ECAI 2012
Luc De Raedt et al. (Eds.)

doi:10.3233/978-1-61499-098-7-546

Game-theoretic Approach to Adversarial Plan Recognition

Viliam Lisý and **Radek Píbil** and **Jan Stiborek** and **Branislav Bošanský** and **Michal Pěchouček** [1]

Abstract. We argue that the problem of adversarial plan recognition, where the observed agent actively tries to avoid detection, should be modeled in the game theoretic framework. We define the problem as an imperfect-information extensive-form game between the observer and the observed agent. We propose a novel algorithm that approximates the optimal solution in the game using Monte-Carlo sampling. The experimental evaluation is performed on a synthetic domain inspired by a network security problem. The proposed method produces significantly better results than several simple baselines on a practically large domain.

1 Introduction

Intention and plan recognition is an important capability of an intelligent agent. In cooperative settings, it enables coordination with other agents in situations where explicit communication is not possible or desirable. The examples of such situations are unobtrusive intelligent assistance to people or coordination of robotic systems. In competitive or adversarial settings, plan recognition allows, e.g., detecting malicious behavior, or predicting the opponent's future actions. Most of the research on plan recognition assumes that the observed agent is either indifferent about the recognition process, or actively cooperates and provides hints that make the recognition process easier. In many (security) applications the situation is exactly opposite. An agent is required to recognize plans or intentions of its adversaries that are aware of the presence of the recognition process and actively try to avoid it while pursuing their own goals. The adversary may choose actions that are hard to detect, or use deception to mislead the observer. The problem of recognizing plans in such settings is termed *adversarial plan recognition*. It is present, for example, in warfare scenarios, different security domains, such as airport security, or computer networks. The latter is the main scenario we use for experimental validation of the proposed techniques.

The previous studies on adversarial plan recognition identified the problem to be different from other forms of plan recognition (i.e., keyhole and intended plan recognition). They list the differences from the domain perspective and identify the possibility of missing observations to be the most crucial one [7, 10]. However, they perform the recognition as if the observations were missing by accident. They do not model that the observed agent intentionally selects actions to avoid detection and how it influences the recognition process. In this paper, we focus on exactly this aspect of adversarial plan recognition under the assumption of rationality of both agents. Further, we complete the model by allowing the observer to perform actions that influence its chances to observe specific actions of the observed agent.

We define the problem of the adversarial plan recognition as a two-player zero-sum extensive-form game (EFG) between the observer and the observed agent (further called the *actor* for clarity). None of the players can directly observe the actions taken by the other player; however, we assume that the observer receives noisy probabilistic observations of the actions of the actor from the environment. The solution of this game (in the form of Nash equilibrium) provides the optimal strategy for both players. It allows the actor to find the best tradeoff between its plan cost and observability and it prescribes the observer how to choose its actions to maximize the chance of observing and identifying the plan.

In order to achieve better scalability of the approach, we show that thanks to the special structure of the game, the size of the game representation can be substantially reduced. Using the reduced representation, we introduce an algorithm for solving the adversarial plan recognition game. It approximates the optimal solution under real-world time constraints, based on the Monte-Carlo Tree Search (MCTS). Finally, we evaluate the approach experimentally, showing it performs significantly better than a set of baseline algorithms.

2 Related Work

The most crucial difference between the adversarial plan recognition and the other kinds of plan recognition is that the actor actively tries to prevent the recognition process. The previous work on adversarial plan recognition [7, 10] focused only on the effect of this intention – the missing observations, but did not model the intention itself and its effect to the adversary's plan selection. Reasoning of the actor was not part of their models. In this paper, we model the problem more completely, including the intention of both players under the assumption of their rationality.

The concept of rationality is not new in the field of plan recognition. It was first introduced by Mao and Gratch [9] in the problem of utility-based plan recognition. They assume that the actor is rational and tries to maximize its utility. If more plan hypotheses are consistent with the current observations, they propose to use the one with highest expected utility for the actor as the most likely one. This paper, however, uses the assumption of rationality only partially. They assume all plans are equally probable a priory, which is inconsistent with the assumption of rationality. In fact, only the plans that lead to the maximal expected utility can be selected by a truly rational agent.

The problem of utility-based plan recognition was studied also in [3]. In this paper, the authors assume rationality of the observer. It prefers to detect the less likely, but more dangerous behaviors. After computing the probabilities of the hypothesis, they are multiplied

[1] Agent Technology Center, Dept. of Computer Science and Engeneering, FEE, Czech Technical University, Czech Republic, email: {lisy, pibil, stiborek, bosansky, pechoucek}@agents.felk.cvut.cz

by their utility (harm to the observer) and the one with the highest expected utility is the result of the recognition process.

In case of adversarial plan recognition between rational agents, there is often only a small difference between considering the utility of the observer and the utility of the actor. The plans that are more important to be detected for the observer are those that the actor wants to keep hidden. Hence, we model the problem as a zero-sum game. A similar approach was taken by Braynov in [4]. He presents a conceptual framework of planning and plan recognition as a deterministic full-information simultaneous-moves game and argues that rational players would play the Nash equilibrium in the game. The goal of the actor is to traverse an attack graph from a source to one of few targets and the observer can remove one of the edges in the attack graph per move. The approach relays on a plan recognition algorithm as a plugin and the paper does not provide any experimental validation.

Example The following example demonstrates the main difference between the existing and the proposed plan recognition approaches. Assume the actor can execute plans A, B, C with expected utilities $9, 9, 7$ for the observed agent. The classical (even adversarial) plan recognition does not take the utilities into account and assumes uniform a priori probability of all plans. If the observations indicate the probabilities of the plans are $0.2, 0.3, 0.5$, the classical plan recognition [7] answers plan C was most likely executed. The utility-based plan recognition [9] multiplies the expected utilities and the posterior probabilities of the plans to get $1.8, 2.7, 3.5$. It answers plan C was most likely executed by a "rational" agent, because it gives the highest product. In this paper, we use the assumption of rationality strictly. We identify that the rational player can never play C, as this choice is suboptimal. The remaining two plans have the same prior probability. Hence we choose B due to observations.

Furthermore, our model defines different utility values for executing the plan based on its successful observation. It allows defining actions of the observer influencing chances of detecting individual actions in the actor's plan and automatic reasoning about the optimal plan and its recognition.

3 Adversarial Plan Recognition Game

The *adversarial plan recognition game* (APRG) is an imperfect-information zero-sum EFG between the actor and the observer.

3.1 Game Definition

APRG is defined as a tuple (A, P, g, D, m), where

- A is a set of actions available to the actor. It can contain the action None, corresponding to actor not performing any action.
- P is the set plans (sequences of actions) of the actor. We assume the actions have preconditions and effects; therefore, not all sequences are legal plans. We often assume the plans are organized as a tree by identical prefixes.
- $g : P \to \mathbb{R}$ is the actor's payoff for the actor in case the plan is performed unobserved.
- D is a set of actions available to the observer. Generally, it corresponds to using a specific sensor / behavior classifier, each with different chances to detect the actor's actions correctly.
- $m : D \to \mathbb{R}^{|A|\times|A|}$ is a collection of one confusion matrix for each observer's action $d \in D$. Such matrix expresses the probability that the observer observes action $o \in A$ in case the actor executes $a \in A$ (i.e., for each $d \in D$ we have $p(o|a, d)$).[2]

[2] Note that the set of observations can be different form the set of actions. However, it would require a more complex definition of the utility function.

The game is played in discrete time steps. In each time step, the players choose actions simultaneously. After the actor has performed $(a_1 \dots a_n) \in A^n$, it chooses an action $a \in A$, such that $(a_1 \dots a_n, a)$ is a prefix of some sequence in P. The observer simultaneously selects an action $d \in D$. Based on these two actions, the nature selects stochastically the observed action according to $m(d)$.

Utility The game ends when the actor completes any sequence from P. The utility function of the game captures the aims of the players. The utility value for the actor depends on the payoff g for the executed plan and it is discounted by the number of times an action from the executed plan has been correctly observed:

$$u(a_1 \dots a_h, o_1 \dots o_h) = \frac{g(a_1 \dots a_h)}{1 + \sum_{i \in \{1 \dots h\}; o_i = a_i} 1} \quad (1)$$

We assume the game to be zero-sum, which is a natural assumption in the adversarial plan recognition. The utility value is maximized by the actor and minimized by the observer. The method we propose for solving this game in Section 4 does not depend on the definition of the utility function so it can easily be adjusted for needs of a specific domain. Even if the zero-sum assumption does not hold, the model can still be used to compute solutions with a guaranteed (possibly sub-optimal) quality against any adversary.

Information We assume that both players know all parts of the definition of the game. The information about the progress of the game is much more limited. Neither of the players can directly observe the actions of the other player. The observer knows its own actions (selection of the classifiers) and the observations generated by nature based on the actions. The actor can observe only its own actions.

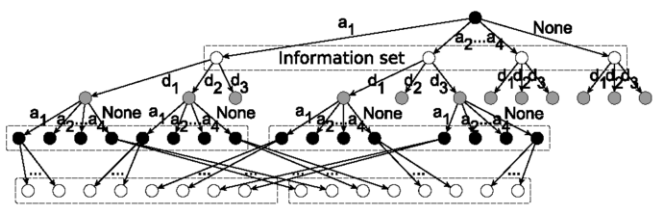

Figure 1: Portion of the extensive-form representation of the game. The leafs in the picture are not actual leafs in the game, the game continues for large (possibly infinite) number of steps.

APRG as an Extensive Form Game The restrictions on the available information create a specific structure of information sets[3]. Part of the complete extensive-form tree of the game is presented in Figure 1. The tree of the game can be arbitrarily deep. The longest branch is three times longer than the length of the longest plan in P. Hence, the "leaves" in the figure are just root nodes of subtrees omitted for clarity. Three plies of nodes are periodically repeating in the tree. The actor's decision nodes (black), the observer's decision nodes (white) and the observation nodes (grey). The actor's decision nodes must respect the plan tree P; hence, the available actions in one stage of the game are based on the actions performed by the actor in the previous stages. The observer's decision nodes contain always the same set of actions corresponding to all available classifiers (D). The observation nodes are chance nodes with probability distribution based on the classifier quality matrices (m) and the preceding actions of the actor and the observer. The structure of the information

[3] Information set is a set of game states that cannot be distinguished from each other by a player.

sets is consistent with the assumptions stated above as well as with the intuition of simultaneous moves of the players. We refer to the three plies of different players starting with the actor's decision as a (simultaneous) *move*.

3.2 Two Solutions of the Game

The proposed APRG is both a game and a plan recognition problem. As a result, it has two solutions. From the game perspective, the solutions are the action selection strategies of the players that optimize the tradeoffs of plan value and detectability. From the plan recognition perspective, it is the plan of the actor that is most probable given the observations. Both solutions are based on the assumption of rationality of the players.

Game-theory perspective For the game-theoretic solution, we use the well-known concept of Nash equilibrium (NE). It is a pair of strategies, such that none of the players has an incentive to deviate from its strategy if the other player's strategy stays unchanged, i.e., the observer cannot improve the chance to detect the plans the actor is likely to play and the actor cannot pick a plan that would have a better utility after the discount for detection. Further, we allow the agents to use *mixed strategies*, i.e., they can choose their actions based on fixed probability distributions in each information set of the game. This is necessary to achieve the optimal behavior in this game. For example, if the actor has two similarly good actions that can be detected by different actions of the observer, it should randomize among these options to make detection more difficult.

The most fundamental refinement of Nash equilibrium for imperfect-information extensive-form games is the *Sequential Equilibrium*, an imperfect-information-equivalent of the better known subgame-perfect Nash equilibrium used in perfect-information extensive-form games [11]. For this paper, it is sufficient to know that the strategies in this equilibrium can be represented as *behavioral strategies*, i.e., the probabilities of selecting each action in each information set.

Plan recognition perspective Even before the game starts, many of the plans in set P are clearly not going to be selected by a rational actor. Only the plans that have non-zero probability in some Nash equilibrium of the game can be selected. If the game has a single Nash equilibrium that does not require randomization, it can be recognized as the plan of the actor without any observations. On the other hand, more complex games require randomization. In that case, the task of the observer is to identify what are the random choices of the actor in the equilibrium strategy it currently plays. The result of the plan recognition process is the posterior probability distribution on the set P. This can be used to identify the plan that has been most probably executed by the actor or determine the probability of a specific plan chosen in advance.

4 Solution method

The game defined above is a standard extensive-form game with imperfect information and chance nodes and, in principle, it can be optimally solved by a standard algorithm for solving the games, such as, the linear program for sequence-form representation of the game[11]. The size of the linear program is polynomial in the size of the game tree. However, the size of the game tree is exponential in the number of actions of the players and the length of the game. For example, if each player has 5 applicable choices in each decision node and the game is played for 10 moves (corresponds to 30 plies) the size of the full game tree is more than 10^{20}. As a result, even traversing the whole game tree is practically impossible with current hardware. Therefore, we propose a method that approximates the optimal solution of the game.

4.1 Reduced Game Representation

The full tree of APRG has a very specific structure caused by the limited amount of information available to the players. The actor never learns any information about the observer's actions and the observer learns only the outcomes of the chance nodes. As a result, the information sets of the players include a large number of nodes in the tree. This fact allows us to represent the game in a more compact way without any loss of information or the expressiveness of the computed strategies. The basis of the representation is the concept of a signal tree [5]. A signal tree represents the space of all possible developments of a game from a single player's perspective. In general, plies of player's actions and observations are alternating in the tree. After any observation node, a child corresponding to each of the possible observations that can follow the player's decision is added.

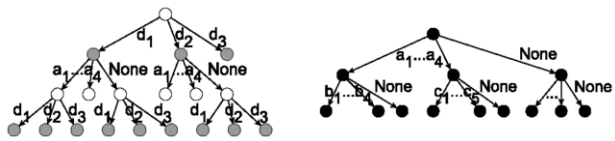

Figure 2: Signal trees of the observer (left) and the actor (right).

The actor does not obtain any observation in the game. Hence, its signal tree is the tree of its possible plans (see Figure 2). The observer obtains an observation after each decision. His signal tree contains both the observation and the decision plies. In general, each decision node has exactly $|D|$ children and each observation node has exactly $|A|$ children. However, forward pruning based on the domain-specific knowledge can be applied. The depth of the defender's signal tree, in the worst case, is uniform and it is two times deeper than the length of the longest attack plan. If the confusion matrices m contain sharp ones or zeros, the size of the tree can be reduced.

Any branch from one tree in combination with a branch of the matching length from the other tree defines a unique history h in the extensive-form game – the full game tree. All histories that correspond to the same decision node in the signal tree of a player belong to the same information set of the player. Also, all information sets that exist in the full EFG are represented by a node in one of the signal trees. As a result, any behavioral strategy can be represented as a probability distribution in each decision node of the signal tree.

The size of the two signal trees is much smaller than the size of the full game tree. With the 5 choices and 10 moves, the size of the attacker's signal tree is less than 10^7 and the size of the defender's tree is less than 10^{14}. It is still not possible to traverse the observer's signal tree in a reasonable time, but the reduction is substantial. In order to make this representation of the game equivalent to the full game tree, we have to solve two issues. First, the chance nodes in the observer's tree must become more complex. One chance node in the observer's signal tree corresponds to many chance nodes in the full game tree. However, the probabilities of the observations in the observer's signal tree depend only on the current (last) action of the actor. Hence, if the chance node stores a conditional probability $p(o|a)$, we do not lose any information about the distributions. The second issue are the utilities, which could, in principle, be different for each leaf of the full game tree. All this information cannot be

explicitly represented in the signal trees. On the other hand, the utility in more complex EFGs is usually represented implicitly, in the form of an algorithm that computes the utilities for each history of the game (see formula (1)). For any pair of branches from the signal trees, the corresponding history in the full game can be computed and we can use the same algorithm to compute the utilities.

4.2 Concurrent Monte-Carlo Tree Search

We propose a method, for computing the strategy in the game, based on concurrent construction of both the actor's and the observer's signal tree by Monte-Carlo Tree Search (MCTS). Our method is a generalization of MCTS developed for perfect information games [8]. A similar approach has been proposed by [2] and applied to the game of phantom tic-tac-toe. However, we extend the algorithm in two directions to make it applicable to our problem. The first is handling of the conditional chance nodes that appear in the defender's signal tree. The second extension is enabling the algorithm to further improve the players' strategies during the game play.

The algorithm maintains two MCTS-like trees corresponding to the two signal trees defined in Section 4.1. We present the pseudocode of the algorithm in Figure 3. At the beginning of the algorithm, both trees contain only one node – the root. The trees are then gradually expanded. We denote the root nodes $aRoot$ and $dRoot$ for the actor and the observer respectively. The main loop of the algorithm performs identical iterations until it is out of time. It is an anytime algorithm; the more computation time it has allocated the better is the result it produces.

In each iteration, first the plan for the actor is selected (procedure $selectActorPlan$). Starting from the root node of the actor's tree, the actions of the actor's plan are selected based on a suitable MCTS selection function, which balances the exploration and exploitation (lines 1-5). When the selection reaches a leaf of the currently constructed part of the tree and it is not the end of some actor's plan, all possible direct successors of the node are added to the tree (line 7). Then, the plan is competed randomly to a full plan of the actor (lines 8-12). At the end, the actor's plan is returned $(a_1 \dots a_h) \in P$.

Next, a plan for the observer is selected in a similar way from his signal tree (procedure $selectObserverPlan$). In the observer's decision nodes, a suitable selection function is applied (line 4). In the observation nodes, conditional probability distributions $p(o|a, d)$ from m are used to select a child. The algorithm first selects the plan for the actor; hence, we can identify the right distribution in the matrix at this point (line 6). If the observation node is a leaf of the constructed part of the tree, the descent in the tree stops (line 8). Otherwise, the child corresponding to the generated observation is selected (line 9). The selection of the observer's plan continues until either depth h (i.e., the length of the actor's plan) is reached or a leaf in the observer's tree is selected. In the earlier case, the procedure ends (line 11). In the latter case, the children of the selected leaf are added to the tree (lines 13, line 19) and the variables are modified so that the simulation can start in an observer's decision node. The rest of the observer's actions are selected randomly and the resulting observations are computed (lines 22-27). At the end, we have the observer's plan and the observations it produced $(d_1 \dots d_h, o_1 \dots o_h)$.

With the plan of the actor and the observer's observations, we can compute the utility function of the players in the $Main$ procedure (line 4). The utility is then back-propagated in both trees as in MCTS (procedure $backPropagate$). Starting in the selected leafs of the trees, the statistics in the nodes are updated. Afterwards, next iteration of the $Main$ procedure starts.

Procedure: Main
1: **loop**
2: (aLeaf, aPlan) = selectActorPlan(aRoot)
3: (dLeaf, dPlan, Obs) = selectObserverPlan(dRoot, aPlan)
4: u = utility(aPlan, Obs)
5: backPropagate(aLeaf, u); backPropagate(dLeaf, u)
6: **end loop**

Procedure: selectActorPlan(aRoot)
1: curNode = aRoot
2: **while** curNode is not leaf **do**
3: action = select(curNode); aPlan = aPlan + action
4: curNode = child(curNode, action)
5: **end while**
6: **if** curNode is plan end **then return** (curNode, aPlan)
7: addNewChildren(curNode)
8: aLeaf = select(curNode); curNode = aLeaf
9: **while** curNode is not plan end **do**
10: action = random(curNode); aPlan = aPlan + action
11: curNode = createNodeAfter(curNode, action)
12: **end while**
13: **return** (aLeaf, aPlan)

Procedure: selectObserverPlan(dRoot, aPlan)
1: curNode = dRoot
2: **while** curNode not leaf & length(aPlan) > 0 **do**
3: aAction = popFirst(aPlan)
4: dAction = select(curNode); dPlan = dPlan + dAction
5: obsNode = child(curNode, dAction)
6: obs = confusionMatrixSample(dAction, aAction)
7: Observations = Observations + obs
8: **if** obsNode is leaf **then break**
9: **else** curNode = child(obsNode, obs)
10: **end while**
11: **if** length(aPlan) $= 0$ **then return** (curNode, dPlan, Observations)
12: **if** curNode is leaf & length(aPlan) > 0 **then**
13: addNewChildren(curNode)
14: dAction = select(curNode); dPlan = dPlan + dAction
15: obsNode = child(curNode, dAction)
16: obs = confMatSample(dAction, aAction)
17: Observations = Observations + obs
18: **else**
19: addNewChildren(obsNode)
20: curNode = child(obsNode, obs)
21: **end if**
22: **while** length(aPlan) > 0 **do**
23: aAction = popFirst(aPlan)
24: dAction = random(curNode); dPlan = dPlan + dAction
25: obs = confMatSample(dAction, aAction)
26: Observations = Observations + obs
27: **end while**
28: **if** curNode is leaf **then return** (curNode, dPlan, Observations)
29: **else return** (obsNode, dPlan, Observations)

Procedure: backPropagate(node, u)
1: updateStatistics(node,u)
2: **if** node is not root **then** backPropagate(parent(node),u)

Figure 3: The concurent monte carlo tree search algorithm for the first stage of the game. Procedures $select$ and $updateStatistics$ implement some MCTS selection strategy, such as UCT of EXP3.

4.3 Convergence of the Algorithm

We have implemented this algorithm under the hypothesis that with a suitable selection function, it converges to the Nash equilibrium of the game. We do not have a full formal proof supporting it yet.

It has been proven that in a one stage normal form game, the overall frequencies of using individual actions with EXP3.1[1] as the selection function converges to the Nash equilibrium, even if the utilities in the game and the number of actions of the opponent are unknown to the players [6]. This fact has been used to create an algorithm for playing an imperfect information version of tic-tac-toe [2], which is in spirit very similar to our algorithm. It uses separate trees for the players, but does not deal with chance nodes and continuous reasoning. The author claims that even in this setting, his algorithm converges to a NE of the extensive form game. However, he does not provide a formal proof or extensive experimental evidence to support this claim. On the other hand, his as well as our experimental results indicate that this method can be used to create successful players for imperfect information extensive form games and we believe that it is possible to proof the convergence for a large class of games.

If an agent uses EXP3.1 as the selection function in a repeated problem, its loss for not playing the optimal action all the time (i.e., regret) can be bounded by $O(1/\sqrt{(T)})$ [1], where T is the number of trials. It means the quality of the produced solution gradually improves. Even before full convergence of the algorithm, the computed strategy guarantees a bound on its sub-optimality.

4.4 Continuous Reasoning

The described algorithm finds a suitable course of action for both players at the beginning of the game. In theory, it can be run for a long time before the game starts and the computed trees can be used for playing the game without further computation. However, the size of the trees that would need to be stored and the time required to compute good strategies in their lower levels would be huge. For practical applications, we suggest computing an initial strategy and adding more iterations after each step of the game.

The continuous reasoning is slightly different for each player. We further describe the algorithm for the observer, which is the main focus of this paper. After each move, the root of the observer's tree is substituted by its grandchild corresponding to the selected observer's action and the actual observation in the game. The situation is more complex in the actor's tree. The observer generally cannot be sure about the current actor's tree node in the later stages of the game. Instead, the defender maintains a probability distribution among k most probable nodes where the attacker can be. We call the nodes the *root set*. The probability of the nodes in the set can be computed by Bayesian inference form the probabilities computed by the algorithm in the previous stage and the confusion matrices. For each node in the root set of the current stage, the concurrent MCTS converges to the probability of executing each of the actions by rational actor in the current situation ($p(a)$). The probability that the actor is in the child under the action a given the observation o can be computed as:

$$p(a|o) = \frac{p(o|a)p(a)}{p(o)} = \frac{p(o|a)p(a)}{\sum_{a_i \in A} p(o|a_i)p(a_i)} \qquad (2)$$

The probabilities of different observations given an attacker's action ($p(o|a_i)$) can be extracted from the confusion matrix that corresponds to the classifier selected by the observer in the game.

After the probability of each child of the nodes in the current root set is computed, the k most probable children are selected to the root set in the following stage. Further, for each actor's action, we can compute the probability that the action was the last action executed by the actor. It is the sum of probabilities of executing the action in each node of the root set. The action with the highest execution probability is the observation generated by the algorithm and used in evaluation as the input to the utility function (1).

When concurrent MCTS runs with a root set containing multiple nodes, one of them is randomly selected as the root of the actor's tree in each iteration proportionally to their probabilities.

5 Experimental Evaluation

We evaluate the presented approach on a simplified network security domain. The actor is the attacker that tries to attack a computer network. The attacker can perform one of 12 actions (set A), such as vertical ping scan, or brute-force attack to a specific service. These actions have preconditions and postconditions and their legal sequences generate over 150 thousand different attack plans of different lengths (set P). Each action provides the attacker with some knowledge about the network. The values g of the attack plans are defined in terms of this knowledge and vary between 0 and 120. The observer is the intrusion detection system operating on the network. It has three different settings; each of them can detect some of the attacker's actions better and other worse. The actions of the observer (set D) are selecting one of these settings. The three classes of actions that are detected better in different settings are producing high, medium, and small volume of network traffic. The classes include two, four, and six actions respectively. Each setting detect the actions from the class correctly with 90%, 60%, and 30% precision, misclassifying in remaining cases evenly as any other action. This defines the confusion matrices m.

We performed the experiments with a single-thread implementation of the algorithm presented in Section 4 on Intel(R) Core(TM) i7 CPU @ 3.20GHz. In order to select (close to) rational attack plans, we run the proposed algorithm for 10 minutes in the initial stage of the game. Then, we took five different high probability plans, that are presented in Figure 4. They are ordered by number of samples that used the plans in the attacker's tree, with the first plan having 30% of all samples and the fifth a little more than 1%. For example in the third plan, the attacker attempts to compromise a server behind a firewall. To gain access to the private network, he sends malware via email to the users in the network and gains access to some of their hosts. Afterwards, he scans the private network to find a server running a specific service and he tries to obtain more information about the version of the service to apply a suitable exploit.

The observations generated for the observer are stochastic; hence we run the recognition process 200 times for each plan. The algo-

first: *dns_requests, Horiz_scan_for_spec_service, web_attacks, connect_to_host, fingerprinting* ($g = 86$)
second: *dns_requests, SEND_SPAM, connect_to_host, Horiz_scan_for_spec_service, fingerprinting* ($g = 86$)
third: *SEND_SPAM, connect_to_host, Horiz_scan_for_spec_service, fingerprinting* ($g = 81$)
fourth: *dns_requests, Horiz_scan_for_spec_service, web_attacks, connect_to_host* ($g = 78$)
fifth: *Horiz_scan_for_spec_service, web_attacks, DDOS_TO_SPECIFIC_SERVICE, fingerprinting, connect_to_host* ($g = 93$)

Figure 4: Attack plans used in evaluation. The upper cased actions are in the high traffic category, medium traffic actions have capital letter at the beginning and the low traffic actions are lowercased.

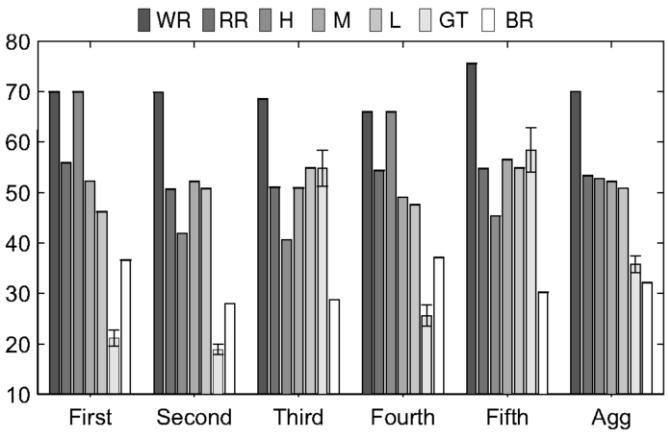

Figure 5: The mean attacker's reward (defenders penalty) with the proposed approach (GT) and the base lines; RR - random classifier; BR - the ex post optimal classifier; and other baselines.

rithm was set to use 2 minutes of computational time per game stage. We compare the performance of the proposed algorithm in terms of the utility to several baseline values. The first three baselines always use only one of the actions available to the defender. We denote them by the first character of the volume class (H,M,L). The other two baselines are the ex post best response (BR) and worst response (WR). With the prior knowledge of the actual attacker's plan, we compute BR (WR) by selecting the defender's action with the highest (lowest) probability of observing the actual attacker action. The last baseline is the random selection of defender's action (RR). As all these baselines are independent of defender's observations, we can quickly compute a good approximation of their quality by averaging over 10^6 trials for each plan.

Figure 5 presents the utility of the proposed method (GT) and the baselines computed by formula (1). The values are presented for each attacker's plan separately and aggregated over all five plans (Agg). Overall, the GT approach is significantly better then all applicable baselines and it is only slightly worse than the BR. The error bars indicates the 95% confidence interval of the mean. For all baseline values, the confidence intervals are hardly noticeable in the graphs, because of their small width. Looking at the individual plans, the GT approach performs very well on the first two plans and the results are worst for the fifth plan. This can be expected as the first plans better approximate the rational behavior of the attacker and the later plans have higher chance to be irrational. Note that the result of GT can be better than BR, because it combines the information from the classifiers with the game theoretic reasoning and does not directly outputs the observation of the classifiers as BR.

In the plan recognition task, the plan actually performed by the attacker was ranked as the most likely by the GT approach in 38.6% of runs and its median position in the final list of the most probable plans was 5. This is a promising result considering there are over 150 thousand plans to choose from.

The number of iterations (samples) of the concurrent MCTS algorithm performed in a single stage varied with the stage of the game. The histogram of the number of instances in which certain number of iterations was performed follows roughly binomial distribution. In most instances, more than 10^7 iterations were performed, however, in some cases even 10^8 can be made in time. One instance of the algorithm never crossed 4.5GB memory limit during the experiment. The memory was used for storing the MCTS trees, size of which can be substantially limited by a more conservative expansion strategy.

6 Conclusion

In this paper, we argue that the problem of adversarial plan recognition, where the observed agent actively tries to avoid observation should be modeled as a game. The observed agent has to model reasoning of the observer and vice versa in order to derive the optimal strategies. We model the problem as a specific subclass of imperfect-information zero-sum extensive form games, which we term Adversarial Plan Recognition Games. We show how problems from this class can be compactly represented by a pair of signal trees of the players. Based on this representation, we propose an algorithm that approximates the optimal solution of the game. The algorithm is a novel generalization of Monte-Carlo tree search. In order to verify applicability of APRG, we use it to model a simplified game between the attacker and the intrusion detection system in the network security domain. The presented experimental evaluation shows that the proposed algorithm can be used to produce good behaviors for the agents in reasonably large domains. The observer using the proposed algorithm performed significantly better than any of the naive baseline approaches we tried.

The proposed model can be extended in several interesting directions. In real world applications, it is not likely that the agents involved will be perfectly rational. It would be interesting to relax the rationality assumption and use a suitable bounded rationality model. Furthermore, it would be interesting to extend the model to allow temporally extended actions. If various actions of the actor take different time, the model cannot be directly applied because of more complex synchronization between the observer and the actor. Solving this issue is also an important line of future research.

ACKNOWLEDGEMENTS

This research was supported by AFOSR (grant no. FA8655-10-1-3016), ONRG (grant no. N62909-12-1-7019), and the Czech Science Foundation (grant no. P202/12/2054).

REFERENCES

[1] P. Auer, N Cesa-Bianchi, Y Freund, and R.E. Schapire, 'The non-stochastic multiarmed bandit problem', *SIAM Journal on Computing*, **32**(1), 48–77, (2003).

[2] D. Auger, 'Multiple Tree for Partially Observable Monte-Carlo Tree Search', in *Applications of Evolutionary Computation*, pp. 53–62. Springer, (2011).

[3] D. Avrahami-Zilberbrand and G.A. Kaminka, 'Incorporating observer biases in keyhole plan recognition (efficiently!)', in *Proc. of the National Conference on Artificial Intelligence*, volume 22, p. 944, (2007).

[4] S. Braynov, 'Adversarial planning and plan recognition: Two sides of the same coin', in *Secure Knowledge Management Workshop*, (2006).

[5] P. Ciancarini and G.P. Favini, 'Monte Carlo tree search in Kriegspiel', *Artificial Intelligence*, **174**(11), 670–684, (jul 2010).

[6] C. Daskalakis, A. Deckelbaum, and A. Kim, 'Near-optimal no-regret algorithms for zero-sum games', in *Proc. of the Twenty-Second Annual ACMSIAM Symposium on Discrete Algorithms*, pp. 235–254, (2011).

[7] C.W. Geib and R.P. Goldman, 'Plan recognition in intrusion detection systems', in *DARPA Information Survivability Conference & Exposition II*, volume 1, pp. 46–55. IEEE, (2001).

[8] L. Kocsis and C. Szepesvári, 'Bandit based Monte-Carlo Planning', in *ECML-06*, (2006).

[9] W. Mao and J. Gratch, 'A utility-based approach to intention recognition', in *AAMAS 2004 Workshop on Agent Tracking: Modeling Other Agents from Observations*, (2004).

[10] X. Qin and W. Lee, 'Attack Plan Recognition and Prediction Using Causal Networks', in *20th Annual Computer Security Applications Conference*, pp. 370–379. Ieee, (2004).

[11] Y. Shoham and K. Leyton-Brown, *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*, Cambridge University Press, 2009.