Learning to Play Stackelberg Security Games

Avrim Blum Nika Haghtalab Ariel D. Procaccia

As discussed in previous chapters, algorithmic research on Stackelberg Security Games has had a striking real-world impact. But an algorithm that computes an optimal strategy for the defender can only be as good as the game it receives as input, and if that game is an inaccurate model of reality then the output of the algorithm will likewise be flawed. Consequently, researchers have introduced Bayesian frameworks that capture uncertainty using a probability distribution over possible games. Others have assumed that the unknown parameters of the game lie within known intervals. These approaches are discussed in Chapter 17 of this book [17].

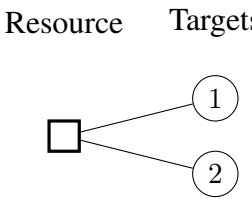
In this chapter, we present an alternative, *learning-theoretic* approach for dealing with uncertainty in Stackelberg security games. In order to paint a cohesive picture, we focus on one type of uncertainty: unknown attacker utilities. Learning will take place in a *repeated Stackelberg security game*, where the defender gathers information about the attacker purely by observing the attacker's responses to mixed strategies played by the defender.

In more detail, we wish to learn a good strategy for the defender without any initial information about the utility function of the attacker (Section 1); when given a distribution over attacker types (Section 2); and when faced with an unknown sequence of attackers (Section 3). In each section we present, in some generality, the relevant learning-theoretic techniques: optimization with membership queries, Monte Carlo tree search, and no-regret learning, respectively. In Section 4 we briefly discuss additional work at the intersection of machine learning and Stackelberg security games.

1 Single Unknown Attacker

In this section, we describe a setting introduced by Letchford et al. [19], and present a method designed by Blum et al. [7] for using best response queries to learn the optimal strategy against a single attacker with unknown utilities. The main result discussed in this section involves learning the optimal strategy of the defender with a number of best-response observations that is polynomial in the parameters of the game.

Consider a *Stackelberg Security Game (SSG)* denoted by a 3-tuple (N, U, M) where N is the set of n *targets*, U indicates the *utilities* associated with the defender and attacker, and M indicates all the possible subsets of targets that can be simultaneously defended in valid deployments of resources. More precisely, M is determined by the set of targets N, set of *resources* R, a set $\mathcal{D} \subseteq 2^N$ of *schedules* where each $D \in \mathcal{D}$ represents a set of targets that can be simultaneously defended by one resource, and function $A : R \to 2^{\mathcal{D}}$ that indicates the set of all schedules that can be defended by a given resource. Let there be m pure strategies, then M is a zero-one $n \times m$ matrix,



Utilities

Targets	Defender		Attacker Type θ_1		Attacker Type θ_2	
i	$u_d^c(\cdot)$	$u_d^u(\cdot)$	$u_a^c(\cdot)$	$u_a^u(\cdot)$	$u_a^c(\cdot)$	$u_a^u(\cdot)$
1	0	-1	0	1	0	1
2	0	-1	0	1	0	0

Figure 1: An example of a security game with one resource and two targets, with two possible utilities for the attacker.

with rows that represent targets and columns that represent pure strategies, such that $M_{ij} = 1$ if and only if target i is covered by some resource in pure strategy j.

A *mixed strategy* of the defender is a distribution over pure strategies and is represented by a probability vector $\mathbf{s} \in \Re^m$. The *coverage probability vector* of a strategy $\mathbf{s}$ is a vector $\mathbf{p} \in [0,1]^n$ such that for all $i \in N$, p_i is the probability with which target i is defended under $\mathbf{s}$. The coverage probability vector of $\mathbf{s}$ is given by $\mathbf{p} = M\mathbf{s}$.

Let $u_d^c(i)$ and $u_d^u(i)$ indicate the defender's payoffs when target i is attacked and is, respectively, covered or not covered by the defender. Similarly, $u_a^c(i)$ and $u_a^u(i)$ are the attacker's payoffs from attacking target i when it is, respectively, covered or not covered by the defender. Then under coverage probability vector $\mathbf{p}$, the expected utility of the defender when target i is attacked is given by $U_d(i,\mathbf{p}) = u_d^c(i)p_i + u_d^u(i)(1-p_i)$ and the expected utility of the attacker is $U_a(i,\mathbf{p}) = u_a^c(i)p_i + u_a^u(i)(1-p_i)$. Note that $U(i,\mathbf{p})$ is a linear function in p_i. Since $U(i,\mathbf{p})$ only depends on p_i, with a slight abuse of notation, we use $U(i,p_i)$ when it is convenient.

Let $\mathbf{p}$ be the coverage probability vector of strategy $\mathbf{s}$. The attacker *best-responds* to $\mathbf{s}$ by attacking target $b(\mathbf{s}) \in \arg\max_{i \in N} U_a(i,\mathbf{p})$. Note that given a coverage probability vector and the attacker's utilities, the attacker's best response is invariant to the mixed strategy that is used to implement that coverage probability vector. Therefore, we can work directly in the space of valid coverage probability vectors.

When $U_a(\cdot)$ is unknown, one cannot find the optimal strategy of the defender directly. However, observing the attacker's response to some carefully chosen strategies can help fill in the missing information required for finding the optimal strategy. As a warmup, consider a security game with two targets and one resource that can defend only one of the targets at a time. The defender's payoff for either of the targets is -1 when they are attacked and are left unprotected, and 0 otherwise. Consider two possible utilities for the attacker. In the first case (type θ_1), the attacker values both targets equally, with utility 1 for attacking a target that is not protected and 0 for attacking a target that is protected. In the second case (type θ_2), the attacker's utility for target 1 is the same as in case 1, however, there is 0 payoff for attacking target 2 whether or not it is protected (See Figure 1). The attacker's type is not known a priori. Consider the strategy where the resource covers target 1 with probability $\frac{2}{3}$ and target 2 with probability $\frac{1}{3}$; it induces the coverage probability vector $\mathbf{p} = (\frac{2}{3}, \frac{1}{3})$. An attacker of type θ_1 would respond to $\mathbf{p}$ by attacking target 2, whereas an attacker of type θ_2 would respond to $\mathbf{p}$ by attacking target 1. So, by observing the attacker's response to $\mathbf{p}$, we can detect the type of the attacker in play and use the optimal strategy for that attacker type.

In the above example, we demonstrated how the uncertainty about two possible discrete types

of attackers was resolved by observing the best response of the attacker. However, the results discussed in this section are more general, as we can pinpoint the optimal strategy with no prior knowledge regarding the possible types of the attackers. To describe these methods, we first review how the optimal strategy can be computed when $U_a(\cdot)$ is known.

One of the methods used for computing the optimal strategy in SSGs is the Multiple LPs approach [12]. In this formulation, we solve one linear program (LP) for every target $i \in N$. This LP, as shown in Equation (1), finds the optimal defender mixed strategy among all strategies that cause an attack on target i. The optimal defender's strategy is the best strategy among the solutions to these LPs; the one with the highest payoff to the defender.

$$\begin{aligned} \text{maximize} \quad & U_d(i, \sum_{j:M_{ij}=1} s_j) \\ \text{s.t.} \quad & \forall i',\ U_a(i', \sum_{j:M_{i'j}=1} s_j) \leq U_a(i, \sum_{j:M_{ij}=1} s_j) \\ & \forall j,\ s_j \geq 0 \\ & \sum_{j=1}^{m} s_j = 1 \end{aligned} \tag{1}$$

When $U_a(\cdot)$ is unknown, one cannot explicitly write down these LPs. However, the attacker's best response to $\mathbf{s}$ tells us which of the n feasible regions $\mathbf{s}$ belongs to. Therefore, in a *repeated* Stackelberg game against a single attacker, the defender can hope to construct the hidden LPs via *best-response queries*, that is, by observing the attacker's responses to a sequence of strategies. This is indeed the approach taken by Letchford et al. [19], but the number of observations they require is polynomial in the number of pure leader strategies in a Stackelberg game, which, in the case of SSGs, could be exponential in the representation of the game.

Here we focus on the approach of Blum et al. [7], which learns the optimal strategy using only a polynomial number of best-response queries in SSGs. This improvement in the number of queries is mainly based on the observation that the defender's utility can be optimized in the feasible region of a given LP without actually reconstructing these feasible regions. Formally:

Theorem 1 (Blum et al. [7])**.** *Consider an SSG with n targets. For any $\epsilon, \delta > 0$, with probability $1 - \delta$, it is possible to learn a defender strategy that is optimal up to an additive term of ϵ, using a number of best response queries that is polynomial in n and logarithmic in $1/\epsilon$ and $1/\delta$.*

In the remainder of this section, we first provide an overview of some learning-theoretic techniques for optimizing linear programs without explicit knowledge of the constraints, and then provide an outline of the algorithm of Blum et al. [7] based on these techniques.

To perform optimization over a region, we need to find at least one point in that region. Conceptually, a key challenge here is that it could be that the optimal defender strategy involves getting the attacker to attack a particular target i, and yet random allocation of resources might have a tiny chance of getting the attacker to do so; thus, some form of intelligent search through the strategy space will be needed. We first describe how to optimize over a region given an initial point in that region, and then discuss how we can perform this intelligent search.

1.1 Optimization With Membership Queries

A classic result of Khachiyan [15] shows that one can solve linear programs in polynomial time using the *ellipsoid algorithm*. This algorithm has the additional property that it does not need to know the constraints of the linear program explicitly so long as it has access to a *separation oracle*. This is an oracle for the feasible region $\mathcal{P} \in \Re^n$ that given a proposed solution x will either answer "yes" if $x \in \mathcal{P}$ or else will produce a violated constraint (a halfspace that contains $\mathcal{P}$ but does not contain x) if $x \notin \mathcal{P}$. However, sometimes one has only a weaker capability: the ability to answer whether a proposed point x belongs to a given convex region $\mathcal{P}$ or not, without being able to produce a violated constraint when answer is "no". Such a capability is called a *membership oracle*. A specialized version of the ellipsoid algorithm is shown in Grötschel, Lovász, and Schrijver [13] to be able to solve optimization problems of this form — that is, optimizing a linear function over a convex region, where the convex region is given a membership oracle — when one is given an initial starting point $\mathbf{x}_{init} \in \mathcal{P}$ such that a ball of sufficiently large radius r about $\mathbf{x}_{init}$ is also contained in $\mathcal{P}$. However, the algorithm is quite complex and slow. More recently, Bertsimas and Vempala [6] and Tauman Kalai and Vempala [24] have shown how simpler and faster random-walk based algorithms can be applied to solve this problem as well, with the fastest being the algorithm of [24]. This algorithm overall makes $\tilde{O}(n^{4.5})$ calls to the given membership oracle, where the $\tilde{O}$ notation hides polylogarithmic factors.

The algorithm of Tauman Kalai and Vempala [24] can be viewed as a form of simulated annealing. Given an initial point $\mathbf{x}_{init}$ such that a ball of radius r about x_{init} is contained inside $\mathcal{P}$, it maintains a "temperature" parameter T that begins initially at a radius R such that we are guaranteed that $\mathcal{P}$ is contained within a ball of radius R. (All these algorithms will have a polylogarithmic dependence on the ratio R/r.) This temperature is then slowly lowered over time. If $\mathbf{c} \cdot \mathbf{x}$ is the objective we are aiming to minimize, the algorithm's goal on each round is to select a random point inside the convex body according to a probability distribution where the probability of some point $\mathbf{x}$ is proportional to $e^{-\mathbf{c}\cdot\mathbf{x}/T}$. This is an easier (though still nontrivial!) task to perform for large values of T. What Tauman Kalai and Vempala [24] show is how to use a solution for one value of T to help solve for an answer for the next lower value of T.

The selection of a random point inside $\mathcal{P}$ from the distribution for a given value of T is done via the following random walk procedure. Starting from the output of the previous value of T, a random vector is drawn through the current point with a distribution that depends on an estimated covariance matrix for the distribution for the previous value of T (this covariance matrix is estimated by taking several samples from the previous value of T). Then the point is updated to a random point inside $\mathcal{P}$ along that line, with probability density proportional to the objective function $e^{-\mathbf{c}\cdot\mathbf{x}/T}$. This process has a stationary distribution that equals the distribution we would like to be sampling from for this value of T, and what is proven in [24] is that the walk quickly approaches this distribution. Finally, once T is sufficiently small, nearly all the probability mass of the stationary distribution is close to the optimum value of the objective. Putting this together, we have the following:

Theorem 2 (Tauman Kalai and Vempala [24])**.** *For any convex set $\mathcal{P} \in \mathbb{R}^n$, suppose we are given a linear objective $\mathbf{c}$, an accuracy parameter ϵ, a starting point $\mathbf{x}_{init} \in \mathcal{P}$, and values r, R such that a ball of radius r about x_{init} is contained in $\mathcal{P}$ and a ball of radius R about x_{init} contains $\mathcal{P}$. Then,*

using $\tilde{O}(n^{4.5})$ calls to a membership oracle for $\mathcal{P}$, the algorithm will with high probability output a point $\mathbf{x}^* \in \mathcal{P}$ such that $\mathbf{c} \cdot \mathbf{x}^* \leq \min_{\mathbf{x} \in \mathcal{P}} \mathbf{c} \cdot \mathbf{x} + \epsilon$, where the $\tilde{O}$ notation hides terms logarithmic in n, R/r, and $1/\epsilon$.

1.2 Using Membership Queries to Learn SSGs

In order to understand the connection between SSGs and optimization using membership queries, we make two changes to the LP in Equation (1). First, since the attacker's best response is independent of the strategy that is used to implement a given coverage probability, we restate the LP to use variables that represent the coverage probability vector rather than the mixed strategy used to implement it. Second, since $U_a(\cdot)$ is unknown, we change the constraints to implicitly determine whether a given target i is attacked as a best response to a strategy.

$$\begin{aligned} \text{maximize} \quad & U_d(i, \mathbf{p}) \\ \text{s.t.} \quad & i \text{ is attacked under } \mathbf{p} \\ & \mathbf{p} \text{ is implementable} \end{aligned} \tag{2}$$

To use the results from Section 1.1, we need to establish four properties regarding the above LP:

1. The objective function is linear in $\mathbf{p}$.
2. The optimization region is convex.
3. There is a membership oracle for the optimization region.
4. We have access to a well-centered initial feasible point in the optimization region.

Below we briefly explain how each of these four properties is satisfied.

For the first requirement, note that by the definition of the utility function, the objective function $U_d(i, \mathbf{p})$ is linear in $\mathbf{p}$. For the second requirement, for any i, let $\mathcal{P}_i$ denote the optimization region in Equation (2), i.e., $\mathcal{P}_i$ is the set of all coverage probabilities that can be implemented by some mixed strategy and lead to target i being attacked. $\mathcal{P}_i$ corresponds to a linear transformation of the optimization region in LP (1), which is itself the intersection of halfspaces defined by the linear constraints and, as a result, is convex. So, $\mathcal{P}_i$ is also convex.

For the third requirement, a natural membership oracle for $\mathcal{P}_i$ involves a two step procedure: For every $\mathbf{p}$, first solve the linear system $M\mathbf{s} = \mathbf{p}$ to find out whether $\mathbf{p}$ can be implemented by some mixed strategy $\mathbf{s}$, and then play $\mathbf{s}$ and observe the attacker's response to learn whether $\mathbf{s}$ causes an attack on target i.

The last requirement for using the results in Section 1.1 is having an initial feasible point in the region of optimization. A natural first attempt for finding an initial point in region $\mathcal{P}_i$ involves an algorithm that repeatedly plays a randomly chosen coverage probability vector and observes the attacker's response until one of the strategies induces an attack on target i. However, hitting a region with randomly chosen points in a high-dimensional space requires exponentially many attempts. So, this algorithm requires exponentially many best response queries just to find initial feasible points.

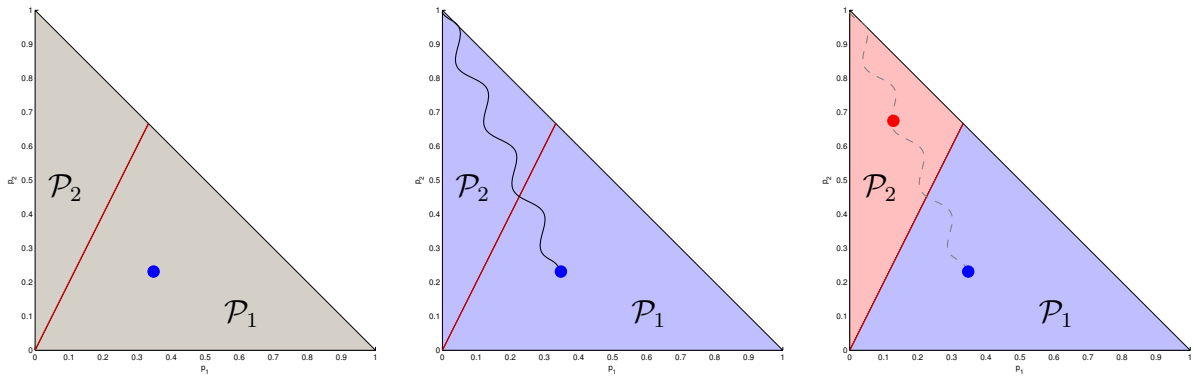


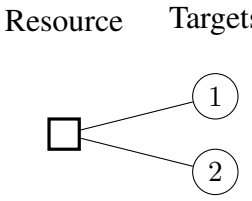
Figure 2: Three major steps of the algorithm for finding initial points in all regions. In the first step, one strategy is queried at random and is found to be in $\mathcal{P}_1$. In the subsequent steps, the optimization process only considers targets that are revealed so far, i.e., target 1. In the last figure, a strategy in region $\mathcal{P}_2$ is revealed, acting as an initial point in the newly discovered region $\mathcal{P}_2$.

Blum et al. [7] take an entirely different approach for finding initial feasible points. The method is based on an intuitive observation that, if a target is left unprotected at all times and yet never attacked even as the defender increases his probability of covering other targets, then the defender cannot possibly achieve a better payoff by defending that target. This insight motivates a procedure that, at every major step, only considers the optimization tasks for the targets that have been observed so far. Moreover, we will explicitly set coverage probabilities on targets never attacked so far to 0 even if the strategies output by the optimization happen as a by-product to give them non-zero coverage. If in this process, we observe that a strategy induces an attack on a target that has not been already seen, e.g. target i, then we have an initial point for this new region, $\mathcal{P}_i$. Otherwise, we have reached the optimal strategy (See Figure 2). Additional caution must be taken to ensure that these initial points are "well-centered". We leave it to the interested reader to refer to [7] for the details of this procedure.

2 Single Attacker Drawn from a Distribution

In Section 1 we discussed the case of a single attacker who is completely unknown initially. While the method given therein eventually converges to an almost optimal strategy for the defender, the learning process itself can lead to significant inefficiencies. The purpose of the approach described in this section is to trade off *exploration* and *exploitation*, that is, take advantage of useful knowledge about the attacker *as it is obtained*, while also taking appropriate actions to learn more about the attacker.

Taking a Bayesian viewpoint, suppose there is a set of possible attacker types Θ, where each attacker type $\theta \in \Theta$ is defined by that attacker's utility function. In this section it is assumed that a single attacker type θ is drawn from a known distribution $\mathcal{D}$ over Θ, and the defender plays a repeated Stackelberg security game with T rounds against an attacker of type θ. Importantly, the



Utilities

Targets	Defender		Attacker Type θ_1		Attacker Type θ_2	
i	$u_d^c(\cdot)$	$u_d^u(\cdot)$	$u_a^c(\cdot)$	$u_a^u(\cdot)$	$u_a^c(\cdot)$	$u_a^u(\cdot)$
1	0	$-L$	0	2	0	1
2	0	-1	0	1	0	1

Figure 3: An example of a security game with one resource and two targets, with two possible attacker types.

assumption that the defender has a prior over attacker types was not made in Section 1, that is, in this section the defender has more information upfront.

To gain some intuition about how the current setting differs from the one of Section 1, consider the game given in Figure 3, which is a slight variant of our previous example. In this game, Θ includes only two attacker types, and the attacker's identity can be determined in a single round by playing $(3/5, 2/5)$: if target 1 is attacked then the attacker is of type θ_1, and if target 2 is attacked then he is of type θ_2.

Now, suppose that $\mathcal{D}$ gives probability $1/2$ to each attacker type, and L is a large number such that $T \ll L$. Then under an optimal policy, the defender would always cover target 1 with probability $2/3$, because otherwise the defender would incur a huge loss in the event (which happens with probability $1/2$) that the attacker is of type θ_1. Indeed, in that case the attacker would attack target 1, causing the defender to incur a huge expected loss (at least $L/3$). In other words, an optimal policy would essentially instruct the defender to play it safe by assuming the attacker is of type θ_1, instead of using a single best response query to learn the attacker's type and play optimally thereafter.

So how does one compute such an optimal policy, given a known distribution over attacker types? Below we present an approach due to Marecki et al. [21], which relies on Monte Carlo Tree Search.

2.1 Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) is a popular search algorithm that tries to pinpoint optimal actions in a sequential decision making process. Its central idea is to iteratively build a search tree by using random sampling in order to evaluate each new node and update the evaluations of its ancestors. MCTS is becoming increasingly popular in artificial intelligence research, in part due to its successful applications to computer Go. Below we briefly present some of the algorithm's most salient features; for a detailed presentation, we refer the reader to the survey by Browne et al. [9].

For now, we will think of the sequential decision making process as being represented by a tree that is gradually uncovered. In its most abstract form, MCTS has two components. First, given the currently explored tree and statistics about its nodes, a *tree policy* traverses the current tree and pinpoints a node to be expanded, and an unexplored action to take at that node. This leads to the discovery of a new child of the expanded node, which is added to the tree. Second, a *default policy* is used to run a *simulation* — sample a path from the newly discovered node to a leaf. The value

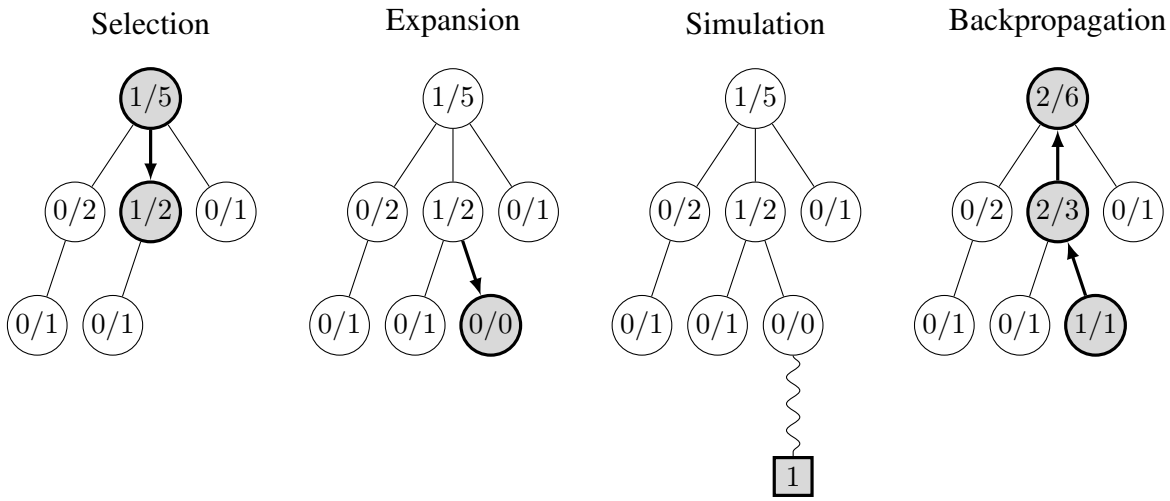


Figure 4: An illustration of one iteration of MCTS. In this example, the values of leaves are in $\{0, 1\}$. Each node is labeled by Q_v/N_v.

of this leaf is then used to update the evaluations of the new node as well as its ancestors.

In more detail, each node v in the search tree has a counter N_v, which counts the total number of simulations that involve v or its descendants, and a total value Q_v, which represents the sum of values of these simulations. The evaluation of v is $\bar{Q}_v = Q_v/N_v$. The main loop of MCTS has four distinct steps (illustrated in Figure 4):

1. *Selection*: Select a node to expand by traversing the tree according to the tree policy. This node must be an internal node that has unexplored children.

2. *Expansion:* Expand an unexplored child v, and add it to the search tree. The choice of specific child to expand is also governed by the tree policy. Set $N_v = 0$, $Q_v = 0$.

3. *Simulation:* Use the default policy to sample a path from v to a leaf node ℓ. For the purposes of this chapter, we will assume that the default policy simply selects an action uniformly at random at each node, that is, the sampled path proceeds to a uniformly random child at each step, until a leaf node is reached.

4. *Backpropagation:* Use the value of ℓ to update the evaluation of v and each of its ancestors. Specifically, for each such node v', increase N_v by 1 and Q_v by the value of ℓ.

The basic ideas behind MCTS are, therefore, quite simple. But the key to its success lies in the design of an effective tree policy. In particular, a good tree policy should strike a balance between exploration — in this case, learning more about the value of underexplored nodes — and exploitation — extracting more value from the most promising nodes.

To this end, Kocsis and Szepesvári [18] propose using the UCB1 policy [1], originally designed for the (stochastic) multi-armed bandit problem (UCB stands for "upper confidence bound"). In this problem, each arm is associated with an unknown distribution over rewards. Playing an arm

gives a reward drawn from its distribution, and also contributes an observation that can be used to better estimate potential rewards from that arm. The UCB1 policy pulls arm j with probability

$$\bar{X}_j + \sqrt{\frac{2 \ln r}{r_j}}, \tag{3}$$

where $\bar{X}_j$ is the average reward from arm j so far, r is the total number of observations of all arms, and r_j is the number of observations of arm j. The theoretical guarantees given by the UCB1 formula are beyond the scope of this chapter, but the high-level idea is clear: the first term in Equation (3) encourages exploitation, whereas the second term drives exploration — it is larger the less explored arm j is compared to other arms.

The variant of MCTS that uses UCB1 as the tree policy is known as the Upper Confidence Bounds for Trees (UCT) algorithm. Analogously to Equation (3), UCT's tree policy approaches the exploration vs. exploitation dilemma by selecting the child v' of v that maximizes

$$\frac{Q_{v'}}{N_{v'}} + C\sqrt{\frac{2 \ln N_v}{N_{v'}}}, \tag{4}$$

where $C > 0$ is a parameter that determines the priority given to exploration. $N_{v'} = 0$ is taken to mean that the value of the expression is ∞, that is, unexplored children are prioritized above explored children.

UCT ultimately selects the action at the root that maximizes $\bar{Q}_v$.[1] As the number of iterations of UCT grows, the evaluations $\bar{Q}_v$ become more accurate, and approach their true values. In particular, the *failure* probability at the root — the probability that a suboptimal action seems optimal — goes to zero. Specifically, Kocsis and Szepesvári [18, Theorem 6] show:

Theorem 3. *With an appropriate choice of C in Equation* (4)*, the failure probability at the root under UCT coverges to* 0 *at a polynomial rate (which depends on the maximum number of actions per state and the depth of the tree) as the number of iterations grows.*

To be a bit more precise, it could be the case that the UCB1 policy guides the search towards nodes that have no unexplored children, all the way down to a leaf node. In that case, the expansion and simulation steps do not take place; values are still updated via backpropagation. This is why $\bar{Q}_v$ is gradually refined even when the entire subtree rooted at v has been expanded.

2.2 Applying MCTS to Repeated SSGs

In order to understand the relation between repeated SSGs and MCTS, we present a slightly different formulation of the sequential decision making domain to which MCTS (specifically, UCT) is applied; this is, in fact, the formalism used by Kocsis and Szepesvári [18]. A *Markov Decision Process (MDP)* consists of states, available actions in each state, and a stochastic transition function T, where $T(s, a, s')$ is the probability of transitioning from state s to state s' when the

[1] There are alternative approaches, e.g., selecting the action that maximizes N_v.

action a is taken at state s. In addition, each possible transition is associated with a reward. We are interested in a *finite-horizon* MDP: the process ends after T steps, and the objective is to maximize the total accumulated reward.

An MDP with horizon T can be represented by a tree where every path from the root to a leaf is of length T. The reward at a leaf is simply the sum of rewards associated with the transitions along the path. Under this interpretation, the results of Section 2.1 — in particular, Theorem 3 — still hold.

Marecki et al. [21] represent a repeated SSG with a prior P over types and T rounds as an MDP with horizon T. Defender mixed strategies are discretized to ensure finite state and action spaces. Each state corresponds to the history of play so far, which is a sequence of pairs — the mixed strategy of the defender, and corresponding best response of the attacker. Given such a history h of observations, let Θ_h be the set of attacker types consistent with the history. Then the posterior distribution $\mathcal{D}_h$ over attacker types is given by

$$\mathcal{D}_h(\theta) = \begin{cases} \frac{\mathcal{D}(\theta)}{\sum_{\theta' \in \Theta_h} \mathcal{D}(\theta')} & \theta \in \Theta_h \\ 0 & \theta \notin \Theta_h \end{cases}$$

$\mathcal{D}_h$ straightforwardly determines the transition function: for action $\mathbf{p}$ and each $\theta \in \Theta$, the pair $(\mathbf{p}, b_\theta(\mathbf{p}))$ is appended to the history h with probability $\mathcal{D}_h(\theta)$. When this transition takes place, the reward $U_d(b_\theta(\mathbf{p}), \mathbf{p})$ is obtained.

Crucially, the UCT implementation of Marecki et al. [21] does not actually compute the posterior distributions at each step. Instead, in each iteration of the algorithm, they sample $\theta \sim \mathcal{D}$ upfront, and use θ to compute the attacker's best response at each step — this is equivalent to sampling the (unknown) posterior distribution at each step. In particular, Marecki et al. [21] obtain the following corollary of Theorem 3:

Corollary 4. *When the Marecki et al. [21] implementation of UCT is applied to a repeated SSG, the failure probability at the root converges to* 0 *at a polynomial rate (which depends on* T *and the number of discretized defender mixed strategies) as the number of iterations grows.*

The experiments of Marecki et al. [21] show that the approach scales well with respect to number of rounds T. The experiments also suggest that trees with a large branching factor — which is the product of the number of discretized mixed strategies available to the defender, and the number of targets — may be hard to deal with, but additional heuristics provide encouraging results.

3 Sequence of Attackers

The methods discussed in Sections 1 and 2 are designed to use repeated interactions with a *single* attacker to learn the missing payoff information. Settings involving *multiple* attackers, however, give rise to additional, fundamentally different sources of uncertainty. For example, it might not be possible to observe which attacker carried out the latest attack, or the distribution over attackers might be unknown or changing. In this section, we address some of these issues. In particular, we

describe a method introduced by Balcan et al. [5] for defending against an adversarial sequence of attackers, when attackers are chosen from a known set of types but with no distributional assumptions. Additionally, this method can be adapted to work even when the identity of the attacker remains unknown after the attack.

As in the previous section, consider a set of k known attacker types $\Theta = \{\theta_1, \ldots, \theta_k\}$ and U_{θ_i} for all $i \in [k]$, such that U_{θ_i} represents the utilities of attacker type θ_i. We assume that all utilities are scaled such that the defender's payoff is in the range $[-1, 1]$. Consider playing a repeated Stackelberg game for T rounds, such that at round t, an unknown attacker $a_t \in \Theta$ is chosen to attack the targets by best-responding to the mixed strategy of the defender at time t, denoted by $\mathbf{p}_t$. The defender's goal is to maximize his payoff over a period of time, even when the sequence of attackers is unknown. That is, without knowing the sequence ahead of the time, at each round the defender must commit to a mixed strategy based on the history of play so far. The defender then receives some "feedback" regarding the attacker at that round (either the attacker type or merely which target was attacked), and adjusts the mixed strategy for future rounds.

In more detail, in this *online* setting, the expected payoff of the defender over T time steps is

$$\mathbb{E}\left[\sum_{t=1}^{T} U_d(b_{a_t}(\mathbf{p}_t), \mathbf{p}_t)\right],$$

where the expectation is taken only over the internal randomness of the defender's online algorithm, and $b_{a_t}(\cdot)$ indicates the best response of attacker a_t. In contrast, if the sequence of attackers, or merely the frequency of each attacker type in the sequence, is known to the defender, the defender can pre-compute a fixed mixed strategy with the best payoff against that sequence of attackers and play it at every time step. Such a strategy is called *the best mixed strategy in hindsight* and denoted by

$$\mathbf{p}^* = \arg\max_{\mathbf{p}} \sum_{t=1}^{T} U_d(b_{a_t}(\mathbf{p}), \mathbf{p}).$$

The goal is to design online algorithms for the defender with payoff that is almost as good as the payoff of the best mixed strategy in hindsight. This difference in utilities is termed *regret* and is equal to

$$R_T = \sum_{t=1}^{T} U_d(b_{a_t}(\mathbf{p}^*), \mathbf{p}^*) - \mathbb{E}\left[\sum_{t=1}^{T} U_d(b_{a_t}(\mathbf{p}_t), \mathbf{p}_t)\right].$$

The results in this section show that, even when faced with this (relatively extreme) type of uncertainty, one can compete with the best fixed mixed strategy in hindsight. In particular, the algorithms introduced here are *no-regret algorithms*: the expected regret goes to zero as T goes to infinity. This indicates that the longer the algorithm runs the better the guarantees are in terms of the regret. In addition, the regret depends only polynomially on the number of types and targets. That is

$$R_T \leq o(T) \cdot \mathrm{poly}(n, k).$$

The feedback the defender receives at each time step plays a major role in the design of the algorithm. We consider two types of feedback, *full information* and *partial information*. In the full

information case, the identity of the attacker is revealed after each attack (e.g., through surveillance). An upper bound on the regret under full information feedback is as follows.

Theorem 5 (Balcan et al. [5])**.** *Given a repeated SSG with full information feedback, there is an algorithm that for any unknown sequence of attackers has regret*

$$R_T \leq O\left(\sqrt{Tn^2k\log nk}\right).$$

On the other hand, in the partial information case, even after the attack occurs the identity of the attacker remains unknown — only the attacked target is observed. Note that full information is indeed strictly more informative, as knowing the attacker a_t is sufficient to compute the target attacked at that round.

Theorem 6. *Given a repeated SSG with partial information feedback, there is an algorithm that for any unknown sequence of attackers has regret*

$$R_T \leq O\left(T^{2/3}\ nk\ \log^{1/3}(nk)\right).$$

In the remainder of this section, we first provide a brief overview of basic no-regret algorithms, and then apply these techniques to repeated SSGs.

3.1 Background on Regret Minimization

Consider the following online scenario. Suppose we have a set $\mathcal{M}$ of possible actions to take each day (e.g., a collection of different possible ways to get to work). At each time step t, we probabilistically select some action j_t according to a probability distribution of our choice $\mathbf{q}_t$ over the actions. An adversary (who may observe our previous actions $j_1, \ldots, j_{t-1}$, and may even know our current distribution $\mathbf{q}_t$ but does not get to observe our selection j_t from $\mathbf{q}_t$) simultaneously chooses a loss vector $\boldsymbol{\ell}_t$ over the actions, where let us assume all losses are in the range $[0, 1]$ (i.e., $\boldsymbol{\ell}_t \in [0, 1]^{|\mathcal{M}|}$). For instance, these might be the travel times of our different options for getting to work. We then incur an *expected loss* of $\mathbf{q}_t \cdot \boldsymbol{\ell}_t$.

Let $L_{alg,T} = \sum_{t=1}^{T} \mathbf{q}_t \cdot \boldsymbol{\ell}_t$ denote the expected total loss of the algorithm in T time steps, and let $L_{min,T} = \min_{j\in\mathcal{M}} \sum_{t=1}^{T} \boldsymbol{\ell}_t(j)$ denote the total loss of the best fixed action under the sequence of loss vectors $\boldsymbol{\ell}_1, \ldots, \boldsymbol{\ell}_T$. That is, $L_{min,T}$ is the loss of the best fixed action in hindsight. The difference, $R_T = L_{alg,T} - L_{min,T}$, is called the *regret* of the algorithm.

As noted above, algorithms with regret sublinear in T are often called *no-regret algorithms* because a sublinear regret implies that the average per-day cost of the algorithm is approaching (or even less than) the average per-day cost of the best fixed action in hindsight. That is, the *per-day regret* R_T/T is approaching 0. So, one does not regret very much having not known initially what the best fixed action in hindsight is.

Hannan in 1957 [14] was the first to develop algorithms with a no-regret guarantee. In the 1990s, motivated by machine learning settings in which $\mathcal{M}$ can be quite large (e.g., corresponding to all possible prediction rules in some class), algorithms were developed whose regret furthermore has only a logarithmic dependence on the size $|\mathcal{M}|$ of the action space. The following theorem formalizes these results.

Theorem 7 (e.g., [20])**.** *There is an efficient algorithm (running time per stage linear in $|\mathcal{M}|$) with the guarantee that for any adversary,*

$$R_T = O(\sqrt{T \log |\mathcal{M}|}).$$

There are by now many algorithms known to achieve the guarantees of Theorem 7. One canonical class of such algorithms are "multiplicative weights" or "randomized weighted majority" methods. Here is one such algorithm, called "polynomial weights" [10]:

Polynomial Weights algorithm (given an input $\epsilon \in (0, 1)$)

1. Initialize weights $\mathbf{w}(j) = 1$ for each $j \in \mathcal{M}$.
2. Choose distribution $\mathbf{q}_t$ proportional to $\mathbf{w}$. That is, $\mathbf{q}_t(j) = \mathbf{w}(j)/W$ where $W = \sum_i \mathbf{w}(i)$.
3. Given $\boldsymbol{\ell}_t$, update $\mathbf{w}(j) \leftarrow \mathbf{w}(j)(1 - \epsilon \boldsymbol{\ell}_t(j))$.

The intuition behind this algorithm is that actions that are far from optimal quickly become highly penalized and so the algorithm gives them negligible probability mass. At a formal level, at each step, the total weight in the system drops by an amount that is directly related to the expected loss of the algorithm; specifically, one can verify that at each step we have $W \leftarrow W(1 - \epsilon \mathbf{q}_t \cdot \boldsymbol{\ell}_t)$. On the other hand, any low-loss actions will have high weight (ensuring that W remains high), and so if there are any low-loss actions this means the algorithm's total expected loss cannot be too great. The overall guarantee is that $L_{alg,T} \leq (1 + \epsilon) L_{min,T} + \frac{\ln |\mathcal{M}|}{\epsilon}$ (in fact, the algorithm has a stronger guarantee but this suffices for our purposes here), and setting $\epsilon = \min(\sqrt{(\ln |\mathcal{M}|)/T}, 1/2)$ gives the bound of Theorem 7. For further discussion of this and related algorithms, see [8].

The discussion so far assumes the algorithm has the ability to observe the entire loss vector $\boldsymbol{\ell}_t$; indeed, this feedback is used in Step 3 of the Polynomial Weights algorithm. However, often one receives a more impoverished form of feedback. For example, one may learn only the loss $\boldsymbol{\ell}_t(j_t)$ of the action one specifically chose: this is called the (adversarial) *multi-armed bandit* problem [2]. Or one may receive partial information about multiple actions: e.g., in choosing a route to drive to work, one would learn about traffic congestion along the roads used in that route, which might provide partial information about other routes that also include some of those roads.

One way to tackle these partial information settings is to use the fact that algorithms for the full information setting really need only a *bounded unbiased estimate* of the loss of each action [2, 4]. For example, if one breaks the overall T time steps into windows, and within each window samples each action once at a random time, then these "exploratory samples" will give an unbiased estimate of the losses of each action in the window; this approach was introduced and analyzed by Awerbuch and Mansour [4]. In the case of actions that provide partial information, an improved method proposed by Awerbuch and Kleinberg [3] is to identify an "exploratory basis" $\mathcal{S} \subseteq \mathcal{M}$ of actions, whose feedback can be used to reconstruct unbiased estimates for all the actions, and then just perform exploration over those. For example, in the case of routes to drive to work, this might correspond to a small set of routes that together include all roads of interest. Using this approach and combining with bounds for the full information setting yields the following theorem (see [5] for details):

Theorem 8. *Suppose that $\mathcal{S} \subseteq \mathcal{M}$ is such that for any time-window τ, sampling the actions in $\mathcal{S}$ at random times in τ will produce estimates $\hat{\boldsymbol{\ell}}_\tau(j)$ for each $j \in \mathcal{M}$ such that:*

1. $\mathbb{E}[\hat{\boldsymbol{\ell}}_\tau(j)] = \boldsymbol{\ell}_\tau(j)$, *where* $\boldsymbol{\ell}_\tau(j) = \frac{1}{|\tau|}\sum_{t\in\tau} \boldsymbol{\ell}_t(j)$, *and*
2. $\boldsymbol{\ell}_\tau(j) \in [-\kappa, \kappa]$.

Then there is an algorithm with regret guarantee $R_T = O(T^{2/3}|\mathcal{S}|^{1/3}\kappa^{1/3}\log^{1/3}|\mathcal{M}|)$ for all $T > \kappa$.

3.2 Applying Regret Minimization Techniques to Repeated SSGs

To use the results from Section 3.1, we first need to define what constitutes actions and loss functions in our setting. Indeed, each valid coverage probability vector $\mathbf{p}$ in a security game corresponds to one action and the loss associated with this action at time t is simply the negation of its utility to the defender, $-U_d(b_{a_t}(\mathbf{p}), \mathbf{p})$. However, this creates an infinitely large set of actions, which renders the guarantees in Theorem 7 meaningless. Instead we show that a carefully chosen subset of the strategy space that is representative of the set of all strategies can be used with the algorithms from Section 3.1.

To this end, we first show that the set of attacker types, Θ, partitions the space of all strategies to convex regions where the attackers' best response remains fixed. As an extension to the definition of $\mathcal{P}_i$ from Section 1.2, let $\mathcal{P}_i^j$ indicate the set of all valid coverage probabilities where an attacker of type θ_j attacks target i. Then for a given set Θ and any function $\sigma : \Theta \to N$, let $\mathcal{P}_\sigma$ indicate the set of all valid coverage probability vectors such that for all $\theta_j \in \Theta$, θ_j attacks $\sigma(j)$. In other words, $\mathcal{P}_\sigma = \bigcap_{\theta_j\in\Theta} \mathcal{P}^j_{\sigma(j)}$. Since $\mathcal{P}_\sigma$ is the intersection of finitely many convex polytopes, it is itself a convex polytope. Figure 5 illustrates these regions.

Consider the set $\mathcal{E}$ of all extreme points of convex polytopes $\mathcal{P}_\sigma$ for all σ. It is easy to show that $|\mathcal{E}| \in O((2^n + kn^2)^{2n})$. Balcan et al. [5] show that a regret minimization algorithm incurs no additional loss when the set of actions is restricted to only those strategies that are in $\mathcal{E}$. Together with the bound on $|\mathcal{E}|$, the proof of Theorem 5 follows directly from Theorem 7.

Next, consider the partial information feedback model. As discussed in Section 3.1, we only need a mechanism to produce bounded unbiased estimators for the loss of all actions. As a naïve approach, first consider sampling each mixed strategy in $\mathcal{E}$ once at random in a window of time and observing its loss. The loss created this way is clearly bounded and unbiased. However, at every time step in which we sample a random strategy, we are left open to experiencing significant regret. As shown in Theorem 8, sampling each strategy individually once at random adds a regret that is polynomial in the number of mixed strategies sampled, which — based on the size of $|\mathcal{E}|$ — is *exponential* in the number of targets. Therefore a more refined mechanism for estimating the loss is needed.

Notice that the loss incurred by different mixed strategies is not independent, rather they all depend on the number of times each target is attacked in a window of time, which in turn depends on the frequency of attacker types. In order to estimate the type frequency, it is sufficient to distinguish between types based on their response to a number of mixed strategies. As an example, assume that there is a mixed strategy where each attacker type responds differently (See Figure 5).

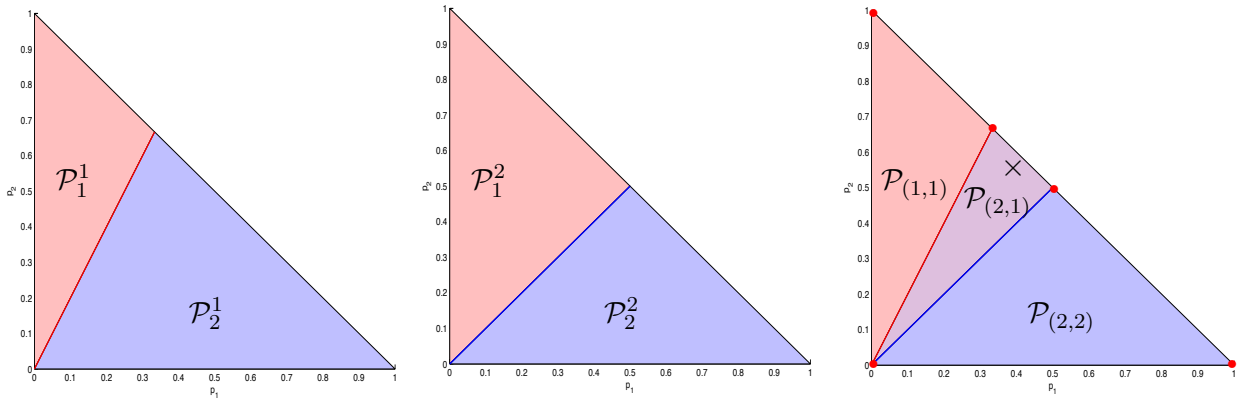


Figure 5: Best response regions in the space of coverage probability vectors. The first two figures define $\mathcal{P}_i^j$ in a game where one resource can cover one of two targets, and with two attacker types. The third figure illustrates $\mathcal{P}_\sigma$ for the intersection of the best response regions of the two attackers. The red points indicate the extreme points, and the black cross is a coverage probability vector where the two attackers respond differently.

Then sampling such a mixed strategy even once at random in a window of time allows us to construct an unbiased estimator of the frequency of all attacker types, and, as a result, create an unbiased estimator of the loss of all actions. In general, such a mixed strategy, where each attacker responds differently, may not exists. Instead, Balcan et al. [5] show how to choose k mixed strategies that construct a loss estimator for all actions, such that the loss is in the range $[-nk, nk]$. The guarantee of Theorem 6 then follows from Theorem 8. We leave it to the interested reader to refer to [5] for the details of this procedure.

4 Further Reading

In this chapter we have focused on a single, relatively cohesive, line of work on learning in SSGs. A number of other learning-based approaches have been proposed to deal with uncertainty in SSGs. Below we briefly outline some of them.

A fundamentally different approach for handling payoff uncertainty assumes that the utilities $u_a^c(i)$ and $u_a^u(i)$ — the utility to the attacker when target i is attacked and it is covered and uncovered, respectively — are not exactly known, but lie in a known interval [16]. Given such a set of uncertainty intervals, two per target, it is possible to reason about the *maximum regret* of the defender strategy $\mathbf{p}$: it is the maximum difference between the defender's utility when playing the optimal strategy and when playing $\mathbf{p}$, where the maximum is taken over all possible utility functions that are consistent with the given uncertainty intervals. From this viewpoint, the defender's strategy should minimize the maximum regret.

Nguyen et al. [22] reduce minimax regret by eliciting information about utilities. This is done via *bound queries*, which ask an expert (say, a risk analyst) whether a given utility is above or below a given threshold. Nguyen et al. introduce three heuristic elicitation strategies, which aim

to achieve the best possible minimax regret under a query budget.

Another type of uncertainty that can be addressed using machine learning is uncertainty about the attacker's behavioral model. We have focused on perfectly rational attackers who best-respond to the defender's strategy, but studies suggest that more nuanced behavioral models give better predictions [23].

Yang et al. [25] adopt the *subjective utility quantal response* model [23], which stipulates that the attacker's probability of choosing an action increases with his *subjective utility* for that action. The subjective utility is induced by three parameters that weight the relative importance of p_i, $u_a^c(i)$, and $u_a^u(i)$. Yang et al. learn a 3-dimensional normal distribution of these parameters (that is, they estimate its mean and covariance matrix).

Finally, we mention work by Chakraborty et al. [11], which deals with learning to play against memory-bounded agents. While their framework is more general, let us describe its application to mass transit security. The goal in this domain is to deter passengers who may not buy a ticket, by deploying patrol officers. Each passenger has a fixed route, so the passenger's decision amounts to whether to buy a ticket or not; a risk neutral passenger would buy a ticket if and only if the expected cost of not buying a ticket (probability of getting caught times fine) is larger than the fare. A passenger estimates his probability of being caught based on patrol strategies in the last L days, where L can be different for different passengers. Given the number of passengers on each route, but not the distribution over their memory size L (which is assumed to be in $\{2, 3, 4\}$ in the experiments), Chakraborty et al. [11] use their TOMMBA algorithm to play against these forgetful passengers.

Acknowledgments

The authors thank Leandro Soriano Marcolino for helpful feedback. The authors were partially supported by the NSF under grants IIS-1350598, CCF-1215883, CCF-1116892, IIS-1065251, and CCF-1415460, and by a Sloan Research Fellowship.

References

[1] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2):235–256, 2002.

[2] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire. The nonstochastic multiarmed bandit problem. *SIAM Journal on Computing*, 32(1):48–77, 2002.

[3] B. Awerbuch and R. Kleinberg. Online linear optimization and adaptive routing. *Journal of Computer and System Sciences*, 74(1):97–114, 2008.

[4] B. Awerbuch and Y. Mansour. Adapting to a reliable network path. In *Proceedings of the 22nd ACM Symposium on Principles of Distributed Computing (PODC)*, pages 360–367, 2003.

[5] M.-F. Balcan, A. Blum, N. Haghtalab, and A. D. Procaccia. Commitment without regrets: Online learning in Stackelberg security games. In *Proceedings of the 16th ACM Conference on Economics and Computation (EC)*, 2015. Forthcoming.

[6] D. Bertsimas and S. Vempala. Solving convex programs by random walks. *Journal of the ACM*, 51(4):540–556, 2004.

[7] A. Blum, N. Haghtalab, and A. D. Procaccia. Learning optimal commitment to overcome insecurity. In *Proceedings of the 28th Annual Conference on Neural Information Processing Systems (NIPS)*, pages 1826–1834, 2014.

[8] A. Blum and Y. Mansour. Learning, regret minimization, and equilibria. In N. Nisan, T. Roughgarden, É. Tardos, and V. Vazirani, editors, *Algorithmic Game Theory*, chapter 4. Cambridge University Press, 2007.

[9] C. Browne, E. J. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, 2012.

[10] N. Cesa-Bianchi, Y. Mansour, and G. Stoltz. Improved second-order bounds for prediction with expert advice. In *Proceedings of the 18th Conference on Computational Learning Theory (COLT)*, pages 217–232, 2005.

[11] D. Chakraborty, N. Agmon, and P. Stone. Targeted opponent modeling of memory-bounded agents. In *Proceedings of the Adaptive Learning Agents Workshop (ALA)*, 2013.

[12] V. Conitzer and T. Sandholm. Computing the optimal strategy to commit to. In *Proceedings of the 7th ACM Conference on Economics and Computation (EC)*, pages 82–90, 2006.

[13] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer, 1988.

[14] J. Hannan. Approximation to Bayes risk in repeated plays. In M. Dresher, A. Tucker, and P. Wolfe, editors, *Contributions to the Theory of Games*, volume 3, pages 97–139. Princeton University Press, 1957.

[15] L.G. Khachiyan. A polynomial algorithm in linear programming. *Soviet Mathematics Doklady*, 20:191–194, 1979.

[16] C. Kiekintveld, T. Islam, and V. Kreinovich. Security games with interval uncertainty. In *Proceedings of the 12th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 231–238, 2013.

[17] C. Kiekintveld and M. Jain. Basic solution concepts and algorithms for Stackelberg security games. In Ali Abbas, Milind Tambe, and Detlof von Winterfeldt, editors, *Improving Homeland Security Decisions*, chapter 17. 2015.

[18] L. Kocsis and C. Szepesvári. Bandit based Monte-Carlo planning. In *Proceedings of the 17th European Conference on Machine Learning (ECML)*, pages 282–293, 2006.

[19] J. Letchford, V. Conitzer, and K. Munagala. Learning and approximating the optimal strategy to commit to. In *Proceedings of the 2nd International Symposium on Algorithmic Game Theory (SAGT)*, pages 250–262, 2009.

[20] N. Littlestone and M. K. Warmuth. The weighted majority algorithm. *Information and Computation*, pages 212–261, 1994.

[21] J. Marecki, G. Tesauro, and R. Segal. Playing repeated Stackelberg games with unknown opponents. In *Proceedings of the 11th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 821–828, 2012.

[22] T. H. Nguyen, A. Yadav, B. An, M. Tambe, and C. Boutilier. Regret-based optimization and preference elicitation for Stackelberg security games with uncertainty. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI)*, pages 756–762, 2014.

[23] T. H. Nguyen, R. Yang, A. Azaria, S. Kraus, and M. Tambe. Analyzing the effectiveness of adversary modeling in security games. In *Proceedings of the 27th AAAI Conference on Artificial Intelligence (AAAI)*, pages 718–724, 2013.

[24] A. Tauman Kalai and S. Vempala. Simulated annealing for convex optimization. *Mathematics of Operations Research*, 31(2):253–266, 2006.

[25] R. Yang, B. J. Ford, M. Tambe, and A. Lemieux. Adaptive resource allocation for wildlife protection against illegal poachers. In *Proceedings of the 13th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 453–460, 2014.