $P_D$ is the set of all defender’s plans.

In other words, the defender’s decision algorithm takes a probabilistic guess about the attacker’s plan and produces a defense plan that is a response to the probabilistic guess.
The defender’s decision algorithm, \( D \), produces the best response if:

\[
D(p_A) = \arg \max_{p_D \in \mathcal{P}_D} U_D(R(p_A), p_D) \quad (1)
\]

It has to be mentioned that the defender’s decision algorithm might not always produce the best response because of complexity problems associated with its computation.

If the defender knew the actual plan of the attacker \( p_A \), the defender would compute \( D(p_A) \) to find a response. Since \( p_A \) is unknown to the defender, he needs to compute his response incrementally using the input from the recognizer algorithm.

Let \( p_D \) denote the actual attacker’s plan. The defender’s plan \( p_D \), a response to \( p_A \), is computed as follows. At moment zero and state zero, the defender decides the first action in \( p_D \) given his expectations about the opponent and the constraints on his actions. At moment one, the defender observes the attacker’s execution trace \( \sigma^1 \), \( \sigma^1 \in \Sigma_A \), which in this case consists of a single action, namely the attacker’s action taken at state zero. The execution trace is fed into the plan recognition and decision algorithms to produce the defender’s plan at moment one:

\[
p_D^1 = D(R(\sigma^1)).
\]

In general, the defender’s plan at moment \( k \) is computed as follows:

\[
p_D^k = D(R(\sigma^k)) \quad (2)
\]

where \( \sigma^k \) is the execution trace at moment \( k \) generated by \( p_A \) and \( p_D^{k-1} \).

Let, \( D(R(p_A)) \) denote the defender’s response to \( p_A \) generated at the end of the game by following the procedure above. We can safely assume that the players’ utility functions are defined on pairs of the type \( (p_A, D(R(p_A))), (p_A, D(R(p_A))) \in \mathcal{P}_A \times \mathcal{P}_D \), because every such pair completely determines the outcome of the game.

Since the attacker knows the defender’s decision and plan recognition algorithms, the attacker can predict the defender’s response to any attack plan. Therefore, the attacker would try to find an attack plan \( p_A^* \) which is a best response to the defender’s plan, i.e., a plan which maximizes the attacker’s utility:

\[
p_A^* = \arg \max_{p_A \in \mathcal{P}_A} U_A(p_A, D(R(p_A))) \quad (3)
\]

The plan \( p_A^* \) is optimal for the attacker, given the defender’s decision algorithm \( D \) and plan recognition algorithm \( R \).

The assumption that the attacker knows the defender’s planning and decision algorithms is realistic and complies with Kerckhoffs’ principle [10] used in cryptography. According to Kerckhoffs’ principle, the security of a system cannot be based on the secrecy of its internal working because sooner or later the adversary gets to know the system.

Suppose that the defender’s decision algorithm \( D \) always computes a best response \( p_D \) according to Equation 1, and the attacker always computes a best response \( p_A \) according to Equation 3. Interestingly, this is not sufficient for the pair \( (p_A, p_D) \) to be Nash equilibrium [12], [13]. The problem is that the defender’s recognition algorithm \( R \) could be susceptible to manipulations by the attacker.

A pair of plans \((p_A, p_D)\) is Nash equilibrium if each plan is a best response to the other. Formally, \((p_A, p_D)\) is Nash equilibrium iff: for every \( p_A^* \in \mathcal{P}_A \):

\[
U_A(p_A, p_D) \geq U_A(p_A^*, p_D)
\]

and for every \( p_D^* \in \mathcal{P}_D \):

\[
U_D(p_A, p_D^*) \geq U_D(p_A, p_D)
\]

A Nash equilibrium would be chosen by rational players since a player would never do better by unilaterally deviating from the equilibrium. In addition, the equilibrium is safe for both players even if they know each other’s plans.

### III. Manipulation of Plan Recognition

The problem with the defender’s recognition algorithm is that the recognition algorithm might fail to produce a good guess about the attacker’s plan, even if the decision algorithm is correct. Because the decision algorithm receives its input from the recognition algorithm, the defender’s plan produced by both algorithms might be inferior, i.e., it might not offer the best protection against the attacker’s plan. Knowing this, the attacker could choose a plan that will make the defender act in the attacker’s advantage. For example, the attacker could make the defender erroneously believe that the attacker is going to act in a particular way, while in fact he is not.

A plan recognition algorithm is accurate if every probabilistic guess about an attack produces the same defense decision as the original attack:

\[
D(p_A) = D(R(p_A)). \quad (4)
\]

In other words, it does not make difference for the defender whether he knows the actual attack plan or uses the probabilistic guess produced by the plan recognition algorithm.

A possibility for manipulation would arise if Equation 4 does not always hold. This, however, might not be sufficient to carry out the manipulation. It is also necessary that the attacker has a plan that invalidates Equation 4 to his advantage. In other words, a recognition algorithm might be manipulable by one attacker and not manipulable by another attacker.

**Definition 9.** The recognizer \( R \) is manipulable if there exists an attacker’s plan, \( p_A \in \mathcal{P}_A \) such that:

\[
U_D(p_A, D(p_A)) > U_D(p_A, D(R(p_A)))
\]

\[
U_A(p_A, D(p_A)) < U_A(p_A, D(R(p_A)))
\]

In Definition 9, \( U_D(p_A, D(p_A)) \) is the defender’s utility if the defender knows the actual attack plan, whereas \( U_D(p_A, D(R(p_A))) \) is the defender’s utility based on the guess produced by the recognition algorithm. According to the definition, the defender regrets acting on the guess produced by the recognition algorithm. If he knew the actual attack plan \( p_A \), he would act differently and choose \( D(p_A) \) instead of choosing \( D(R(p_A)) \). The second inequality in Definition 8
ensures that it is to the advantage of the attacker to manipulate the defender.

As an example of manipulation, consider again the attack graph shown in Figure 1. Assume that the defender is constrained to use only one blocking action for the duration of the game. Suppose that the recognition algorithm is based on the assumption that the attacker always chooses the shortest path to the target. Therefore, if the attacker takes action \( a_3 \) at state 1, the recognition algorithm will predict that the goal is state 6, assuming that if the goal was state 3, the attacker would take action \( a_1 \) at state 1. As a result, the recognition algorithm will predict the plan \( \langle 4, a_4 \rangle, \langle 5, a_6 \rangle \). Based on the prediction, the decision algorithm will suggest blocking action \( a_6 \) at state 4.

This recognition algorithm is manipulable because it is to the advantage of the attacker to deviate from the prediction. For example, the attacker can choose plan \( \langle 1, a_3 \rangle, \langle 4, a_4 \rangle, \langle 5, a_5 \rangle \) for reaching target state 3, thereby fooling the recognition algorithm into believing that he is moving towards state 6. Upon observing action \( a_3 \) in state 1, a non-manipulable algorithm would suggest some probability distribution over possible plans. For example, it could suggest plan \( \langle 4, a_4 \rangle, \langle 5, a_5 \rangle \) with probability \( \frac{1}{2} \) and plan \( \langle 4, a_4 \rangle, \langle 5, a_6 \rangle \) with probability \( \frac{1}{2} \). Obviously, a recognition algorithm is not manipulable if its predictions are self-fulfilling and it is beneficial for the attacker to follow every prediction.

The next proposition follows directly from the definitions above.

**Proposition 1.** A pair \( (p_A, p_D) \) is Nash equilibrium if the defender’s decision algorithm always produces the best response, the recognition algorithm is not manipulable, and the attacker optimizes according to Equation 3.

The interesting part about Proposition 1 is that the Nash equilibrium does not require complete accuracy of the recognition algorithm. This agrees with the intuition that a less accurate recognition algorithm could offer adequate protection against less skilled attackers, whereas more accuracy is necessary when dealing with skillful attackers.

### IV. Planning without Plan Recognition

In many situations, the defender could be constrained and might not have the opportunity to observe and act upon the attacker’s actions. Consider the game defined on the attack graph shown in Figure 1, assuming that the defender is constrained to use only one blocking action for the duration of the game and the players’ utilities are binary. In this game, the defender can apply four blocking actions: he can block \( a_2, a_4, a_5 \) or \( a_6 \). Blocking \( a_4 \) is better alternative than blocking \( a_5 \) or \( a_6 \) because after blocking \( a_4 \) neither \( a_5 \) nor \( a_6 \) is applicable. This reduces the choice of action to \( a_2 \) and \( a_4 \), thereby leaving no room for applying the recognition algorithm. Because \( a_2 \) and \( a_4 \) must be blocked at the initial state, the defender is forced to decide upfront which actions to block, \( a_2 \) or \( a_4 \), before seeing any action of the attacker.

We are looking for a Bayesian Nash equilibrium of the game [7] because the defender has incomplete information about the target chosen by the attacker. Assume that the defender assigns probability \( \alpha \) to target state 3 and probability \( 1 - \alpha \) to target state 6. This is equivalent to a situation in which the defender plays the game against either of two attackers. The first attacker always selects target state 3, while the second attacker always selects target state 6. The probability of facing the first attacker is \( \alpha \) and the probability of facing the second attacker is \( 1 - \alpha \).

The equilibrium we are looking for is the following. If the target is state 6, then there is only one possible plan for reaching it: \( \langle 1, a_3 \rangle, \langle 4, a_4 \rangle, \langle 5, a_6 \rangle \). If the target is state 3, then the attacker follows a probabilistic plan in which he chooses plan \( p_1 = \langle 1, a_1 \rangle, \langle 2, a_2 \rangle \) with probability \( q \) and plan \( p_2 = \langle 1, a_3 \rangle, \langle 4, a_4 \rangle, \langle 5, a_5 \rangle \) with probability \( 1 - q \). The defender follows a probabilistic plan in which he randomizes between blocking \( a_2 \) with probability \( r \) and blocking \( a_4 \) with probability \( 1 - r \).

The attacker will randomize between plans \( p_1 \) and \( p_2 \) only if they are equally attractive to him. This implies that \( r = \frac{1}{2} \), i.e., the defender blocks each plan with probability \( \frac{1}{2} \).

In order for the defender to randomize between blocking \( a_2 \) and blocking \( a_4 \), both blocking actions must have the same expected utility. Therefore:

\[
q = \begin{cases} 
    1 & \text{if } \alpha \geq \frac{1}{2}, \\
    \frac{1}{2\alpha} & \text{if } \alpha < \frac{1}{2}.
\end{cases}
\]

Given this choice of attack and counterattack plans, the probability of thwarting the attack is \( \alpha \) if \( \alpha \geq \frac{1}{2} \), and \( \frac{1}{2} \) if \( \alpha < \frac{1}{2} \).

### V. Discussion

In the ideal case, the defender could remove all incoming arcs to target states, thereby preventing all possible attacks. Such an approach could be wasteful and will not work if the duration of the attack is shorter than the number of the arcs to be removed. In general, the defender would try to determine the minimal set of blocking actions that will prevent the attacker from reaching a target from a given state. This corresponds to finding the minimum edge-cut in the subgraph induced by the current and the target states. One of the fastest deterministic algorithms for finding a minimum cut in an unweighted graph requires \( O(nm) \) steps [11]. Sheyner et al., [16], [9] have also proposed an algorithm for computing the minimal critical set of actions whose removal will prevent an attack.

Our approach differs from previous analysis of attack graphs in that the defender does not need to block the whole critical set in advance. This could be too costly and time consuming especially for large attack graphs. In our approach, the defender watches how the attack develops and blocks actions only when needed based on the prediction produced by the recognition algorithm. This is more effective because the defender needs to work only on a part of the attack graph suggested by the recognition algorithm, which is usually
smaller than the current portion of the attack graph (the portion that starts at the current state). In addition, when choosing blocking actions, the defender could take into account the probability of the attacker pursuing a particular attack path.

Another problem for the defender is to come up with a prior probability distribution over the set of possible targets. In the absence of any other information, the defender could use the structure of the current portion of the attack graph to determine his priors. For example, one estimate of the probability that state \( s_i \) is the target when state \( s \) has been reached is

\[
\frac{N(s, s_i)}{T(s)}
\]

where \( N(s, s_i) \) is the number of paths from the current state \( s \) to the target state \( s_i \), and \( T(s) \) is the total number of target paths from \( s \), i.e., paths between \( s \) and any target. We can use entropy as a measure of how predictive the attack graph is about possible targets:

\[
E(s) = -\sum_{s_i \in \mathcal{T}} \frac{N(s, s_i)}{T(s)} \log \frac{N(s, s_i)}{T(s)} \quad (5)
\]

The entropy is 0 if all attack paths from the current state \( s \) lead to a single target. Figure 2 shows an attack graph and the entropies of each non-target state.

If the defender derives his priors from the attack graph shown in Figure 2, then state \( s_3 \) is the target with probability \( \frac{2}{3} \) and state \( s_6 \) is the target with probability \( \frac{1}{3} \). The previous discussion in Section 4 implies that in this case, the attacker will choose plan \( p_1 = \{a_1, a_2\} \) with probability \( \frac{2}{3} \) and plan \( p_2 = \{a_3, a_4, a_5\} \) with probability \( \frac{1}{3} \) when the target is state \( s_3 \).

It has to be pointed out that deriving prior probabilities from the attack graph is susceptible to manipulation by the attacker. For example, by taking specific actions the attacker can reduce the attack graph to a portion of it which could induce erroneous beliefs.

**VI. Conclusion**

In the paper, we studied a general non-cooperative game defined on an attack graph. To solve the game, we developed a game-theoretic model that integrates adversarial planning with adversarial plan recognition. We showed that plan recognition affects the way the adversary is planning and vice versa, the way the adversary is planning could affect plan recognition. In particular, we showed how plan recognition could be manipulated to the advantage of the adversary in the presence of incomplete information.

The model presented in the paper is general and could be applied to a wide range of applications, including computer security, military operations, surveillance and intelligence gathering, economic analysis of competition, etc.

**References**


