The Deployment-to-Saturation Ratio in Security Games

Manish Jain
manish.jain@usc.edu
University of Southern California,
Los Angeles, California 90089.

Kevin Leyton-Brown
kevinlb@cs.ubc.edu
University of British Columbia
Vancouver, B.C., Canada V6T 1Z4.

Milind Tambe
tambe@usc.edu
University of Southern California,
Los Angeles, California 90089.

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 problem instances for which this ratio is 0.5 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. This finding has 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 computationally hard region is also one where optimization would be of most benefit to security agencies, and thus requires significant attention from researchers in this area. Furthermore, we use the concept of phase transitions to better understand this computationally hard region. We define a decision problem related to security games, and show that the probability that this problem has a solution exhibits a phase transition as the deployment-to-saturation ratio crosses 0.5. We also demonstrate that this phase transition is invariant to changes both in the domain and the domain representation, and that the phase transition point corresponds to the computationally hardest instances.

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 investigate what properties of Stackelberg security game instances make them hard to solve in practice, across different input sizes and different security domains. We show that this question can be answered using the novel concept of the deployment-to-saturation (d:s) ratio. This ratio, which we denote d:s, is defined as the number of deployed defender resources divided by the number of resources beyond which additional deployments would not provide any benefit to the defender. We show that the hardest computational instances arise when this ratio is 0.5, independent of the domain representation, model and solver. Specifically, we consider three different classes of security domains, eight different MIP algorithms (including two algorithms actually deployed in practice), five different underlying MIP solvers, two different variations on the Stackelberg equilibrium concept, and a variety of domain sizes and conditions.

We identify two important implications of our 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 computationally hard region is the point where optimization offers the greatest benefit to security agencies, implying that problems in this region deserve increased attention from researchers.

Finally, we leverage the concept of *phase transitions* to better understand this algorithm-independent, computationally hard region. This approach to understanding algorithm-independent structural properties was pioneered by Cheeseman, Kanefsky, and Taylor (1991) and Mitchell, Selman, and Levesque (1992). They showed that the probability that a uniform-random 3-SAT instance will be satisfiable exhibits a phase transition when the number of variables is fixed and the number of clauses crosses roughly 4.3 times the number of variables, which corresponds to the point where the probability that the instance will be solvable crosses 0.5. Phase transitions have also been used to understand the computational

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

Figure 1: Example security game with two targets and one attacker type.

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 the decision version of the optimization problem (i.e., asking whether or not a solution exists with a given objective function value). We show that security games exhibit a phase transition at 0.5 for random Stackelberg security game instances, and that this phase transition corresponds to the computationally hardest instances at the d:s ratio of 0.5.

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 targets she can protect, 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.

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.

A Strong Stackelberg Equilibrium (SSE) is a mixed strategy for the defender that maximizes her expected utility, given that the attacker plays a pure strategy best response to this mixed strategy. A MILP to compute the SSE is given in Problem 1.[1] Equation (2) computes the expected defender payoff d^λ achieved against attacker of type λ when the defender plays mixed strategy $\mathbf{x}$ and the attacker's best response is a_j^λ. Similarly, Equations (3), (4), (5) and (8) together guarantee that every attacker type plays a pure strategy best response a_j^λ to the defender mixed strategy $\mathbf{x}$. Finally, Equation (6) and (7) ensure that the defender plays a valid mixed

[1]Problem 1 is a generic definition of SSE. In practice, however, different formulations of the problem are used with heuristic algorithms to provide speedups on specific domains.

Problem 1 (Strong Stackelberg Equilibrium (SSE))

$$\text{maximize} \sum_{\lambda \in \Lambda} d^\lambda p^\lambda \tag{1}$$

$$d^\lambda - \sum_i D^\lambda(i,j) x_i \leq (1 - a_j^\lambda) Z \qquad \forall j, \forall \lambda \tag{2}$$

$$k^\lambda - \sum_i A^\lambda(i,j) x_i \geq 0 \qquad \forall j, \forall \lambda \tag{3}$$

$$k^\lambda - \sum_i A^\lambda(i,j) x_i \leq (1 - a_j^\lambda) Z \qquad \forall j, \forall \lambda \tag{4}$$

$$\sum_{j \in J} a_j^\lambda = 1 \qquad \forall \lambda \tag{5}$$

$$\sum_i x_i = 1 \tag{6}$$

$$x_i \geq 0 \qquad \forall i \in I \tag{7}$$

$$a_j^\lambda \in \{0, 1\} \qquad \forall j \in J \tag{8}$$

Symbol	Definition
D	Defender
A	Attacker
Λ	Set of attacker types
λ	An element of set Λ
p^λ	The probability that type λ will be realized
$\mathbf{x}$	Mixed strategy of D
$\mathbf{a}^\lambda$	Mixed strategy of A^λ
$D^\lambda(i,j)$	Expected utility of D when playing pure strategy i and when attacker of type λ plays pure strategy j
$A^\lambda(i,j)$	Expected utility of A when playing pure strategy i and when attacker of type λ plays pure strategy j
Z	Large positive constant

Table 1: Notation.

strategy $\mathbf{x}$. We refer the readers to Paruchuri et al. (2008) and Kiekintveld et al. (2009) for further details on this MILP and the computation of SSE. The notation used in this MILP is given in Table 1. Here, i and j refer to the i^{th} pure strategies of the defender and the j^{th} pure strategy of the attacker respectively. While the set of attacker pure strategies in all domains is equivalent to the set of targets to attack, the pure strategies of the defender vary from domain to domain as we will describe below.

Computing the SSE solution for Bayesian Stackelberg games is NP-hard (Conitzer and Sandholm 2006). However, the problem is nevertheless important to solve: Bayesian Stackelberg games are an appropriate model for many real-world security scenarios, and hence many algorithms have been designed for solving these games. Many software assistants built on these algorithms have also been successfully deployed (Jain et al. 2010b; An et al. 2011).

Security Domains

In this paper, we investigate the runtime performance of different algorithms for three different security domains. This section describes these domains.

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. Thus, in this domain, a pure strategy i of the defender is a joint allocation of all defender resources to targets. Two different representations have been proposed for modeling SPNSC problems.

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. A real-world application of SPNSC is setting up vehicular inspection checkpoints at the Los Angeles International Airport, as implemented in ARMOR (Pita et al. 2008; Jain et al. 2010b), which was deployed in August, 2007. ARMOR models the problem as an SPNSC problem and uses DOBSS to compute the SSE.

Security game compact representation. This representation only captures the probability of the *defender* covering any particular target, since the expected utilities of both the defender and the attacker in a security game only depends on the target attacked and the probability of defender protecting that target. ERASER (Kiekintveld et al. 2009) is the only algorithm that has been proposed for computing SSE for Bayesian Stackelberg games represented in this form.

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. Therefore, in this domain, a pure strategy i of the defender is a joint schedule of all defender resources. 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). The IRIS (Jain et al. 2010b) system was deployed in October 2009 to solve the scheduling problem faced by the Federal Air Marshal Service; it models the problem as a SPARS problem and uses the ASPEN algorithm.

SPPC Domain

We introduce a new security 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. For example, if $L = 5$, then the length of the tour taken by each defender resource must not exceed 5 units. Thus, similar to the SPARS domain, a pure strategy i of the defender in this domain is a joint tour of all defender resources. The defender's objective is to find the optimal mixed strategy over tours for all its resources in order to maximize her expected utility. This domain captures properties of patrolling problems studied by researchers across many real-world domains (An et al. 2011; Bosansky et al. 2011; Vanek et al. 2011). We consider the two variants of this domain featuring different attacker models. In each case, because no other algorithms are available, we propose a novel algorithm for computing SSE.

Multiple Attackers. For this domain we propose a branch-and-price based formulation for computing the optimal strategy for the defender, similar in spirit to ASPEN (Jain et al. 2010a). We omit the details of our algorithm for space reasons; they can be found in Section 2 of our online Appendix (`http://teamcore.usc.edu/DTS/Appendix.pdf`).

Bayesian Single Attacker. We also consider the Bayesian version of the patrolling domain, in which there is a single attacker with many potential types. For example, this problem is faced by the US Coast Guard when patrolling a set of targets along the port to protect against potential threats. The PROTECT system (An et al. 2011) deployed in April 2011, models this problem as an SPPC problem with a Bayesian single attacker. We propose a second branch-and-price algorithm for computing SSE in this domain; again the details are available in Section 3 of our online Appendix.

Runtime Variation

As mentioned previously, this paper focuses on the runtime required by the different algorithms that compute solutions for instances of security games for the three domains described above. Figure 2 shows the runtime for computing solutions using DOBSS, ASPEN and multiple-attacker branch-and-price algorithm for the SPNSC, SPARS and SPPC domains respectively. (Recall that these three configurations are particularly interesting, as they are the ones that have been deployed in practice.) The x-axis in each figure shows the number of available resources to the defender, and the y-axis shows the runtime in seconds. In the SPNSC and the SPARS domains we define the number of resources as the number of officers available to the defender; in the SPPC domain, we

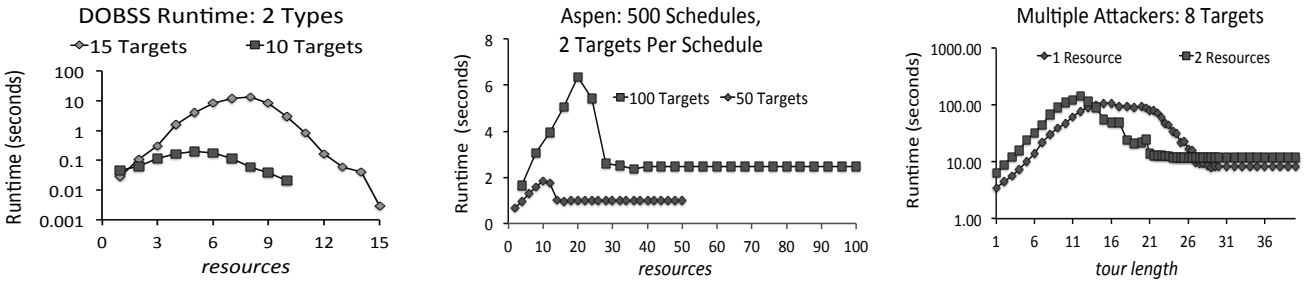


Figure 2: Average running time of DOBSS, ASPEN and multiple-attacker branch-and-price algorithm for SPNSC, SPARS and SPPC domains respectively.

define it as the maximum feasible tour length. These graphs show that there is no unified value of the number of defender resources which makes security game instances hard to solve. We also tried normalizing the number of resources by the number of targets, however, we found similar inconsistencies across different domains even with that normalization.

Deployment to Saturation Ratio

We now propose the novel concept of the *deployment-to-saturation* (d:s) ratio, a concept that unifies the domain independent properties of problem instances across different security domains. Specifically, we consider the three security domains introduced before, 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. We show experimentally that the hardest problem instances occur when the dts ratio is about 0.5. (Specifically, in our experiments, the hardest problem instances occurred at the d:s ratios between 0.48 and 0.54. However, because of discretization, we were not able to test all d:s ratios in all domains. Without exception, the hardest value was the feasible value closest to 0.5. This fact, in combination with our phase transition arguments presented at the end of the paper, lead us to conclude that the hardest region is precisely d:$s = 0.5$.)

More specifically, 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 this definition, *deployment* denotes the number of defender resources available to be allocated, and *saturation* denotes 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. For the SPNSC and the SPARS domain, deployment denotes the number of available security personnel, whereas saturation refers to the minimum number of officers required to cover all targets with a probability of 1. For the SPPC domain, deployment denotes the maximum feasible tour length, while saturation denotes the minimum tour length required by the team of defender resources such that the team can tour all the targets with a probability of 1.

We now present results for all the three security domains. 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 and the y-axis shows the runtime in CPU-seconds. Experiments were conducted using CPLEX 12.2 unless otherwise noted.

SPNSC Domain

For this domain, we considered both the general-sum and security game compact representations.

General sum representation. We conducted experiments varying the algorithm, number of attacker types and number of targets. The results for the general sum representation are plotted in Figures 3(a), 3(b) and 3(c). The payoffs for the two players were selected uniformly at random: the payoffs for success and failure were selected from the ranges $[1, 10]$ and $[-10, -1]$ respectively.

Figure 3(a) shows that the runtime required by all three algorithms (MultipleLPs, DOBSS and HBGS) peaks at d:$s = 0.53$. (While we would like to have observed peaks at 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. However, there is also a more critical issue: 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. 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. This experiment shows that computation is hardest for the SPNSC problem instances when the d:s ratio is about 0.5.

The next two experiments study runtime variation when the number of targets and the number of types in the domain vary. Figure 3(b) varies the number of targets in the domain from 10 to 15. Similarly, Figure 3(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),

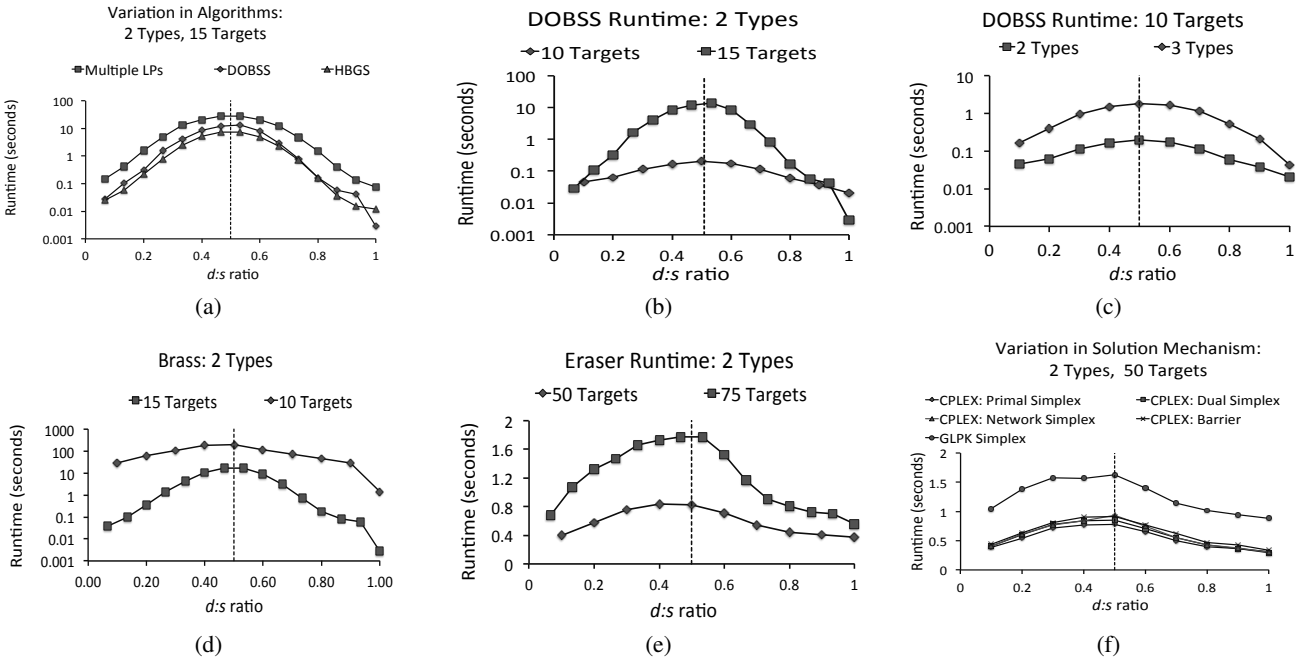


Figure 3: Average runtime of computing the optimal solution for a SPNSC problem instance. The vertical dotted line shows $d{:}s = 0.5$.

We also ran experiments with BRASS, which computes an ϵ-Stackelberg equilibrium (Pita et al. 2010). These results are shown in Figure 3(d). 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.

Security game compact representation. 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 3(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. This was the computationally hardest point for 75 targets. We also varied the number of types, and those results also confirmed our claim. These results can be found in Section 4 of our online Appendix.

Our next experiment investigated the effect on ERASER's runtime of changing its underlying solver and solution mechanism. 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. TThese experiments again show that the $d{:}s$ ratio of 0.5 corresponds with the computationally hardest instances across a range of underlying solver and solution mechanisms for the SPNSC domain.

SPARS Domain

We conducted experiments varying the length of a schedule, the number of targets and the number of schedules in SPARS problem instances. ASPEN was used to compute solutions. 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, denoted by $|S|$, was 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. This was computationally the hardest point for $|S| = 2$. For $|S| = 4$, the computationally hardest point required 28.9 seconds at $d{:}s = 0.5$ (10 available resources, 20 required for saturation).

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. 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 which, again, was the computationally hardest point for 50 targets.

SPPC Domain

We present results for both the multiple attacker and the Bayesian single attacker variants in Figure 5. Figure 5(a) shows the results for the multiple attacker SPPC domain with

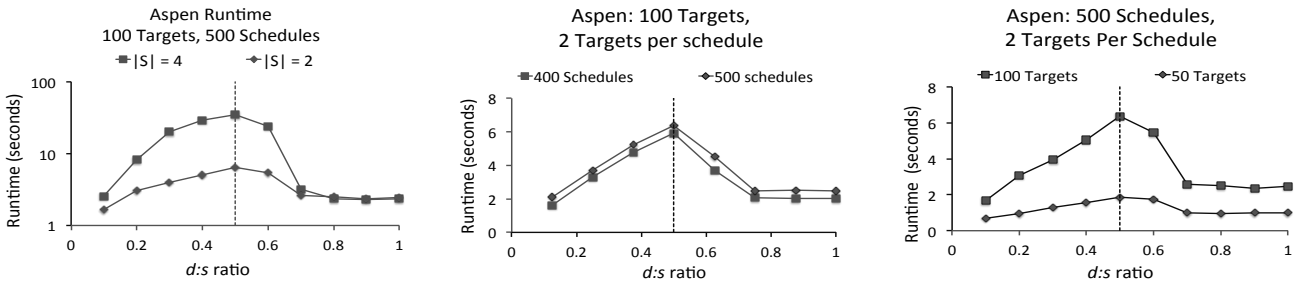


Figure 4: Average runtime of computing the optimal solution for a SPARS game using ASPEN. The vertical dotted line shows $d{:}s = 0.5$.

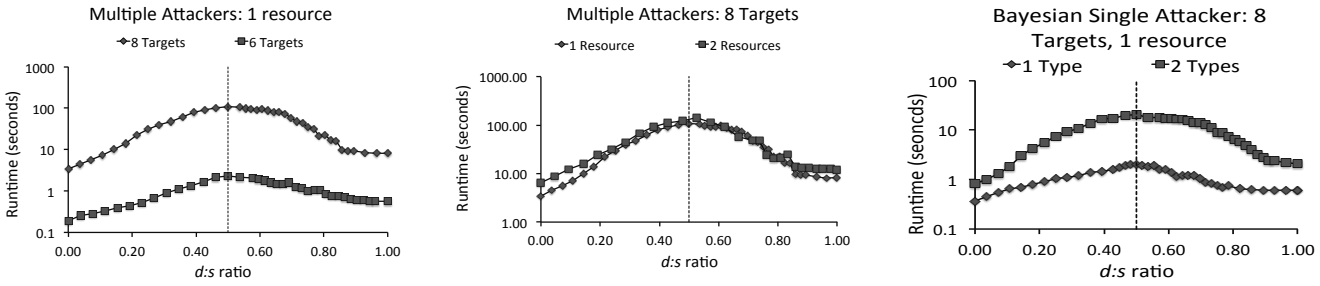


Figure 5: Average runtime for computing the optimal solution for a patrolling domain. The vertical dotted line shows $d{:}s = 0.5$.

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. Figure 5(b) shows the runtime required to compute the optimal solution for the multple attacker 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. This was again the computationally hardest point.

Similarly, Figure 5(c) shows the runtime required for our branch-and-price based algorithm to compute an optimal solution for the Bayesian single attacker SPPC 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.

Implications of our Findings

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 is available for download at `http://teamcore.usc.edu/DTS`. It allows users to generate instances for all domains described above, as well as to compute solutions for 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

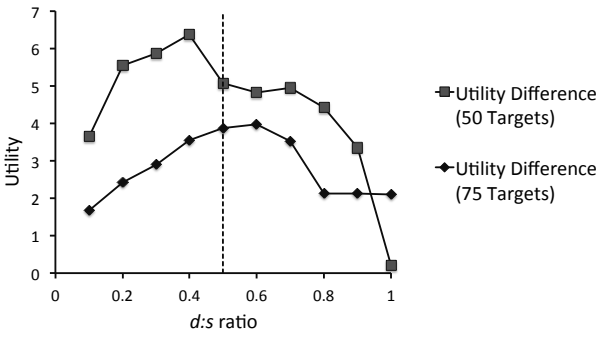


Figure 6: The difference between expected defender utilities from ERASER and a naïve randomization policy. The vertical line shows $d{:}s = 0.5$.

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. To make this intuition concrete, we show an example from random data from the SPNSC domain in Figure 6. 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 the attacker's payoff for successfully attacking a target, and 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.

Phase Transitions

All our runtime results show an easy-hard-easy computational pattern as the $d{:}s$ ratio increases from 0 to 1, with the hardest problems at $d{:}s = 0.5$. Such easy-hard-easy patterns have also been observed in other NP-complete problems, most notably 3-SAT (Cheeseman, Kanefsky, and Taylor 1991; Mitchell, Selman, and Levesque 1992). In 3-SAT, the hardness of the problems varies with the clause-to-variable (c/v) ratio, with the hardest instances occurring at about $c/v = 4.26$. The SAT community has used the concept of *phase transitions* to better understand this hardness peak.

Phase transitions have also been used to study properties of optimization problems. Specifically, one can derive the decision version of an optimization problem by asking "does there exist a solution with objective function value $\geq k$?", and then looking for a phase transition in the decision problem. This approach was pioneered by Gent et al. (1995), who studied the properties of four different optimization problems including TSP and Boolean circuit synthesis. They computed the optimal tour length from the TSP optimization problem, and then used this value as k to define the decision version of the TSP instance. They showed that a phase transition in the solubility of this problem corresponded to the computationally hardest region. The same approach was later also used by Gent and Walsh (1996).

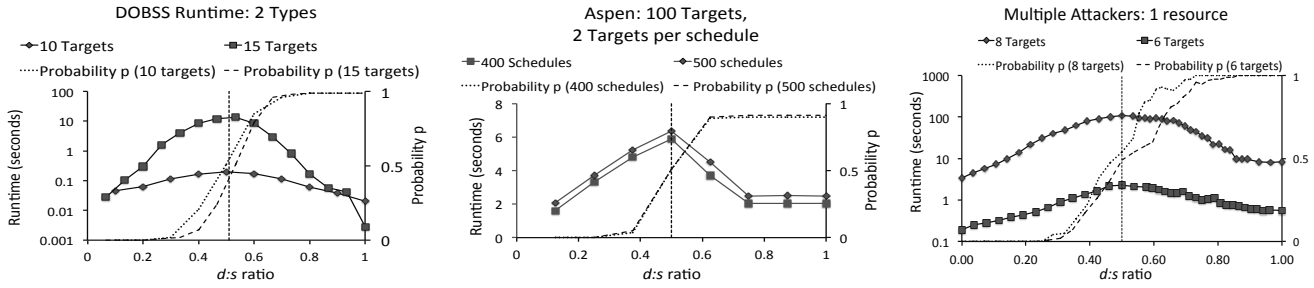
In the context of game theory, phase transitions have been used to analyze the efficiency of markets in adaptive games (Savit, Manuca, and Riolo 1999) and the probability of cooperation in evolutionary game theory (Hauert and Szab 2005). To our knowledge, our work is the first that identifies such structural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

Phase Transitions in Security Games

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. 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. Following Gent et al. (1995), 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. (Observe that this definition guarantees that at a $d{:}s$ ratio of 0.5, exactly 50% of problem instances will have a feasible solution. On the other hand, it does not guarantee that there will be a phase transition—i.e., a sharp change—in the probability of solvability.) We estimated D^* by sampling 100 random problem instances at $d{:}s = 0.5$, and computing the sample median of their objective function values.

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 at $d{:}s = 0.5$. This phase transition is independent of the security domain or its representation. Furthermore, this phase transition aligns with the computationally hardest instances.*

To support this claim, we computed the probability of solvability of the decision problem $SSE(D^*)$ for all the security domains and algorithms mentioned above. We only include one result from each of the three domains here; the rest can be obtained from Section 5 of our online Appendix. The results are shown in Figure 7. The x-axis shows the $d{:}s$ ratio as before, the primary y-axis shows the runtime in seconds, and the secondary y-axis shows the probability p of finding a solution to $SSE(D^*)$. We plot runtimes using solid lines, as before, and plot p using a dashed line. Figure 7(a) presents results for the DOBSS algorithm for the SPNSC domain for 10 targets and 2 and 3 attacker types. As expected, the $d{:}s$ ratio of 0.5 corresponds with $p = 0.51$ as well as the computationally hardest instances; more interestingly,



(a) (b) (c)

Figure 7: Average runtime for computing the optimal solution for an example setting for all the three security domains, along with the probability p plotted on the secondary y-axis. The vertical dotted line shows $d{:}s = 0.5$.

we observe that p undergoes a phase transition as the $d{:}s$ grows. Similarly, Figure 7(b) shows results for the ASPEN algorithm for the SPARS domain with 100 targets, 2 targets per schedule and 400 and 500 schedules, and Figure 7(c) shows results for the multiple attacker SPPC domain for 1 defender resource. In both cases, we again observe a phase transition in p.

One might wonder how we can decide that the increase in p is steep enough to be called a phase transition. We know from both the experimental and theoretical literature on phase transitions in decision problems that the transition becomes steeper as problem size increases, approaching a step function in the limit (Kirkpatrick and Selman 1994; Friedgut 1998). This is a property that we can check for experimentally. We conducted experiments in the SPNSC domain, since it is the easiest to solve at widely varying problem sizes; the results are shown in Figure 8. The x-axis shows the $d{:}s$ ratio, and the y-axis shows the probability p of finding a feasible solution to the decision version of a corresponding SPNSC problem; we plot results for problem instances with 50, 100 and 150 targets. We observe the desired result: the phase transition indeed becomes sharper as the number of targets increases.

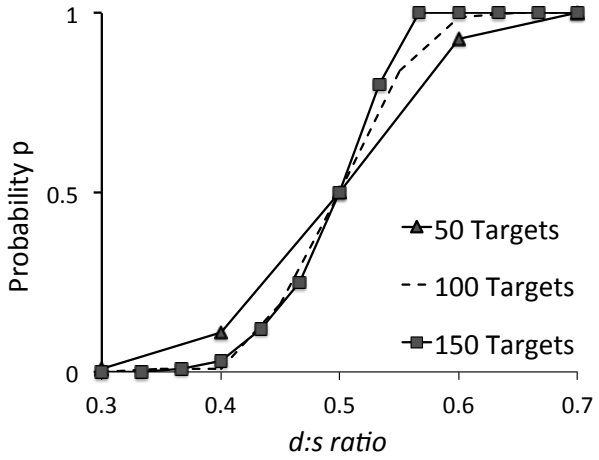


Figure 8: Probability that the decision problem $SSE(D^*)$ is soluble for SPNSC instances of three problem sizes. The phase transition gets sharper as the problem size increases.

Conclusions

Stackelberg security games are a 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. Our evidence for this correlation of the $d{:}s$ ratio of 0.5 with the computationally hardest instances 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 robust across all of these settings. We provided evidence for two important implications of our results. 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. 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.

Acknowledgements

This research was supported by the United States Department of Homeland Security through the Center for Risk and Economic Analysis of Terrorism Events (CREATE) under grant number 2010-ST-061-RE0001, and Google Faculty Research Award and NSERC Discovery Grant. All opinions, findings, conclusions and recommendations in this document are those of the authors and do not necessarily reflect views of the United States Department of Homeland Security.

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 Exch*, volume 10, 31–34.

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.

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.; Gent, I. P.; and Walsh, T. 1998. Asymptotic and Finite Size Parameters for Phase Transitions: Hamiltonian Circuit as a Case Study. *Information Processing Letters* 65:241–245.

Friedgut, E. 1998. Sharp Thresholds of Graph Properties, and the k-SAT Problem. *Journal of the American Mathematical Society* 12:1017–1054.

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. 1995. Phase Transitions from Real Computational Problems. In *Proceedings of the 8th International Symposium on Artificial Intelligence*, 356–364.

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. *American Journal of Physics* 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.

Kirkpatrick, S., and Selman, B. 1994. Critical Behavior in the Satisfiability of Random Boolean Formulae. *Science* 264:1297–1301.

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.

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.; Western, C.; Portway, C.; Tambe, M.; Ordóñez, F.; Kraus, S.; and Paruchuri, P. 2008. Deployed ARMOR Protection: The Application of a Game-theoretic Model for Security at the Los Angeles International Airport. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS), Industry Track*, 125–132.

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.

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.