Which Security Games are Hard to Solve?

Manish Jain*, Kevin Leyton-Brown+, Milind Tambe*
* Computer Science Department
University of Southern California
Los Angeles, CA. USA. 90089.
+ Department of Computer Science
University of British Columbia
Vancouver, B.C., Canada. V6T IZ4

Abstract

Stackelberg security games form the backbone of systems like ARMOR, IRIS and PROTECT, which are in regular use by the Los Angeles International Police, US Federal Air Marshal Service and the US Coast Guard respectively. An understanding of the runtime required by algorithms that power such systems is critical to furthering the application of game theory to other real-world domains. This paper identifies the concept of the *deployment-to-saturation ratio* in random Stackelberg security games, and shows that in a decision problem related to these games, the probability that a solution exists exhibits a phase transition as the ratio crosses 0.5. We demonstrate that this phase transition is invariant to changes both in the domain and the domain representation. Moreover, problem instances at this phase transition point are computationally harder than instances with other deployment-to-saturation ratios for a wide range of different equilibrium computation methods, including (i) previously published different MIP algorithms, and (ii) different underlying solvers and solution mechanisms. Our findings have at least two important implications. First, it is important for new algorithms to be evaluated on the hardest problem instances. We show that this has often not been done in the past, and introduce a publicly available benchmark suite to facilitate such comparisons. Second, we provide evidence that this phase transition region is also one where optimization would be of most benefit to security agencies, and thus requires significant attention from researchers in this area.

Introduction

Software security assistants built on the framework of Stackelberg security games (Kiekintveld et al. 2009) have been deployed by a variety of real-world security agencies. For example, ARMOR (Jain et al. 2010b) has been in use by the police at Los Angeles International Airport since 2007. Similarly, IRIS (Jain et al. 2010b) and PROTECT (An et al. 2011) have been in use by the US Federal Air Marshals Service and the US Coast Guard since 2009 and 2011 respectively. Many different algorithms have been proposed for computing solutions to such problems (Conitzer and Sandholm 2006; Paruchuri et al. 2008; Gatti 2008; Kiekintveld et al. 2009; Jain et al. 2010a; Dickerson et al. 2010; Letchford and Vorobeychik 2011; Bosansky et al. 2011) with a focus on scalability to enable the application of these models to newer and more complex real-world domains. In this paper, we ask what structural properties make such Stackelberg security game instances hard to solve in practice.

Perhaps the most influential work on understanding algorithm-independent structural properties of a combinatorial problem is the line of work on phase transitions in uniform-random 3-SAT, pioneered by Cheeseman, Kanefsky, and Taylor (1991) and Mitchell, Selman, and Levesque (1992). This work showed that the probability that a 3-SAT instance will be satisfiable exhibits a phase transition when the number of variables is fixed and the number of clauses increases. Specifically, this probability starts near 1 for small numbers of clauses, sharply falls almost to 0 as the clauses-to-variables ratio crosses 4.26 (Crawford and Auton 1996), and then remains near 0 as the number of clauses grows. This phase transition in random 3-SAT instances is important because it correlates very strongly with computational hardness: the hardest problem instances correspond to the point where the probability of satisfiability is 0.5 (Mitchell, Selman, and Levesque 1992; Larrabee and Tsuji 1993; Selman, Mitchell, and Levesque 1996; Cook and Mitchell 1997). Phase transitions have also been used to analyze the computational impact of problem structure in optimization problems, such as MAX-SAT (Slaney and Walsh 2002) and TSP (Gent and Walsh 1996; Frank, Gent, and Walsh 1998). The approach taken in this work is to identify a phase transition in a decision version of the optimization problem (i.e., asking whether or not a solution exists with a given objective function value).

Our own work is concerned with an optimization problem faced by security forces. Like this previous work, we identify a phase transition in the corresponding decision problem to understand the underlying properties of the security domains. More specifically, we introduce the concept of the deployment-to-saturation (d:s) ratio, show that it exhibits a phase transition at 0.5 for random Stackelberg security game instances, and show that the hardest such instances arise at this point. The d:s ratio is the number of deployed defender resources divided by the number of resources beyond which additional resources do not provide any benefit to the defender. We show that the phase transition at the d:s ratio of 0.5 is independent of the domain representation, model and

solver. We provide evidence for this phase transition and correspondingly hardest instances of Stackelberg security games from three different classes of security domains, eight different MIP algorithms (including two algorithms in the real world), five different underlying MIP solvers, two different equilibrium concepts in Stackelberg security games, and a variety of domain sizes and conditions.

We discuss two important implications of these findings. First, new algorithms should be compared on the hardest problem instances; we show that most previous research has compared the runtime performance of algorithms only at low d:s ratios, where problems are comparatively easy. Second, we argue that this phase transition region is the point where optimization has the greatest benefit to security agencies, implying that problems in this region deserve increased attention from researchers.

Other Related Work

Research on identifying underlying structures of optimization problems has used insights from corresponding decision versions of the optimization problem (Slaney and Walsh 2002; Gent and Walsh 1996) . Monasson et. al (1999) study computational hardness for MAX 2+p-SAT, since it interpolates smoothly from the polynomial 2-SAT to the NP-complete 3-SAT. They showed that optimization problems have their own structure and identified relationships between average clause length, constrainedness and hardness. Thereafter, research moved towards understanding sources of empirical hardness beyond the clauses-to-variable ratio (Nudelman et al. 2004) and to graph-based NP-complete problems (Frank and Martel 1995; Frank, Gent, and Walsh 1998). Gent et al. (1996) studied the properties of four different optimization problems including TSP and Boolean circuit synthesis. They compute the optimal tour length from the TSP optimization problem, and then use this value to define a corresponding TSP decision problem. A phase transition in the solubility of this problem corresponds to the computationally hardest region.

In the context of game theory, phase transitions in the probability of cooperation have been studied for evolutionary game theory (Hauert and Szab 2005). Phase transitions have also been used to analyze the efficiency of markets in adaptive games (Savit, Manuca, and Riolo 1999). To our knowledge, our work is the first that identifies such structural properties in the context of Stackelberg security games.

Stackelberg Security Games

Stackelberg security games are played between a defender and an attacker, and conform to a leader–follower paradigm (Conitzer and Sandholm 2006; Paruchuri et al. 2008; Kiekintveld et al. 2009). The defender first commits to a mixed strategy, to which the attacker then responds. The actions of the defender correspond to different security measures she can undertake, and the action space of the attacker is a choice of the targets to attack. With each target four payoff values are associated: reward and penalty to both the defender and the attacker for an unsuccessful and successful attack. Table 1 shows an example Stackelberg security game with 2 targets. The defender is the row player, and the attacker is the column player. In this example, the defender would get a payoff of 5 if she chose to *cover* (i.e., protect) Target 1 and the attacker did attack Target 1. The solution concept of choice in security games is the strong Stackelberg equilibrium (SSE); it is described formally e.g., by Kiekintveld et al. (2009). SSE defines the optimization problem we focus on in this paper: finding a mixed strategy for the defender that maximizes her expected utility, given that the attacker best responds to this mixed strategy. As previously mentioned, a large number of algorithms are being designed for these games, and software assistants built on these frameworks have been successfully deployed (Jain et al. 2010b; An et al. 2011).

	Target 1	Target 2
Target 1	$5, -5$	$-1, 3$
Target 2	$-6, 3$	$1, -5$

Figure 1: Example security game with two targets.

A Bayesian Stackelberg security game extends the framework to multiple types of attackers, with each such type identified by its own payoff matrix. The defender does not know which attacker type she will face in a given instance, but knows the probability distribution from which the attacker's type is drawn.

Phase Transitions in Security Games

In this section, we demonstrate the existence of a phase transition in a decision version of the SSE optimization problem at d:$s = 0.5$, and also show that this point corresponds to the hardest random instances for a range of problem formulations and algorithms. We begin by defining the decision version of the SSE optimization problem, which we denote SSE(D). SSE(D) asks whether there exists a defender strategy that guarantees expected utility of at least the given value D.

The deployment-to-saturation (d:s) ratio is defined in terms of *defender resources*, a concept whose precise definition differs from one domain to another. Given such a definition, *deployment* refers to the number of defender resources available to be allocated, and *saturation* refers to the minimum number of defender resources such that the addition of further resources beyond this point yields no increase in the defender's expected utility.

We want to claim that a phase transition in the decision problem correlates with the hardest random problem instances. However, we obtain different phase transitions for different values of D. We define D^* as the median objective function value achieved in the SSE optimization problem when the d:s ratio is set to 0.5. We can estimate D^* by sampling random problem instances at d:$s = 0.5$, and computing the sample median of their objective function values; denote such an estimate $\widehat{D^*}$. In our experiments, we computed $\widehat{D^*}$ by taking 100 samples.

Claim 1 *As the d:s ratio varies from 0 to 1, the probability p that a solution exists to SSE(D^*) exhibits a phase transition*

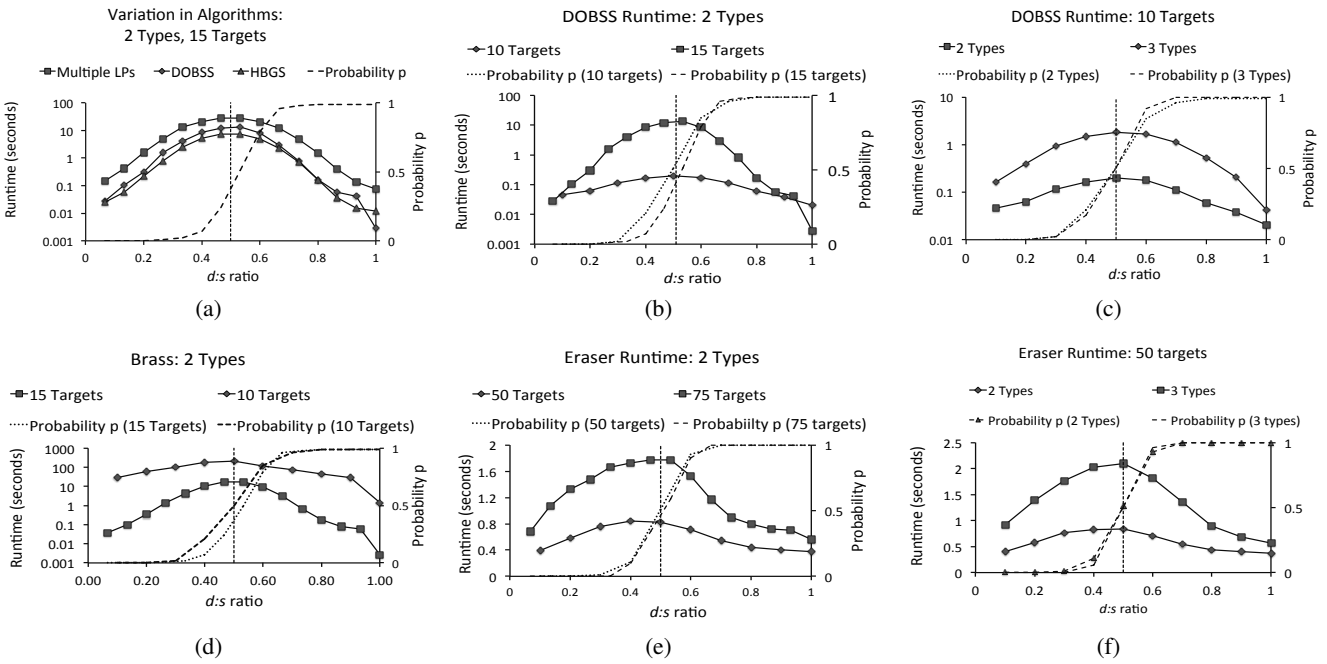


Figure 2: Average runtime of computing the optimal solution for a SPNSC problem instance, along with the probability p that the corresponding decision problem is solvable. The vertical dotted line shows $d{:}s = 0.5$.

at $d{:}s = 0.5$. This phase transition is independent of the security domain or its representation. Furthermore, the computationally hardest problem instances are those for which $d{:}s = 0.5$, independent of the representation or solver.

To support this claim, we show results for three different classes of security domains, including results using eight different MIP algorithms (including two deployed algorithms), five different underlying MIP solvers, two different equilibrium concepts in Stackelberg security games, and a variety of domain sizes and conditions. All the results shown below are averaged over 100 samples, and were collected on a machine with a 2.7GHz Intel Core i7 processor and 8GB main memory. In all graphs, the x-axis shows the $d{:}s$ ratio, the solid lines show the runtime, and the probability p is drawn with dashed lines. Experiments were conducted using CPLEX 12.2 unless otherwise noted.

SPNSC Domain

SPNSC (Security Problems with No Scheduling Constraints) refers to a security domain with distinct targets, a set of defender resources and many attacker types. Each defender resource can defend any one target, and there are no scheduling constraints on the defender. In this domain, the $d{:}s$ ratio is the ratio of the number of available defender resources to the number of targets. An real-world application of SPNSC is setting up vehicular inspection checkpoints at the Los Angeles International Airport, as implemented in ARMOR (Jain et al. 2010b). We now examine two different representations of the SPNSC domain.

General sum representation This representation models every possible combination of actions of the defender's resources as a pure strategy for the defender. Thus, the number of defender pure strategies for this domain is exponential in the number of defender resources. Three algorithms, MultipleLPs (Conitzer and Sandholm 2006), DOBSS (Paruchuri et al. 2008) and HBGS (Jain, Kiekintveld, and Tambe 2011), have been proposed to compute SSE for this representation.

We conducted experiments varying the algorithm, number of attacker types and number of targets. The results are plotted in Figures 2(a), 2(b) and 2(c). The payoffs for the two players were selected uniformly at random: the rewards for success were selected from the range $[1, 10]$, and the penalty for failure was picked from the range $[-10, -1]$.

Figure 2(a) shows that the runtime required by all three algorithms peaks at $d{:}s = 0.53$. (While we would like to have observed peaks at exactly $d{:}s = 0.5$ in all of our experiments, we typically observed values that were slightly different. The explanation that might come first to mind is variance due to having measured an insufficient number of samples; indeed, that does explain noise we observed in our measurement of p. However, there is also a more critical issue, which indeed arises here. Because our numbers of resources and targets are discrete, we were not able to measure every $d{:}s$ value. Here, 8 resources and 15 targets corresponded to $d{:}s = 0.53$.) This set of experiments considered 2 attacker types and 15 targets. The graph shows a phase transition in p, with values spiking from 0 to 1 within a narrow range of $d{:}s$ values. The value of p at the $d{:}s$ ratio of 0.53 was 0.51. (Of course, we could not measure p at exactly $d{:}s = 0.5$ either. The vertical line in Figure 2(a) really is at $d{:}s = 0.5$. However, because the dashed line representing p interpolates between datapoints we actually observed, the crossing point between the vertical line and the dashed

line is imprecise.) Notice that our calculation of p is a measurement about the domain itself which is independent of any algorithm or solver. Moreover, all the three algorithms required the maximum runtime when the d:s ratio was 0.53. For example, MultipleLPs required 27.9 seconds to compute the optimal solution. We learn from this experiment that (i) the computation is hardest when the d:s ratio is about 0.5, and (ii) there is a phase transition in p that lines up with these hardest instances.

The next two experiments study the phase transition when the number of targets and the number of types in the domain vary. Figure 2(b) varies the number of targets in the domain from 10 to 15. Similarly, Figure 2(c) varies the number of types from 2 to 3 in the security domain. For example, Dobss took 1.8 seconds on average for instances with 3 attacker types at the d:s ratio of 0.5 (5 resources and 10 targets), As with the previous results, the d:s ratio of 0.5 corresponds with $p = 0.51$ as well as the computationally hardest instances.

We also ran experiments with BRASS, which computes an ϵ-Stackelberg equilibrium (Pita et al. 2010). We defined ϵ-SSE(D) in the obvious way, and again varied d:s. In Figure 2(d), we show that a phase transition also exists for the ϵ-Stackelberg equilibrium solution concept. The hardest instances for BRASS in a domain with 15 targets corresponded to a d:s ratio of 0.53 (8 resources), with BRASS taking 17 seconds. As hypothesized, the corresponding probability p was 0.50. Thus, the presence of the phase transition is independent of the solution concept as well.

Security game compact representation This representation computes the probability of the defender protecting a target (Kiekintveld et al. 2009). It does not model joint distributions of multiple defender resources, but only operates on the marginal probability of the defender covering a target. ERASER (Kiekintveld et al. 2009) is the only algorithm for this compact representation that computes optimal solution for Bayesian Stackelberg security games.

We conducted experiments with ERASER that varied the number of targets and the number of attacker types. We generated payoffs for the players as before. Figure 2(e) shows our results for problem instances with 2 attacker types and both 50 and 75 targets. For example, the runtime required for 75 targets for the d:s ratio of 0.49 (37 resources) was 1.8 seconds, while the probability p was 0.48. This was the computationally hardest point for 75 targets. This experiment again shows the presence of a phase transition in p at the d:s ratio of 0.5, and that the computationally hardest instances occur when p is equal to 0.5. We also varied the number of types, and those results also confirmed our claim. These results can be found in Figure 2(f). This figure also shows a phase transition in p at the d:s ratio of 0.5, and it corresponds to the computationally hardest instances.

Our next experiment investigated the effect on ERASER's runtime of changing its underlying solver and solution mechanism. In Figure 3, we plot the runtime required by ERASER with CPLEX Primal Simplex, CPLEX Dual Simplex, CPLEX Network Simplex, CPLEX Barrier and GLPK Simplex methods. Again, we generated the payoffs and game instances as before. A sample result is that, for the d:s ratio of 0.50 (25 targets), the runtime required by CPLEX Dual Simplex was 0.9 seconds and the runtime required by GLPK Simplex was 1.6 seconds. This was the computationally hardest point for all solver/solution methods and it corresponded to $p = 0.51$. These experiments again show the phase transition and its correspondence with the computationally hardest instances hold across a range of underlying solver and solution mechanisms for the SPNSC domain.

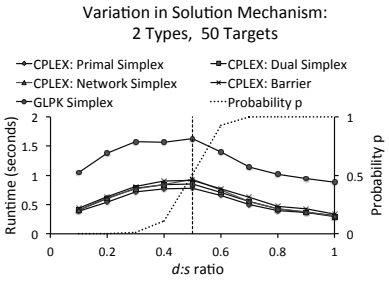


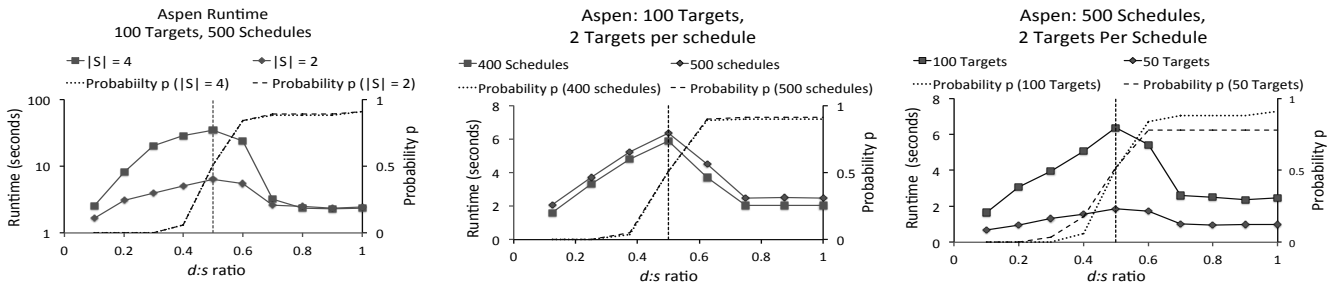
Figure 3: The figure shows the average runtime of computing the optimal solution using Eraser in a domain with 50 targets, 2 attacker types with different solvers and solution mechanisms, along with the probability of finding a solution of the decision problem.

SPARS Domain

The SPARS (Security Problems with ARbitrary Schedules) domain (Jain et al. 2010a) models a setting in which defender resources may not be homogeneous, and are required to satisfy scheduling constraints. A practical example is the scheduling problem faced by the US Federal Air Marshals Service, where every air marshal is a defender resource who has to obey spatio-temporal and logistical constraints in selecting flight tours. In a SPARS problem instance, a resource (e.g., an air marshal) selects a schedule (e.g., a flight tour), where each schedule can cover multiple targets (e.g., flight tour spans multiple flights). Thus, each resource is capable of protecting multiple targets. ASPEN (Jain et al. 2010a) is the only algorithm that has been proposed to compute optimal solutions for SPARS instances; it is based on a branch-and-price algorithm (Barnhart et al. 1994).

We conducted experiments varying the length of a schedule, the number of targets and the number of schedules. As before, payoffs for the two players were selected uniformly at random; the rewards for success were selected uniformly at random from the interval $[1, 10]$, and the penalty for failure uniformly at random from the interval $[-10, -1]$.

Figure 4(a) shows our results for SPARS problem instances with 100 targets and 500 schedules, when the number of targets covered by each schedule varied. For example, the runtime required for $|S| = 2$ at the d:s ratio of 0.50 (20 deployed resources, 40 required for saturation) was 6.4 seconds, with a corresponding $p = 0.51$. This was computationally the hardest point for $|S| = 2$. For $|S| = 4$, the computationally hardest point required 28.9 seconds at



(a) (b) (c)

Figure 4: Average runtime of computing the optimal solution for a SPARS game using ASPEN, along with the probability p. The vertical dotted line shows $d{:}s = 0.5$. Graphs with multiple data series also have multiple vertical lines; however, these tend to overlap, and so appear to be a single line.

$d{:}s = 0.5$ (10 available resources, 20 required for saturation), and again corresponded to a probability of $p = 0.51$. (Note that throughout Figure 4, the highest probability is strictly less than 1. This is because some of the decision problems were infeasible, due to a constraint requiring that multiple resources not cover the same target.)

Figure 4(b) shows the results for SPARS problem instances with 100 targets and 2 targets per schedule, considering 400 and 500 schedules. For example, the runtime required for 500 schedules for the $d{:}s$ ratio of 0.50 (20 resources, 40 required for saturation) was 6.4 seconds, while the probability p was 0.51. Figure 4(c) shows the results for SPARS problem instances with 500 schedules, 2 targets per schedule for 50 and 100 targets. For example, the runtime required for 50 targets for the $d{:}s$ ratio of 0.50 (10 resources, 20 required for saturation) was 1.9 milliseconds, while the probability p was 0.52. As expected, $d{:}s = 0.5$ corresponded to $p = 0.5$, and the computationally hardest instances in both experiments.

SPPC Domain

We now introduce a new domain: Security Problems with Patrolling Constraints (SPPC). This is a generalized security domain that allows us to consider many different facets of the patrolling problem. The defender needs to protect a set of targets, located geographically on a plane, using a limited number of resources. These resources start at a given target and then conduct a tour that can cover an arbitrary number of additional targets; the constraint is that the total tour length must not exceed a given parameter L. We consider two variants of this domain featuring different attacker models.

1. There are multiple independent attackers, and each target can be attacked by a separate attacker. Each attacker can learn the probability that the defender protects a given target, and can then decide whether or not to attack it.
2. There is a single attacker with many types, modeled as a Bayesian game. The defender does not know the type of attacker she faces. The attacker attacks a single target.

These variants were designed to capture properties of patrolling problems studied by researchers across many real-world domains (An et al. 2011; Bosansky et al. 2011; Vanek et al. 2011). An example for the Bayesian single attacker setting is the US Coast Guard patrolling a set of targets along the port to protect against potential threats. The defender's objective is to find the optimal mixed strategy over tours for all its resources in order to maximize her expected utility. In this case, the deployment-to-saturation ratio corresponds to the ratio between the allowed tour length and the minimum tour length required to cover all targets with the given number of defender resources.

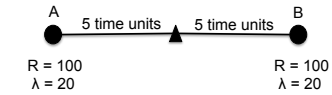
Payoff Structure With each target in the domain are associated payoffs, which specify the payoff to both the defender and the attacker in case of an successful or an unsuccessful attack. The attacker pays a high penalty for getting caught, where as the defender gets a reward for catching the attacker. On the other hand, if the attacker succeeds, the attacker gets a reward where as the defender pays a penalty. Both the players get a payoff of 0 if the attacker chooses not to attack. The payoff matrix for each target is given in Table 1. Thus, the defender gets a reward of τ_s units if she succeeds in protecting the attack on target s, i.e. if the defender is *covering* the target s when it is attacked. On the other hand, the attacker pays a penalty of $-P$ on being caught. Similarly, the reward to the attacker is R_s for a successful attack on site s, whereas the corresponding penalty to the defender for leaving the target *uncovered* is $-R_s$.

	No Attack	Attack
Covered	$0, 0$	$\tau_s, -P$
Uncovered	$0, 0$	$-R_s, R_s$

Table 1: Payoff structure for each target: defender gets a reward of τ_s units for successfully preventing an attack, while the attackers pays a penalty $-P$. Similarly, on a successful attack, the attacker gains R_s and the defender loses $-R_s$. Both players get 0 in case there is no attack.

Multiple Attackers In this game model, there are as many attackers as the number of targets in the domain. Each attacker can choose to attack or not attack a distinct target. Each attacker can observe the net *coverage*, or probability of the target being on a defender's patrol, for the target that

the attacker is interested in. In our formulation, we assume that the attackers are independent and do not coordinate or compete. Figure 5 shows an example problem and solutions for this example. There are just two targets, A and B, which are placed 5 units away from the home (starting) location of the defender's resources. There are two attackers, one for each target. The tour length allowed in this example was 10 units, that is, the defender can only patrol exactly one target in each patrol route. The penalty P was set to 70 units where as the reward R for a successful attack to the attacker was 100 units. For this particular example, the defender cannot protect the attacks on both sites and the optimal defender strategy is to cover one target with probability 0.588, cover the other target with probability 0.412 with the optimal defender reward being -50.588.



Attacker Penalty P = 70
Time Bound = 10 time units
Number of inspectors = 1
Inspector Reward = -50.588

Inspector Strategy

A	0.588
B	0.412

Figure 5: Example 1

Solution Methodology: We propose a branch and price based formulation to compute optimal defender strategies in this domain. Branch and price is a framework for solving very large mixed integer optimization problems that combines branch and bound search with column generation. Branch and bound search is used to address the integer variables: each branch sets the values for some integer variable, whereas column generation is used to scale up the computation to very large input problems.

There is a binary variable associated with each attacker: either an attacker chooses to attack or he does not. Binary variables are non-linear and are a well-known challenge for optimization. This challenge is handled using a branch and bound tree, where each branch of this tree assigns a specific value to each attacker variable. Thus, each leaf of this tree assigns a value for every attacker, that is, for every binary variable.

Column Generation: Column generation is used to solve each node of the above branch and bound tree. The problem at each leaf is formulated as a linear program, which is then decomposed into a *Master* problem and a *Slave* problem. The master solves for the defender strategy $\mathbf{x}$, given a restricted set of tours T. The objective function for the slave is updated based on the solution of the master, and the slave is solved to identify the best new column to add to the master problem, using reduced costs (explained later). If no tour can improve the solution further, the column generation procedure terminates.

Master Formulation: The master problem solves for the best probability distribution $\mathbf{x}$ that maximizes the defender's expected utility given a limited number of patrol tours $\mathbf{T}$. The defender's expected utility is a sum of defender utilities d_s over all the targets s. The master formulation is given in Equations (1) to (7). The notation is described in Table 2. Equations (3) and (4) capture the payoff the defender. They ensure that d_s is upper bounded by the payoff to the defender at target s, Equation (3) capturing the payoff when the attacker chooses to attack s (i.e. $q_s = 1$) whereas (4) captures the defender's payoff when the attacker chooses to not attack s (i.e. $q_s = 0$). Similarly, Equations (5) and (6) capture the payoff of the attacker. They ensure that the assignment $q_s = 1$ is feasible if and only if the payoff to the attacker for attacking the target s, $(\sum_{t \in T} x_t z_{st})(-P - R_s) + R_s$, is greater than 0, the attacker's payoff for not attacking target s. Equations (2) and (7) ensure that the strategy $\mathbf{x}$ is a valid probability distribution.

Variable	Definition
$\mathbf{S}$	Set of sites (targets)
$\mathbf{T}$	Set of tours
L	Upper bound on the length of a defender tour
$\mathbf{x}$	Probability distribution over $\mathbf{T}$
$\mathbf{q}$	Attack vector
z_{st}	Binary value indicating whether or not $s \in T$
d	Defender reward
k	Adversary reward
P	Penalty to attacker
R_s	Reward to attacker at site s
τ_s	Defender reward for catching attacker on site s
M	Huge Positive constant

Table 2: Notation

$$\min_{x,y,d,q} \quad -\sum_{s \in S} d_s \tag{1}$$

$$\text{s.t.} \quad \sum_{\mathrm{t} \in \mathrm{T}} \mathrm{x_t} \leq 1 \tag{2}$$

$$d_s - \sum_{t \in T} x_t z_{st}(\tau_s + R_s) + M q_s \leq M - R_s \tag{3}$$

$$d_s - M q_s \leq 0 \tag{4}$$

$$M q_s (P + R_s) + \sum_{t \in T} x_t z_{st}(P + R_s) \leq M + R_s \tag{5}$$

$$-M q_s (P + R_s) - \sum_{t \in T} x_t z_{st}(P + R_s) \leq -R_s \tag{6}$$

$$x_t \in [0, 1] \tag{7}$$

Slave Formulation: The slave problem find the best patrol tour to add to the current set of tours $\mathbf{T}$. This is done using reduced cost, which captures the total change in the defender payoff if a tour is added to the set of tours $\mathbf{T}$. The candidate tour with the minimum reduced cost improves the objective value the most (**?**). The reduced cost $\bar{c}_t$ of variable x_t, associated with tour T , is given in Equation 8, where w, y, v and h are dual variables of master constraints (3), (5), (6)

and (2) respectively. The dual variable measures the influence of the associated constraint on the objective, and can be calculated using standard techniques:

$$\overline{c}_t = \sum_{s \in S} (w_s(\tau_s + R_s) + (v_s - y_s)(P + R_s))z_{st} - h \quad (8)$$

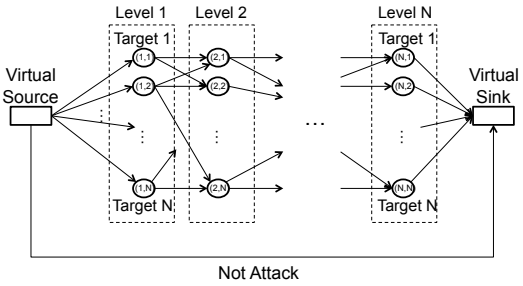


Figure 6: This figure shows an example network-flow based slave formulation. There are as many levels in the graphs as the number of targets. Each node represents a specific target. A path from the source to the sink maps to a tour taken by the defender.

One approach to identify the tour with the minimum reduced cost would be to iterate through all possible tours, compute their reduced costs, and then choose the one with the least reduced cost. However, we propose a minimum-cost integer network flow formulation that efficiently finds the optimal column (tour). Feasible tours in the domain map to feasible flows in the network flow formulation and vice-versa. The minimum cost network flow graph is constructed in the following manner. A virtual source and virtual sink are constructed to mark the beginning and ending locations, i.e. home base, for a defender tour. These two virtual nodes are directly connected by an edge signifying the "Not attack" option for the attacker. As many levels of nodes are added to the graph as the number of targets. Each level contains nodes for every target. There are links from every node on level i to every node to level $i+1$. Each node on every level i is also directly connected to the sink. Additionally, the length of the edge between any two nodes is the Euclidean distance between the two corresponding targets. Constraints are added to the slave problem to disallow a tour that covers two nodes corresponding to the same target (i.e. a network flow going through node (1,1) and (2,1) in the figure would be disallowed since both these nodes correspond to target 1). An additional constraint is added to the slave to ensure that the total length of every flow (i.e. sum of lengths of edges with a non-zero flow) is less than the specified upper bound L. Thus, the slave is setup such that there exists a one-to-one correspondence between a flow generated by the slave problem and patrol route that the defender can undertake. Figure 6 shows an example graph for the slave.

Each node representing a target is split into two dummy nodes with an edge between them. Link costs are put on these edges. The costs on these graphs are defined by decomposing the reduced cost of a tour, $\overline{c}_t$, into reduced costs over individual targets, $\hat{c}_s$. We decompose $\overline{c}_t$ into a sum of cost coefficients per target $\hat{c}_s$, so that $\hat{c}_s$ can be placed on the edges between the two dummy nodes of each target. $\hat{c}_s$ are defined as follows:

$$\overline{c}_t = \textstyle\sum_{s \in S} \hat{c}_s z_{st} - h \quad (9)$$

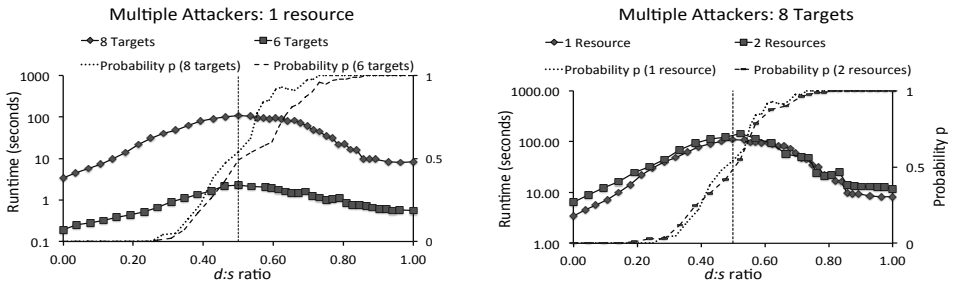
$$\hat{c}_s = (w_s(\tau_s + R_s) + (v_s - y_s)(P + R_s)) \quad (10)$$

Results: We present the results of experiments with this algorithm in Figure 7. Figure 7(a) shows the results for the SPPC domain with 1 defender resource, and with the number of targets varying from 6 to 8. For example, for the $d{:}s$ ratio of 0.50, the algorithm took 108.2 seconds to compute the optimal solution, and the value of p was 0.54 for 8 targets. Figure 7(b) shows the runtime required to compute the optimal solution for the SPPC domain with 8 targets. It varies the number of defender resources from 1 to 2. For example, for the $d{:}s$ ratio of 0.50, the algorithm took 144.0 seconds to compute the optimal solution for 2 resources, and the probability p was equal to 0.49. Additionally, p again shows a phase transition as the $d{:}s$ ratio is varied from 0 to 1.

Bayesian Single Attacker The second game model is a standard Bayesian game with a single attacker who could be of many types. Each attacker type is identified by a different payoff matrix. The defender does not know the type of the attacker she would be facing, however, the defender does know a prior probability of facing each type. The attacker knows his type as well the defender strategy, and then computes his best response. The results presented in this section show that the easy-hard-easy computation pattern is not restricted to just one domain representation but to other representations as well.

Solution Methodology: We modified the branch-and-price formulation to compute optimal solutions for this variant of the domain. Here, again, the branch-and-price formulation is composed of a branch and bound module and a column generation module. Again, the actions of the attacker are modeled as an integer variable. The branch and bound assigns a value (i.e. a specific target to attack) to this integer in every branch. The solution at each node of this tree is computed using the column generation procedure. The master and the slave problems for this column generation procedure are described below.

Master Formulation: The objective of the master formulation is to compute the probability distribution $\mathbf{x}$ over the set of tours $\mathbf{T}$ such that the expected defender utility is maximized. The master formulation is given in Equations (11) to (16). Λ specifies the set of adversary types, and is subscripted using λ. Again, Equation (13) computes the payoff of the defender. Equations (14) and (15) compute the payoff of the attacker, while ensuring that $q_s^\lambda = 1$ is feasible if and only if attacking target s is the best response of the attacker of type λ. Equations (12) and (16) ensure that $\mathbf{x}$ is a valid



(a) (b)

Figure 7: Average runtime for computing the optimal solution for a patrolling domain, along with the probability p. The vertical dotted line shows $d{:}s = 0.5$.

probability distribution.

$$\min_{x,d,q} -\sum_{\lambda\in\Lambda} d^{\lambda} \quad (11)$$

$$\text{s.t.} \sum_{\mathrm{t}\in\mathrm{T}} \mathrm{x_t} \quad \leq 1 \ (12)$$

$$d^{\lambda} - \sum_{t\in T} x_t z_{st}(\tau_s^{\lambda} + R_s^{\lambda}) + Mq_s^{\lambda} \leq M - R_s^{\lambda} \quad (13)$$

$$-k^{\lambda} - \sum_{t\in T} x_t z_{st}(P^{\lambda} + R_s^{\lambda}) + R_s^{\lambda} \quad \leq 0 \ (14)$$

$$k^{\lambda} + \sum_{t\in T} x_t z_{st}(P^{\lambda} + R_s^{\lambda}) + Mq_s^{\lambda} - R_s^{\lambda} \quad \leq M (15)$$

$$x_t \in [0,1] \quad (16)$$

Slave: The objective of the slave formulation is the compute the next best tour to add to the set of tours $\mathbf{T}$. This is again done using a minimum cost integer network flow formulation. The network flow graph is constructed in the same way as before. The updated reduced costs for this variant of the domain are computed using the same standard techniques and are given in the Equation (17). Here, w^{λ}, y^{λ}, v^{λ} and h represent the duals of Equations (13), (14), (15) and (12) respectively.

$$\overline{c}_t = \sum_{\lambda\in\Lambda}\sum_{s\in S}(w_s^{\lambda}(\tau_s^{\lambda} + R_s^{\lambda}) + (v_s^{\lambda} - y_s^{\lambda})(P_s^{\lambda} + R_s^{\lambda}))z_{st} - h \quad (17)$$

This reduced cost of a tour $\overline{c}_t$ is again decomposed into reduced costs per target in the following manner:

$$\overline{c}_t = \textstyle\sum_{s\in S} \hat{c}_s z_{st} - h \quad (18)$$

$$\hat{c}_s = \textstyle\sum_{\lambda\in\Lambda}(w_s^{\lambda}(\tau_s^{\lambda} + R_s^{\lambda}) + (v_s^{\lambda} - y_s^{\lambda})(P_s^{\lambda} + R_s^{\lambda})) \quad (19)$$

These reduced costs per target, $\hat{c}_s$, are then put as the costs on the links of the minimum cost network flow formulation.

Results: Figure 8 shows the runtime required for our algorithm to compute an optimal solution for this domain with 8 targets and 1 defender resource, along with the probability p that the decision problem is solvable. It varies the number of types from 1 to 2. For example, for the $d{:}s$ ratio of 0.50, the algorithm took 2.0 seconds to compute the optimal solution for 1 type, and the probability p was also equal to 0.47. Additionally, p again shows a phase transition as the $d{:}s$ ratio varies from 0 to 1.

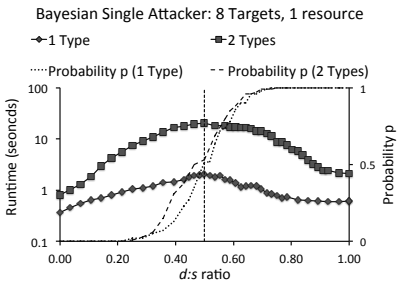


Figure 8: Average runtime for computing the optimal solution for a Bayesian Single Attacker patrolling domain, along with the probability p. The vertical dotted line shows $d{:}s = 0.5$.

Implications

We have provided evidence that the hardest random Stackelberg game instances occur at a deployment-to-saturation ratio of 0.5. This finding has two key implications.

First, it is important to compare algorithms on hard problems. If random data is used to test algorithms for security domains, it should be generated at a $d{:}s$ ratio of 0.5. There has indeed been a significant research effort focusing on the design of faster algorithms for security domains. Random data has often been used; unfortunately, we find that it tends not to have come from the $d{:}s = 0.5$ region. (Of course, the concept of a deployment-to-saturation ratio did not previously exist; nevertheless, we can assess previous work in terms of the $d{:}s$ ratio at which data was generated.) For example, Jain, Kiekintveld, and Tambe (2011) compared the performance of HBGS with DOBSS and MultipleLPs, but they only compared $d{:}s$ ratios between 0.10 and 0.20. Similarly, Pita et al. (2010) presented runtime comparisons between different algorithms, varying the number of attacker types in the security domain; all experiments in this paper were fixed at $d{:}s = 0.30$ (10 targets, 3 resources). Jain et al. (2011) showed scalability results for

RUGGED, testing at $d{:}s$ ratios of 0.10 and (mostly) 0.20. Runtime results have also been presented in other security settings (Dickerson et al. 2010; Vanek et al. 2011; Bosansky et al. 2011); these algorithms compute defender strategies for networked domains. Their experiments keep the number of resources fixed and increase the size of the underlying network; however, none of these papers provides enough detail about how instances were generated to allow us to accurately compute the $d{:}s$ ratio.

To make it easier for future researchers to test their algorithms on hard problems, we have written a benchmark generator for security games that generates instances from $d{:}s = 0.5$. This generator is written in Java, and will be provided on request. It allows users to generate instances for all domains described above, as well as to execute all the algorithms mentioned in this research. It also allows switching between GLPK and CPLEX.

Second, we observe that intermediate values of the $d{:}s$ ratio, the computationally hard region, is also the region where optimization is most valuable and hence where security officials are most likely to seek help in optimizing their resource deployment. If the $d{:}s$ ratio is large, there are enough resources to protect almost all targets, and performing a rigorous optimization offers little additional benefit. If $d{:}s$ is small, there are not enough resources for optimized deployment to have a significant impact. We show an example from random data from the SPNSC domain in Figure 9. We present results for 50 and 75 targets, each averaged over 100 random instances. The x-axis plots the $d{:}s$ ratio, and the y-axis shows the difference between the defender utilities obtained by the optimal strategy and a naïve randomization strategy. A low utility difference implies that the naïve strategy is almost as good as the optimal strategy, whereas a high difference shows that it is worthwhile to invest in computing the optimal strategy. The naïve strategy we use here prioritizes targets based on attacker's payoff of successfully attacking a target. It then uniformly distributes its resources over twice as many top targets as the number of resources. For example, at a $d{:}s$ ratio of 0.5, for 50 targets, the difference in utilities between the optimal solution and the solution from the randomized strategy was 5.07 units, whereas it was 0.21 units at a $d{:}s$ ratio of 1. This suggests that computationally hard settings are also those where security forces would benefit the most from adopting nontrivial strategies; hence, researchers should concentrate on these problems.

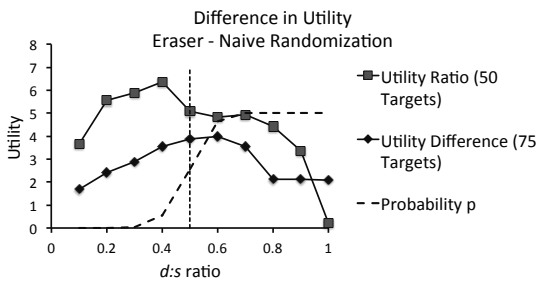


Figure 9: The difference between expected defender utilities from ERASER and a naïve randomization policy. The vertical line shows $d{:}s = 0.5$.

Conclusions

Stackelberg security games are an widely studied model of security domains, with important deployed applications. We introduced the concept of the deployment-to-saturation ($d{:}s$) ratio, a domain-spanning measure of the density of defender coverage in any security problem. We showed that the computationally hardest random instances of such games occur at a $d{:}s$ ratio of 0.5. We further demonstrated that this hard region corresponds to a phase transition in the probability that a corresponding decision problem for the Stackelberg security game has a solution. Our evidence for this correlation of the computationally hardest instances with the phase transition was based on eight different algorithms, two deployed in real-world applications, and in each case variations in the number of targets, attacker types, solvers used to solve them, and/or different underlying solution mechanisms; our results were amazingly robust across all of these settings. We argued that our results have two important implications. First, researchers comparing and benchmarking algorithms for Stackelberg security games on random data should concentrate on problems with $d{:}s = 0.5$ (as, unfortunately, much previous work has failed to do); we wrote a free benchmark generator to help researchers do this in the future. Second, we argued that problems of real-world interest are likely to arise in the computationally hardest region, around the $d{:}s$ ratio of 0.5, and backed up this claim by showing that an extremely naïve defender strategy works almost as well as the optimal strategy at both large and small $d{:}s$ values.

References

An, B.; Pita, J.; Shieh, E.; Tambe, M.; Kiekintveld, C.; and Marecki, J. 2011. GUARDS and PROTECT: Next Generation Applications of Security Games. In *SIGECOM*, volume 10.

Barnhart, C.; Johnson, E.; Nemhauser, G.; Savelsbergh, M.; and Vance, P. 1994. Branch and Price: Column Generation for Solving Huge Integer Programs. In *Operations Research*, volume 46, 316–329.

Bosansky, B.; Lisy, V.; Jakob, M.; and Pechoucek, M. 2011. Computing Time-Dependent Policies for Patrolling Games with Mobile Targets. In *Tenth International Conference on Autonomous Agents and Multiagent Systems*, 989–996.

Cheeseman, P.; Kanefsky, B.; and Taylor, W. M. 1991. Where the Really Hard Problems are. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 331–337.

Conitzer, V., and Sandholm, T. 2006. Computing the Optimal Strategy to Commit to. In *Proceedings of the ACM Conference on Electronic Commerce (ACM-EC)*, 82–90.

Cook, S. A., and Mitchell, D. G. 1997. Finding Hard Instances of the Satisfiability Problem: A Survey. In *DIMACS workshop on the satisfiability problem: theory and applications*, 1–17. American Mathematical Society.

Crawford, C. J. M., and Auton, L. D. 1996. Experimental Results on the Crossover Point in Random 3SAT. *Artificial Intelligence* 81:31–35.

Dickerson, J.; Simari, G.; Subrahmanian, V.; and Kraus, S. 2010. A Graph-Theoretic Approach to Protect Static and Moving Targets from Adversaries. In *Proceedings of the Ninth International Conference on Autonomous Agents and Multiagent Systems (AAMAS-2010)*, 299–306.

Frank, J., and Martel, C. U. 1995. Phase Transitions in the Properties of Random Graphs. In *Principles and Practice of Constraint Programming (CP-95*, 62–69.

Frank, J.; Gent, I. P.; and Walsh, T. 1998. Asymptotic and Finite Size Parameters for Phase Transitions: Hamiltonian Circuit as a Case Study. *Inf. Process. Lett.* 65:241–245.

Gatti, N. 2008. Game Theoretical Insights in Strategic Patrolling: Model and Algorithm in Normal-Form. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, 403–407.

Gent, I. P., and Walsh, T. 1996. The TSP Phase Transition. *Artificial Intelligence* 88(12):349 – 358.

Hauert, C., and Szab, G. 2005. Game theory and Physics. *Am. J. Phys.* 73(5):405–414.

Jain, M.; Kardes, E.; Kiekintveld, C.; Ordóñez, F.; and Tambe, M. 2010a. Security Games with Arbitrary Schedules: A Branch and Price Approach. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*.

Jain, M.; Tsai, J.; Pita, J.; Kiekintveld, C.; Rathi, S.; Tambe, M.; and Ordóñez, F. 2010b. Software Assistants for Randomized Patrol Planning for the LAX Airport Police and the Federal Air Marshals Service. *Interfaces* 40:267–290.

Jain, M.; Korzhyk, D.; Vanek, O.; Conitzer, V.; Pechoucek, M.; and Tambe, M. 2011. A Double Oracle Algorithm for Zero-Sum Security Games on Graphs. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.

Jain, M.; Kiekintveld, C.; and Tambe, M. 2011. Quality-bounded Solutions for Finite Bayesian Stackelberg Games: Scaling up. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.

Kiekintveld, C.; Jain, M.; Tsai, J.; Pita, J.; Tambe, M.; and Ordóñez, F. 2009. Computing Optimal Randomized Resource Allocations for Massive Security Games. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 689–696.

Larrabee, T., and Tsuji, Y. 1993. Evidence for a Satisfiability Threshold for Random 3CNF Formulas. In *Proceedings of the AAAI Spring Symposium*.

Letchford, J., and Vorobeychik, Y. 2011. Computing Randomized Security Strategies in Networked Domains. In *Proceedings of the Workshop on Applied Adversarial Reasoning and Risk Modeling (AARM) at AAAI*.

Mitchell, D.; Selman, B.; and Levesque, H. 1992. Hard and Easy Distributions of SAT Problems. In *Proceedings of the American Association for Artificial Intelligence*, 459–465.

Monasson, R.; Zecchina, R.; Kirkpatrick, S.; Selman, B.; and Troyansky, L. 1999. Determining Computational Complexity from Characteristic 'Phase Transitions'. *Nature* 400(6740):133–137.

Nudelman, E.; Leyton-Brown, K.; Hoos, H.; Devkar, A.; Shoham, Y.; and Wallace, M. 2004. *Understanding Random SAT: Beyond the Clauses-to-Variables Ratio*, volume 3258. Springer Berlin / Heidelberg. 438–452.

Paruchuri, P.; Pearce, J. P.; Marecki, J.; Tambe, M.; Ordóñez, F.; and Kraus, S. 2008. Playing Games with Security: An Efficient Exact Algorithm for Bayesian Stackelberg games. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 895–902.

Pita, J.; Jain, M.; Ordonez, F.; Tambe, M.; and Kraus, S. 2010. Robust Solutions to Stackelberg Games: Addressing Bounded Rationality and Limited Observations in Human Cognition. *Artificial Intelligence Journal, 174(15):1142-1171, 2010*.

Savit, R.; Manuca, R.; and Riolo, R. 1999. Adaptive Competition, Market Efficiency, Phase Transitions and Spin-Glasses. *University of Michigan* 82:2203–2206.

Selman, B.; Mitchell, D.; and Levesque, H. 1996. Generating Hard Satisfiability Problems. *Artificial Intelligence* 81:17–29.

Slaney, J., and Walsh, T. 2002. Phase Transition Behavior: from Decision to Optimization. In *Proceedings of the 5th International Symposium on the Theory and Applications of Satisfiability Testing, SAT*.

Vanek, O.; Jakob, M.; Lisy, V.; Bosansky, B.; and Pechoucek, M. 2011. Iterative Game-theoretic Route Selection for Hostile Area Transit and Patrolling. In *Tenth International Conference on Autonomous Agents and Multiagent Systems*, 1273–1274.