Playing Games for Security: An Efficient Exact Algorithm for Solving Bayesian Stackelberg Games

Praveen Paruchuri
Intelligent Automation Inc.,
Rockville, MD
pparuchuri@i-a-i.com

Jonathan P. Pearce, Janusz Marecki, Milind Tambe, Fernando Ordonez
Univ. of Southern California,
Los Angeles, CA
{jppearce,marecki,tambe,fordon}@usc.edu

Sarit Kraus
Bar-Ilan University,
Ramat-Gan 52900, Israel
sarit@cs.biu.ac.il

ABSTRACT

In a class of games known as Stackelberg games, one agent (the leader) must commit to a strategy that can be observed by the other agent (the follower or adversary) before the adversary chooses its own strategy. We consider Bayesian Stackelberg games, in which the leader is uncertain about the types of adversary it may face. Such games are important in security domains, where, for example, a security agent (leader) must commit to a strategy of patrolling certain areas, and a robber (follower) has a chance to observe this strategy over time before choosing its own strategy of where to attack. This paper presents an efficient exact algorithm for finding the optimal strategy for the leader to commit to in these games. This algorithm, DOBSS, is based on a novel and compact mixed-integer linear programming formulation. Compared to the most efficient algorithm known previously for this problem, DOBSS is not only faster, but also leads to higher quality solutions, and does not suffer from problems of infeasibility that were faced by this previous algorithm. Note that DOBSS is at the heart of the ARMOR system that is currently being tested for security scheduling at the Los Angeles International Airport.

Categories and Subject Descriptors

I.2.11 [Computing Methodologies]: Artificial Intelligence: Distributed Artificial Intelligence - Intelligent Agents

General Terms

Security, Design, Theory

Keywords

Security of Agent Systems, Game Theory, Bayesian and Stackelberg Games

1. INTRODUCTION

In some multiagent settings, one agent must commit to a strategy before the other agents choose their own strategies. These scenarios are known as Stackelberg games [6, 12]. In a Stackelberg game, a leader commits to a strategy first, and then a follower selfishly optimizes its own reward, *considering the action chosen by the leader.* Stackelberg games are commonly used to model attacker-defender scenarios in security domains [2], as well as in patrolling [12], and could potentially be used in many other situations such as network routing [10], pricing in transportation systems [4], setting up security checkpoints and other adversarial domains. For example consider a domain in which a single security agent is responsible for patrolling a region, searching for a robber. Since the security agent (the leader) cannot be in all areas of the region at once, it must instead choose some strategy of patrolling various areas within the region, one at a time. This strategy could be a mixed strategy in order to be unpredictable to the robber (follower). The robber, after observing the pattern of patrols over time, can then choose its own strategy of choosing a location to rob.

Although the follower in a Stackelberg game is allowed to observe the leader's strategy before choosing its own strategy, there is often an advantage for the leader over the case where both players must choose their moves simultaneously. To see the advantage of being the leader in a Stackelberg game, consider a simple game with the payoff table as shown in Table 1, adapted from [5]. The leader is the row player and the follower is the column player.

	c	d
a	2,1	4,0
b	1,0	3,2

Table 1: Payoff table for example normal form game.

The only pure-strategy Nash equilibrium for this game is when the leader plays a and the follower plays c which gives the leader a payoff of 2; in fact, for the leader, playing b is strictly dominated. However, in the simultaneous game if the leader can commit to playing b before the follower chooses its strategy, then the leader will obtain a payoff of 3, since the follower would then play d to ensure a higher payoff for itself. If the leader commits to a uniform mixed strategy of playing a and b with equal (0.5) probability, then the follower will play d, leading to a payoff for the leader of 3.5.

This paper focuses on the problem of determining the optimal strategy for a leader to commit to in a Bayesian Stackelberg game, i.e. a Stackelberg game where the leader may face one of multiple follower types. Such a Bayesian Stackelberg game may arise in a security domain because for example, when patrolling a region, a security robot may only have uncertain knowledge about different robber types it may face. Unfortunately, this problem of choosing an optimal strategy for the leader to commit to in a Bayesian Stackelberg game is NP-hard [5]. This result explains the compu-

Cite as: Playing Games for Security: An Efficient Exact Algorithm for Solving Bayesian Stackelberg Games, Praveen Paruchuri, Jonathan P. Pearce, Janusz Marecki, Milind Tambe, Fernando Ordonez, Sarit Kraus, *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, Padgham, Parkes, Müller and Parsons (eds.), May, 12-16.,2008,Estoril,Portugal,pp.895-902.

tational difficulties encountered in solving such games. In particular, methods for finding optimal strategies for non-Bayesian games can be applied to Bayesian Stackelberg games[5] by converting the Bayesian game into a normal-form game by the Harsanyi transformation [7]. However, by transforming the game, the compact structure of the Bayesian game is lost. In addition, methods such as the one outlined in [5] require running a set of multiple linear programs. If on the other hand, we wish to compute the highest-reward Nash equilibrium, new methods such as MIP-Nash, using mixed-integer linear programs (MILPs) [15] may be used, since the highest-reward Bayes-Nash equilibrium is equivalent to the corresponding Nash equilibrium in the transformed game. However, as stated above the compactness in structure of the Bayesian game is lost and the procedure is NP-hard. In addition, since the Nash equilibrium assumes a simultaneous choice of strategies, the advantages of being the leader are not considered. Finally, more recently, an approximation technique named ASAP for solving the Bayesian Stackelberg games has been developed in [12]. However, as we observe from our computational experiments, ASAP encounters difficulties in finding feasible solutions since it has to solve a problem with many more integer variables, particularly as we scale up problem instances. Thus, finding more efficient and compact techniques for choosing the optimal strategies for the Bayesian Stackelberg games is an important open issue.

In this paper we focus on such Bayesian Stackelberg games. In particular, we introduce an efficient exact method for finding the optimal leader strategy in such games, known as DOBSS (Decomposed Optimal Bayesian Stackelberg Solver). This method has three key advantages. First, the method allows for a Bayesian game to be expressed compactly without requiring conversion to a normal-form game via the Harsanyi transformation. Second, the method requires only one mixed-integer linear program (MILP) to be solved, rather than a set of linear programs as in [5], thus leading to a further performance improvement. Third, it directly searches for an optimal leader strategy, rather than a Nash (or Bayes-Nash) equilibrium, thus allowing it to find high-reward non-equilibrium strategies (thus exploiting the advantage of being the leader). In addition to introducing DOBSS, additional contributions in this paper include proofs of its correctness and a detailed experimental comparison with competing methods and its analysis. Furthermore, DOBSS is at the heart of the ARMOR [13] system that is currently being tested for security scheduling at the Los Angeles International Airport, which has been described in popular scientific magazines and news media such as [11].

2. PROBLEM DEFINITION AND RELATED WORK

A Bayesian game contains a set of N agents, and each agent n must be one of a given set of types θ_n. The Bayesian Stackelberg games we consider in this paper have two agents, the leader and the follower. θ_1 is the set of possible types for the leader and θ_2 is the set of possible types for the follower. For the security games of interest in this paper, we assume that there is only one leader type (e.g. only one robot enforcing security), although there are multiple follower types (e.g. multiple robber types). Therefore, while θ_1 contains only one element, there is no such restriction on θ_2. However, the leader does not know the follower's type. For each agent (leader or follower) n, there is a set of strategies σ_n and a utility function $u_n : \theta_1 \times \theta_2 \times \sigma_1 \times \sigma_2 \rightarrow \Re$. Our goal is *to find the optimal mixed strategy* for the leader to commit to; given that the follower may know this mixed strategy when choosing its strategy.

A Bayesian game can be transformed into a normal-form game using the Harsanyi transformation [7]. Once this is done, existing linear-program(LP)-based methods for finding optimal strategies [5] or the MIP-Nash technique for finding the best Nash equilibrium [15], for normal-form games can be used to find a strategy in the transformed game; this strategy from the transformed game can then be used back in the original Bayesian game.

Given that the Harsanyi transformation is a standard concept in game theory, we describe its key properties when applying to Bayesian Stackelberg games briefly without detailing the actual transformation. Using the Harsanyi technique involves introducing a chance node, that determines the follower's type, thus transforming the leader's incomplete information regarding the follower into imperfect information [3]. The resulting normal-form game matrix generated by the transformation contains the same set of possible actions for the leader as in the original game. However, the set of possible actions for the follower is the cross product of each follower type's set of possible actions. In other words, using the Harsanyi transformation on the Bayesian Stackelberg games we consider results in a normal-form game with the same number of rows as there are leader actions; however the number of columns has increased exponentially, since every combination of actions taken by each follower type is considered as one possible action for the follower in the transformed game.

Fortunately, one key advantage of the DOBSS approach is that it operates directly on the compact Bayesian representation, without requiring the Harsanyi transformation. In particular, DOBSS obtains a decomposition scheme by exploiting the property that the followers are independent of each other. Since the problem is NP-hard, we would not anticipate a simple decomposition. Instead, the key in the DOBSS decomposition scheme is the observation that evaluating the leader strategy against a Harsanyi-transformed game matrix is equivalent to evaluating against each of the game matrices for the individual follower types. This decomposition is analogous to ASAP [12], which can also operate directly on the untransformed Bayesian game to find a high-quality strategy for the leader. However as revealed by our detailed experimental analysis in this paper, ASAP generates infeasible solutions as the problem sizes increase and does not guarantee an optimal solution due to controlled randomization. In addition, our experiments show that ASAP is slower than DOBSS as the number of follower types increases.

DOBSS's other main competitor is the *multiple LPs* method introduced by [5] to compute optimal leader strategies for non-Bayesian games. However, this method faces an exponential explosion when applied to domains of interest in this paper. Furthermore, it is unlikely to be decomposable into a small number of games given that the problem being attacked is NP-hard; DOBSS has the advantage of decomposition, but must work with mixed-integer linear programs (MILPs) rather than LPs. Finally, DOBSS requires solution of only one optimization problem, rather than a series of problems as in the LP method of [5]. Finally, the sequence form [8, 9] provides an alternative compact representation to normal form representation, and has been shown in games like Poker to provide significant speedups in finding equilibrium solutions over approaches based on normal form representations. However, this representation cannot be directly used in our Stackelberg games. In particular, our game assumes that an adversary knows not the specific strategy (e.g. patrolling plan) that an agent will follow, but rather only the agent's mixed strategy. Representing such a commitment to a mixed strategy in a sequence form representation is difficult; it would need to represent all possible mixed strategies in advance. Furthermore, [8, 9] have not focused on computing optimal response in stackelberg games, but rather in only finding equilibria.

3. DOBSS APPROACH

As mentioned earlier, one major advantage of the DOBSS approach is that it operates directly on the compact Bayesian representation. In particular, it exploits the independence of the different follower types to obtain a decomposition scheme. In order to explain the DOBSS approach, in the following subsections, we first define the problem in its most intuitive form as a mixed-integer quadratic program (MIQP), and then show how this problem can be decomposed and then finally show its conversion into an MILP. The model we propose explicitly represents the actions by agent and adversary in the problem solved by the agent, which includes optimality conditions for the adversary's actions. For a detailed discussion of MILPs, please see one of many references on the topic, such as [16].

Note that for a single follower type, we simply take the mixed strategy for the leader that gives the highest payoff when the follower plays a reward-maximizing strategy. We need only to consider the reward-maximizing pure strategies of the followers, since for a given fixed strategy x of the leader, each follower type faces a problem with fixed linear rewards. If a mixed strategy is optimal for the follower, then so are all the pure strategies in the support of that mixed strategy.

3.1 Mixed-Integer Quadratic Program

We begin with the case of a single follower. Let the leader be the row player and the follower the column player. We denote by x the leader's policy, which consists of a vector of the leader's pure strategies. The value x_i is the proportion of times in which pure strategy i is used in the policy. Similarly, q denotes the vector of strategies of the follower. We also denote X and Q the index sets of the leader and follower's pure strategies, respectively. The payoff matrices R and C are defined such that R_{ij} is the reward of the leader and C_{ij} is the reward of the follower when the leader takes pure strategy i and the follower takes pure strategy j.

We first fix the policy of the leader to some policy x. We formulate the optimization problem the follower solves to find its optimal response to x as the following linear program:

$$\begin{array}{ll} \max_q & \sum_{j\in Q}\sum_{i\in X} C_{ij}x_i\ q_j \\ \text{s.t.} & \sum_{j\in Q} q_j = 1 \\ & q_j \geq 0. \end{array} \tag{1}$$

The objective function maximizes the follower's expected reward given x, while the constraints make feasible any mixed strategy q for the follower. It is straightforward that it is optimal to set $q_j = 1$ for a j which has a maximal value of $\sum_{i\in X} C_{ij}x_i$. This is also evident from the dual problem, given by

$$\begin{array}{ll} \min_a & a \\ \text{s.t.} & a \geq \sum_{i\in X} C_{ij}x_i \quad j \in Q, \end{array} \tag{2}$$

which by LP duality has the same optimal solution value. Linear programming optimality conditions characterize the optimal solutions to the follower's problem. These conditions are: primal feasibility constraints in (1), dual feasibility constraints in (2), and complementary slackness

$$q_j\left(a - \sum_{i\in X} C_{ij}x_i\right) = 0 \qquad j \in Q.$$

These conditions show that the follower's maximum reward value, a, is the value obtained for every pure strategy with $q_j > 0$, i.e. in the support of the optimal mixed strategy. Therefore each of these pure strategies is optimal. These conditions will be used in the leader's optimization problem to characterize the optimal follower's response. The leader seeks the solution x that maximizes its own payoff, given that the follower uses an optimal response $q(x)$. Therefore the leader solves the following problem:

$$\begin{array}{ll} \max_x & \sum_{i\in X}\sum_{j\in Q} R_{ij}q(x)_j\ x_i \\ \text{s.t.} & \sum_{i\in X} x_i = 1 \\ & x_i \in [0\ldots 1]. \end{array} \tag{3}$$

Problem (3) maximizes the leader's reward with follower's best response, denoted by vector $q(x)$ for every leader strategy x. We complete this problem by including the characterization of q(x) through linear programming optimality conditions. To simplify writing the complementary slackness condition, we consider only the pure optimal strategies for the follower, which exist always. In fact we note above that given any optimal mixed strategy $q(x)$ all pure strategies in its support are also optimal. This allows us to represent the optimal pure strategies using binary variables and helps to linearize the complementary slackness conditions. The leader's problem then becomes:

$$\begin{array}{ll} \max_{x,q,a} & \sum_{i\in X}\sum_{j\in Q} R_{ij}x_iq_j \\ \text{s.t.} & \sum_{i\in X} x_i = 1 \\ & \sum_{j\in Q} q_j = 1 \\ & 0 \leq (a - \sum_{i\in X} C_{ij}x_i) \leq (1-q_j)M \\ & x_i \in [0\ldots 1] \\ & q_j \in \{0,1\} \\ & a \in \Re\,. \end{array} \tag{4}$$

Here, M is some large constant and a is the follower's maximum reward value as defined in Problem (2). Constraints with free indices mean they are repeated for all values of the index. For example, the next to last constraint means $q_j \in \{0,1\}$ for all $j \in Q$. The first and fourth constraints enforce a feasible mixed policy for the leader, and the second and fifth constraints enforce a feasible pure strategy for the follower. The third constraint enforces dual feasibility of the follower's problem (leftmost inequality) and the complementary slackness constraint for an optimal pure strategy q for the follower (rightmost inequality). In fact, since only one pure strategy can be selected by the follower, say $q_h = 1$, this last constraint enforces that $a = \sum_{i\in X} C_{ih}x_i$ imposing no additional constraint for all other pure strategies which have $q_j = 0$.

We conclude this subsection noting that Problem (4) is an integer program with a non-convex quadratic objective. This model is similar to the one in [5]; however that work solves a different agent problem for each possible value of the adversary's actions, thus eliminating the quadratic objective at the cost of solving multiple problems. There are two challenges in problem (4) if we were to apply it to Harsanyi transformed game given multiple follower types: the potential large number of joint actions of the adversaries and the nonlinear objective function. We show below how this formulation can be decomposed to handle multiple follower types without requiring the Harsanyi transformation. We then explain how to linearize the objective function using a change of variable.

3.2 Decomposed MIQP

The MIQP developed in the previous section handles only one follower. To extend this Stackelberg model to handle multiple follower types we follow a Bayesian approach and assume that there is an a priori probability p^l that a follower of type l will appear, with L denoting the set of follower types. We adapt the notation defined

in the previous section to reason about multiple follower types. We denote by x the vector of strategies of the leader and q^l the vector of strategies of follower $l \in L$. We also denote by X and Q the index sets of leader and follower l's pure strategies, respectively. We also index the payoff matrices of the leader and each of the follower types l by the matrices R^l and C^l. Using this modified notation, we characterize the optimal solution of follower l's problem given the leader's policy x, with the LP optimality conditions:

$$\begin{aligned} \sum_{j \in Q} q_j^l &= 1 \\ a^l - \sum_{i \in X} C_{ij}^l x_i &\geq 0 \\ q_j^l (a^l - \sum_{i \in X} C_{ij}^l x_i) &= 0 \\ q_j^l &\geq 0 \end{aligned}$$

Again, considering only optimal pure strategies for follower l's problem we can linearize the complementarity constraint above. We incorporate these constraints on the leader's problem that selects the optimal policy. Therefore, given *a priori* probabilities p^l, with $l \in L$ of facing each follower type, the leader solves the following problem:

$$\begin{array}{ll} \max_{x,q,a} & \sum_{i \in X} \sum_{l \in L} \sum_{j \in Q} p^l R_{ij}^l x_i q_j^l \\ \text{s.t.} & \sum_i x_i = 1 \\ & \sum_{j \in Q} q_j^l = 1 \\ & 0 \leq (a^l - \sum_{i \in X} C_{ij}^l x_i) \leq (1 - q_j^l) M \\ & x_i \in [0 \dots 1] \\ & q_j^l \in \{0, 1\} \\ & a \in \Re \end{array} \tag{5}$$

Notice that Problem (5) is a decomposed MIQP in the sense that it does not utilize a full-blown Harsanyi transformation; in particular, it essentially solves multiple smaller problems using individual adversaries' payoffs (hence payoffs indexed by l) rather than a single, large, Harsanyi-transformed payoff matrix formed from the cross-product of actions of each follower type.

However, does this decomposition cause any suboptimality? The answer is no. We now show that Problem (5) above is equivalent to Problem (4) with the payoff matrix from the Harsanyi transformation for a Bayesian Stackelberg game (i.e. problem without decomposition). By equivalent we mean that optimal solution values are the same and that an optimal solution to (5) leads to an optimal solution to (4) and vice-versa.

The Harsanyi transformation determines the follower's type according to a given a priori probabilities p^l. For the leader it is as if there is a single follower type, whose action set is the cross product of the actions of every follower type. Each action $\hat{j}$ of the follower in the Harsanyi transformed game *corresponds* to a vector of actions $(j_1, ...j_{|L|})$ in the decomposed game, one action for each follower type. We can refer to these corresponding actions by the pure strategies that take them. For example, consider a pure strategy q in the Harsanyi game which has $q_j = 1$ and $q_h = 0$ for all $h \neq j$ and one pure strategy q^l for each opponent type l which has $q_{j_l}^l = 1$ and $q_h^l = 0$ for all $h \neq j_l$. We say that a Harsanyi pure strategy corresponds to a vector of decomposed pure strategies when their respective actions correspond to each other. The rewards related to these actions have to be weighted by the probabilities of occurrence of each follower. Thus, the reward matrices for the Harsanyi transformation are constructed from the individual reward matrices as follows, as in [7]:

$$R_{ij} = \sum_{l \in L} p^l R_{ij}^l \text{ and } C_{ij} = \sum_{l \in L} p^l C_{ij}^l. \tag{6}$$

PROPOSITION 1. *Problem (5) for a Bayesian game with multiple follower types is equivalent to Problem (4) on the payoff matrices given by the Harsanyi transformation (6).*

Proof: To show the equivalence we show that a feasible solution to (5) leads to a feasible solution to (4) of same objective value or better and vice-versa. This implies the equality in optimal objective value and the correspondence between optimal solutions.

Consider x, q^l, a^l with $l \in L$ a feasible solution to Problem (5). We now construct a feasible solution to (4). From its second constraint and integrality of q we have that for every l there is exactly one j_l such that $q_{j_l}^l = 1$. Let j be the Harsanyi action that corresponds to $(j_1, \dots, j_{|L|})$ and let q be its pure strategy (i.e. q is a strategy in the transformed game where $q_j = 1$, and $q_h = 0$ for all other $h \neq j$). We now show that the objective of (5) equals that of (4) exploiting these corresponding actions. In particular:

$$\begin{aligned} \sum_{i \in X} \sum_{l \in L} p^l x_i \sum_{h \in Q} R_{ih}^l q_h^l &= \sum_{i \in X} x_i \sum_{l \in L} p^l R_{ij_l}^l \\ &= \sum_{i \in X} x_i R_{ij} = \sum_{i \in X} \sum_{h \in Q} x_i R_{ih} q_h \end{aligned}$$

So now we just have to show that x, q, and $a = \sum_{l \in L} p^l a^l$ is feasible for Problem (4). Constraints 1, 2, 4, and 5 in (4) are easily satisfied by the proposed solution. Constraint 3 in (5) means that $\sum_{i \in X} x_i C_{ij_l}^l \geq \sum_{i \in X} x_i C_{ih}^l$, for every $h \in Q$ and $l \in L$, leading to

$$\sum_{i \in X} x_i C_{ij} = \sum_{l \in L} p^l \sum_{i \in X} x_i C_{ij_l}^l \geq \sum_{l \in L} p^l \sum_{i \in X} x_i C_{ih_l}^l = \sum_{i \in X} x_i C_{ih'} \,,$$

for any pure strategy $h_1, \dots, h_{|L|}$ for each of the followers and h' its corresponding pure strategy in the Harsanyi game. We conclude this part by showing that

$$\sum_{i \in X} x_i C_{ij} = \sum_{l \in L} p^l \sum_{i \in X} x_i C_{ij_l}^l = \sum_{l \in L} p^l a^l = a \,.$$

Now we start with (x, q, a) feasible for (4). This means that $q_j = 1$ for some pure action j. Let $(j_1, \dots, j_{|L|})$ be the corresponding actions for each follower l. We show that x, q^l with $q_{j_l}^l = 1$ and $q_h^l = 0$ for $h \neq j_l$, and $a^l = \sum_{i \in X} x_i C_{ij_l}^l$ with $l \in L$ is feasible for (5). By construction this solution satisfies constraints 1, 2, 4, 5 and has a matching objective function. We now show that constraint 3 holds by showing that $\sum_{i \in X} x_i C_{ij_l}^l \geq \sum_{i \in X} x_i C_{ih}^l$ for all $h \in Q$ and $l \in L$. Let us assume it does not. That is, there is an $\hat{l} \in L$ and $\hat{h} \in Q$ such that $\sum_{i \in X} x_i C_{ij_{\hat{l}}}^{\hat{l}} < \sum_{i \in X} x_i C_{i\hat{h}}^{\hat{l}}$. Then by multiplying by $p^{\hat{l}}$ and adding $\sum_{l \neq \hat{l}} p^l \sum_{i \in X} x_i C_{ij_l}^l$ to both sides of the inequality we obtain

$$\sum_{i \in X} x_i C_{ij} < \sum_{i \in X} x_i \left(\sum_{l \neq \hat{l}} p^l C_{ij_l}^l + p^{\hat{l}} C_{i\hat{h}}^{\hat{l}} \right) \,.$$

The right hand side equals $\sum_{i \in X} x_i C_{ih}$ for the pure strategy h that corresponds to $(j_1, \dots, \hat{h}, \dots, j_{|L|})$, which is a contradiction since constraint 3 of (4) implies that $\sum_{i \in X} x_i C_{ij} \geq \sum_{i \in X} x_i C_{ih}$ for all h. ■

3.3 Arriving at DOBSS: Decomposed MILP

We now address the final hurdle for the DOBSS algorithm: eliminating non-linearity of the objective function in the MIQP to generate a MILP. We can linearize the quadratic programming problem 5 through the change of variables $z^l_{ij} = x_i q^l_j$, obtaining the following problem

$$\begin{array}{ll} \max_{q,z,a} & \sum_{i\in X}\sum_{l\in L}\sum_{j\in Q} p^l R^l_{ij} z^l_{ij} \\ \text{s.t.} & \sum_{i\in X}\sum_{j\in Q} z^l_{ij} = 1 \\ & \sum_{j\in Q} z^l_{ij} \le 1 \\ & q^l_j \le \sum_{i\in X} z^l_{ij} \le 1 \\ & \sum_{j\in Q} q^l_j = 1 \\ & 0 \le (a^l - \sum_{i\in X} C^l_{ij}(\sum_{h\in Q} z^l_{ih})) \le (1-q^l_j)M \\ & \sum_{j\in Q} z^l_{ij} = \sum_{j\in Q} z^1_{ij} \\ & z^l_{ij} \in [0\dots 1] \\ & q^l_j \in \{0,1\} \\ & a \in \Re \end{array} \tag{7}$$

PROPOSITION 2. *Problems (5) and (7) are equivalent.*

Proof: Consider x, q^l, a^l with $l \in L$ a feasible solution of (5). We will show that q^l, a^l, $z^l_{ij} = x_i q^l_j$ is a feasible solution of (7) of same objective function value. The equivalence of the objective functions, and constraints 4, 7 and 8 of (7) are satisfied by construction. The fact that $\sum_{j\in Q} z^l_{ij} = x_i$ as $\sum_{j\in Q} q^l_j = 1$ explains constraints 1, 2, 5 and 6 of (7). Constraint 3 of (7) is satisfied because $\sum_{i\in X} z^l_{ij} = q^l_j$.

Lets now consider q^l, z^l, a^l feasible for (7). We will show that q^l, a^l and $x_i = \sum_{j\in Q} z^1_{ij}$ are feasible for (5) with the same objective value. In fact all constraints of (5) are readily satisfied by construction. To see that the objectives match, notice for each l one q^l_j must equal 1 and the rest equal 0. Let us say that $q^l_{j_l} = 1$, then the third constraint in (7) implies that $\sum_{i\in X} z^l_{ij_l} = 1$. This condition and the first constraint in (7) give that $z^l_{ij} = 0$ for all $i \in X$ and all $j \neq j_l$. In particular this implies that

$$x_i = \sum_{j\in Q} z^1_{ij} = z^1_{ij_1} = z^l_{ij_l},$$

the last equality from constraint 6 of (7). Therefore $x_i q^l_j = z^l_{ij_l} q^l_j = z^l_{ij}$. This last equality is because both are 0 when $j \neq j_l$ (and $q^l_j = 1$ when $j = j_l$). This shows that the transformation preserves the objective function value, completing the proof. ■

We can therefore solve this equivalent linear integer program with efficient integer programming packages which can handle problems with thousands of integer variables. We implemented the decomposed MILP and the results are shown in the next section.

We now provide a brief intuition into the computational savings provided by our approach. To have a fair comparison, we analyze the complexity of DOBSS with the competing exact solution approach, namely the Multiple-LPs method by [5]. The DOBSS method achieves an exponential reduction in the problem that must be solved over the multiple-LPs approach due to the following reasons: The multiple-LPs method solves an LP over the exponentially blown harsanyi transformed matrix for each joint strategy of the adversaries (also exponential in number). In contrast, DOBSS solves a problem that has one integer variable per strategy for every adversary. In the proposition below we explicitly compare the work necessary in these two methods.

PROPOSITION 3. *The DOBSS procedure exponentially reduces the problem over the Multiple-LPs approach in the number of adversary types.*

Proof Sketch: Let X be the number of agent actions, Q the number of adversary actions and L the number of adversary types. The DOBSS procedure solves a MILP with XQL continuous variables, QL binary variables, and 4QL+2XL+2L constraints. The size of this MILP then is O(XQL) and the number of feasible integer solutions is Q^L, due to constraint 4 in (7). Solving this problem with a judicious branch and bound procedure will lead in the worst case to a tree with $O(Q^L)$ nodes each requiring the solution of an LP of size $O(XQL)$. Here the size of an LP is the number of variables + number of constraints.

On the other hand the multiple-LPs method needs the Harsanyi transformation. This transformation leads to a game where the agent can take X actions and the joint adversary can take Q^L actions. This method then solves exactly Q^L different LPs, each with X variables and Q^L constraints, i.e. each LP is of size $O(X+Q^L)$.

In summary the total work for DOBSS in the worst case is $O(Q^L XQL)$, given by the number of problems times the size of LPs solved, while the work of the Multiple-LPs method is exactly $O(Q^L(X + Q^L))$. This means that there is an $O(Q^L)$ reduction in the work done by DOBSS. We note that the branch-and-bound procedure seldom explores the entire tree as it uses the bounding procedures to discard sections of the tree which are provably non optimal. The multiple-LPs method on the other hand must solve all Q^L problems. ■

4. EXPERIMENTS

4.1 Experimental Domain

For our experiments, we chose the domain presented in [12], since: (i) it is motivated by patrolling and security applications, which have also motivated this work; (ii) it allows a head-to-head comparison with the ASAP algorithm from [12] which is tested within this domain and shown to be most efficient among its competitors. However, we provide a much more detailed experimental analysis, e.g. experimenting with domain settings that double or triple the number of agent actions.

In particular, [12] focuses on a patrolling domain with robots (such as UAVs [1] or security robots [14]) that must patrol a set of routes at different times. The goal is to come up with a randomized patrolling policy, so that adversaries (e.g. robbers) cannot safely do damage knowing that they will be safe from the patrolling robot. A simplified version of the domain is then presented as a stackelberg game consisting of two players: the security agent (i.e. the patrolling robot or the leader) and the robber (the follower) in a world consisting of m houses, $1\dots m$. The security agent's set of pure strategies consists of possible routes of d houses to patrol (in some order). The security agent can choose a mixed strategy so that the robber will be unsure of exactly where the security agent may patrol, but the robber will know the mixed strategy the security agent has chosen. For example, the robber can observe over time how often the security agent patrols each area. With this knowledge, the robber must choose a single house to rob, although the robber generally takes a long time to rob a house. If the house chosen by the robber is not on the security agent's route, then the robber successfully robs the house. Otherwise, if it is on the security agent's route, then the earlier the house is on the route, the easier it is for the security agent to catch the robber before he finishes robbing it.

The payoffs are modeled with the following variables:

- $v_{y,x}$: value of the goods in house y to the security agent.

- $v_{y,q}$: value of the goods in house y to the robber.
- c_x: reward to the security agent of catching the robber.
- c_q: cost to the robber of getting caught.
- p_y: probability that the security agent can catch the robber at the yth house in the patrol ($p_y < p_{y'} \iff y' < y$).

The security agent's set of possible pure strategies (patrol routes) is denoted by X and includes all d-tuples $i =< w_1, w_2, ..., w_d >$. Each of $w_1 \ldots w_d$ may take values 1 through m (different houses), however, no two elements of the d-tuple are allowed to be equal (the agent is not allowed to return to the same house). The robber's set of possible pure strategies (houses to rob) is denoted by Q and includes all integers $j = 1 \ldots m$. The payoffs (security agent, robber) for pure strategies i, j are:

- $-v_{y,x}, v_{y,q}$, for $j = l \notin i$.
- $p_y c_x + (1-p_y)(-v_{y,x}), -p_y c_q + (1-p_y)(v_{y,q})$, for $j = y \in i$.

With this structure it is possible to model many different types of robbers who have differing motivations; for example, one robber may have a lower cost of getting caught than another, or may value the goods in the various houses differently. To simulate differing types of robbers, a random distribution of varying size was added to the values in the base case described in [12]. All games are normalized so that, for each robber type, the minimum and maximum payoffs to the security agent and robber are 0 and 1, respectively.

4.2 Experimental Results

We performed three sets of experiments all using CPLEX 8.1 solver on a Dell Dimension 8200 machine, Linux Red Hat 7.3 operating system, Pentium 4, 2.40 GHz processor and 1GB RDRAM memory. The first set of experiments compare the runtimes of the following four methods: DOBSS method introduced in this paper for finding the optimal solution, ASAP procedure that provides best policies with limited randomization [12], the multiple-LPs method presented in [5] that provides optimal policies and the MIP-Nash procedure [15] for finding the best Bayes-Nash equilibrium. The multiple-LPs and the MIP-Nash procedures require a normal-form game as input, and so the Harsanyi transformation is required as an initial step. We do not record this preprocessing time here thus giving those other methods an advantage. For this set of experiments, we created games in worlds of two to seven houses with patrols consisting of two houses, constructing payoff tables as described in the previous subsection. We further divided the runtime analysis experiments into two sets: one set of graphs showing runtime results from two, three and fours houses for all the four methods mentioned above and the second set analyzing runtimes of DOBSS and ASAP for five to seven houses since the other two methods were found to be quite slow in these scenarios.

The first set of runtime graphs in Figure 1 shows the runtime results for all the four methods for two, three and four houses. Each runtime value in the graph(s) corresponds to an average over twenty randomly generated scenarios. The x-axis shows the number of follower types the leader faces starting from 1 to 14 adversary types and the y-axis of the graph shows the runtime in seconds on log-scale ranging from .01 to 10000 seconds. The choice of .01 to 10000 is for convenience of representation of log scale(with base 10). All the experiments that were not concluded in 30 minutes (1800 seconds) were cut off.

From the runtime graphs we conclude that the DOBSS and ASAP methods outperform the multiple-LPs and MIP-Nash methods with respect to runtime. We modeled a maximum of fourteen adversary types for all our domains. For the domain with two houses, while the MIP-Nash and multiple-LPs method needed about 1000s for solving the problem with fourteen adversary types, both the DOBSS and ASAP provided solutions in less than 0.1s. Note that

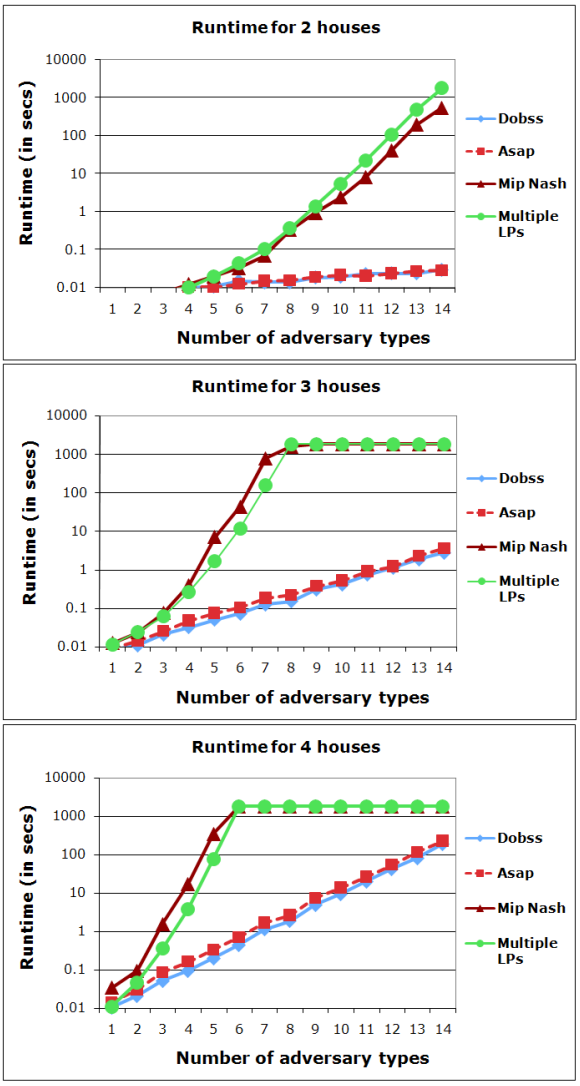

Figure 1: Runtimes(plotted on log scale): DOBSS, ASAP, MIP-Nash and multiple-LP methods

DOBSS provided the optimal solution while ASAP provided the best possible solution with randomization constraints. These randomization constraints also sometimes cause ASAP to *incorrectly* claim solutions to be infeasible, the details of which are presented later on in this section. The runtime for ASAP in all our results is taken as either the time needed to generate an optimal solution or to determine that no feasible solution exists.

The first graph in Figure 1 shows the trends for all these four methods for the domain with two houses. While the runtimes of DOBSS and ASAP show linear increase in runtimes, the other two show an exponential trend. The runtimes of DOBSS and ASAP are themselves exponential since they show a linear increase when plotted on a log-scale graph. Furthermore, they have an exponential speedup over the other two procedures as seen in the graph.

The second graph in Figure 1 presents results for the domain having three houses. Both the MIP-Nash and multiple-LPs could solve this problem only till seven adversary types within the cutoff time of 1800s whereas DOBSS and ASAP could solve the problem for all the fourteen adversary types modeled under 10s. (The cutoff of 1800s is also the reason MIP-Nash and multiple-LPs appear to

have a constant run-time beyond seven adversary types.) Similar trends can be observed in the third graph with a domain of four houses where MIP-Nash and multiple-LPs could solve only till 5 adversary types whereas DOBSS and ASAP could solve till fourteen adversary types within 400s for DOBSS and 500s for ASAP. From this set of three graphs, we conclude that DOBSS and ASAP outperform the other two procedures, by an exponential margin as predicted in our proof earlier.

Between the two fastest methods, i.e DOBSS and ASAP, DOBSS runs faster than ASAP in general. To verify this trend we present a speedup graph in Figure 2 for larger problems i.e., for domains having five, six and seven houses. The x-axis shows the number of adversary types the agent faces and the y-axis represents the speedup obtained by DOBSS over ASAP in percent i.e $100 \cdot runtime(ASAP - DOBSS)/DOBSS$. For example, for the domain with 5 houses and 5 adversary types, the plot shows a speedup of about 95% while for 6 and 7 houses it shows speedups of about 70% and 55% respectively. This implies that if DOBSS needs 100s to solve the problem and has a speedup of 70%, ASAP would need 170s to solve the same problem. All these speedups were calculated as an average of the scenarios (out of the 20 modeled for each instance) that generated optimal solutions (or were declared infeasible for ASAP) within the cutoff time of 1800s. Note that we present results only until 12, 9 and 8 adversary types for 5, 6 and 7 houses respectively, since almost all the 20 instances cross the cutoff times beyond these many adversary types.

From the graph we obtain that DOBSS has a faster algorithm runtime than ASAP in all the cases since there is always a positive speedup. Furthermore, we can notice that the speedups obtained were highest when the number of adversary types are between 2 to 5 and the speedups taper off thereafter. One reason for this trend is as follows: As the number of adversary types increase, the percent of infeasible solutions generated by ASAP increases(as seen in table 3). While DOBSS spends most of its time searching for the optimal solution even if it finds a good solution early-on, ASAP just needs to determine feasibility of the problem whenever it outputs infeasible solutions, hence bringing down the averaged speedups as the number of infeasible solution instances increase — obviously, ASAP is mistakenly determining solutions to be infeasible. (Infeasibility does not completely explain the trend however, and further analysis remains an issue for future work.) Calculating the average speedups over all the adversary scenarios for five, six and seven houses we find that DOBSS has a 62% average speedup over the ASAP method i.e if DOBSS takes 100s, ASAP would need 162s on an average. This quantity becomes significant considering the following issues: (a) ASAP procedure generates infeasible solutions significant number of times while DOBSS is always feasible. (b) DOBSS provides the optimal solution whereas ASAP provides the best solution with limited randomization whenever feasible. The next two sets of results focus on the two issues just presented.

We now introduce our second set of experimental results in Figure 3 to highlight the infeasibility issue. The infeasibility of ASAP is a new result and a significant one given that ASAP is the closest competitor of DOBSS in terms of efficiency — while previous work had just presented the ASAP method, it is our large-scale experiments that have systematically uncovered the issue of infeasibility. In this experiment, the same settings as described above were used. The number of houses was varied between two to seven(columns in the table) and the number of adversary types was varied between one to fourteen(rows in the table). For each fixed number of houses and follower types, twenty scenarios were randomly generated. We ran the ASAP procedure and presented the number of infeasible solutions obtained, as a percentage of all

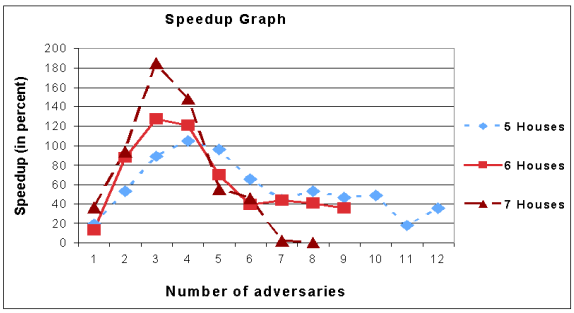

Figure 2: DOBSS vs ASAP

the scenarios tested for each of the fixed number of houses and adversary types. For example, with the 8th adversary type(row numbered 8) and 4 houses(column numbered 4) scenario, ASAP generates 15% infeasible solutions. Note that for the values marked with a star the percentage presented in the table represents an upper bound on the number of infeasible scenarios. In these starred scenarios the ASAP procedure ran out of time in many instances. When ASAP ran out of time, it either indicated infeasibility, in which case it was classified as infeasible solution making it an upper bound (since there might be feasible solution when sufficient time is provided); or it indicated that there was a feasible solution even though it has not found the optimal yet, in which case it was obviously not marked as infeasible. We make the following conclusions about ASAP from the table in Figure 3: (a) In general, given a fixed number of houses, as the number of adversary types increase (i.e from 1 to 14) the percentage of infeasible solutions increase(as we go down the columns). (b) Given a fixed number of adversary types, as the number of houses increase, the percentage of infeasible solutions increase(as we go across the rows). Although there are exceptions to both the conclusions, the general trend is that as the problem size increases (due to increase in either houses or adversary types or both) ASAP tends to generate more infeasible solutions thus making it unsuitable for bigger problems. From the table we obtain that more than 12.5% of the solutions are infeasible for the five house problem when averaged over all the adversary scenarios. This number increases to as high as 18% and 20% on an average for the six and seven house problems. If we perform similar calculations over the last five adversary scenarios i.e., when the number of adversary types are varied from 10 to 14, we obtain 16%, 29% and 25% respectively for the five, six and seven house scenarios. This shows that the ASAP produces more infeasible solutions as the problem size increases. Furthermore, there is no procedure to determine if ASAP will generate a infeasible solution until runtime, thus making the ASAP approach impractical.

Our third set of experiments compared the solution quality provided by all the four methods. Both DOBSS and Multiple-LPs procedure provide the optimal solution and hence are considered equivalent. In Figure 4, we provide a table that shows the quality loss averaged over 20 instances, expressed as a percent loss from the optimal solution(provided by DOBSS), for the ASAP and the MIP-Nash procedures. The averaged results are then presented for all the houses and adversary scenarios as in table 3. The percent loss of quality is defined as $100.quality(DOBSS-x)/DOBSS$, where x is the solution quality of ASAP or MIP-Nash. Each cell (corresponding to a fixed number of houses and columns) contains two numbers. The first number represents the percent of quality loss for ASAP, and the second represents the same for the MIP-Nash procedure. The 'na' in the table indicates that the algorithm was unable to provide any solution in the cutoff time of 1800s while

	2	3	4	5	6	7
1	0	0	0	0	0	10
2	0	0	0	0	5	0
3	0	5	5	15	5	10
4	0	10	15	20	20	20
5	0	10	10	10	5	10
6	0	10	10	10	15	15
7	0	15	20	15	10	20
8	0	5	15	10	30	45
9	0	5	10	15	20	30*
10	5	10	20	5	35*	35*
11	0	5	20	10	35*	40*
12	0	10	20	10	30*	30*
13	0	20	20	25*	25*	0
14	0	20	20	30*	20*	20*

Figure 3: Percent of infeasible solutions for ASAP. Rows represent # of adversary types(1-14), columns represent # of houses(2-7).

Number of adversaries	Number of houses: 2		3		4	
	ASAP	MIP-Nash	ASAP	MIP-Nash	ASAP	MIP-Nash
1	0.00	0.00	0.35	0.00	0.06	11.28
2	0.01	0.00	0.14	0.00	0.09	6.64
3	1.03	0.00	5.14	0.00	5.11	6.52
4	0.02	0.00	10.13	0.00	15.07	5.96
5	0.09	0.00	10.20	0.00	10.12	9.21
6	0.01	0.00	10.10	0.00	10.14	na
7	0.01	0.00	15.35	0.00	20.08	na
8	0.07	0.00	5.23	0.00	15.13	na
9	0.03	0.00	5.12	0.00	10.19	na
10	5.08	0.00	10.08	na	20.12	na
11	0.20	0.00	5.18	na	20.12	na
12	0.49	0.00	10.10	na	20.14	na
13	0.24	0.00	20.05	na	20.14	na
14	0.08	0.00	20.08	na	20.13	na

Figure 4: Quality results for ASAP & MIP-Nash. Rows represent # of adversary types(1-14), columns represent # of houses(2-4).

ASAP generated a solution(feasible or infeasible) in all instances. The quality of infeasible solutions was taken as zero.

As described earlier, rows numbered from 1 to 14 represent the number of adversary types and columns numbered from 2 to 4 represent the number of houses. For example, for 3 houses and 6 adversary types scenario, the quality loss tuple shown in the table is $< 10.1, 0 >$. This means that ASAP has a quality loss of 10.1% while MIP-Nash has 0% quality loss. A quality loss of 10.1% would mean that if DOBSS provided a solution of quality 100 units, the solution quality of ASAP would be 89.9 units. From the table we obtain the following: (a) The quality loss for ASAP is very low for two houses case and increases in general as the number of houses and adversary types increase. The average quality loss was 0.5% over all adversary scenarios for the two house case and increases to 9.1% and 13.3% respectively for three and four houses case. (b) The equilibrium solution provided by the MIP-Nash procedure is also the optimal leader strategy for 2 and 3 houses case; hence the quality loss of 0. The solution quality of the equilibrium is lower than the optimum solution for the four houses case by almost 8% when averaged over all the available data.

From the three sets of experimental results we conclude the following: DOBSS and ASAP are significantly faster than the other procedures with DOBSS being the fastest method. Furthermore, DOBSS provides a feasible exact solution always while ASAP is a heuristic that has low solution quality and also generates infeasible solutions significant number of times. Hence, the need for development of DOBSS, an efficient and exact procedure for solving the Bayesian Stackelberg games.

5. CONCLUSIONS

This paper presents a new exact method called DOBSS for finding the optimal strategy for the leader in a Bayesian Stackelberg game. In these games, one agent (the leader) must commit to a possibly mixed strategy that can be observed by other agents (the followers) before they choose their own strategies. Such games, in which the leader is uncertain about the types of adversary it may face, are extremely valuable in modeling domains involving security, including patrolling, setting up checkpoints, network routing, transportation systems and others; and thus solution techniques such as DOBSS for efficiently solving such games are crucial. The contributions of this paper include the DOBSS algorithm, as well as detailed proofs of correctness of DOBSS, and also importantly, a thorough experimental analysis of performance of DOBSS and its competitors that was previously missing. DOBSS is orders of magnitude faster than the previously published exact method [5]. Compared to the previously published heuristic method [12], it is not only faster, but it avoids the significant problems of infeasibility faced by that method. DOBSS thus represents a significant advance in the state of the art in addressing security domains and is at the heart of the ARMOR system that is currently being tested for security scheduling at the Los Angeles International Airport.

Acknowledgements: This research is supported by the United States Department of Homeland Security through Center for Risk and Economic Analysis of Terrorism Events (CREATE). Sarit Kraus is also affiliated with UMIACS.

6. REFERENCES

[1] R. W. Beard and T. W. McLain. Multiple UAV cooperative search under collision avoidance and limited range communication constraints. In *IEEE CDC*, 2003.

[2] G. Brown, M. Carlyle, J. Salmeron, and K. Wood. Defending critical infrastructure. *Interfaces*, 36(6):530–544, 2006.

[3] J. Brynielsson and S. Arnborg. Bayesian games for threat prediction and situation analysis. In *FUSION*, 2004.

[4] J. Cardinal, M. Labbe, S. Langerman, and B. Palop. Pricing of geometric transportation networks. In *17th Canadian Conference on Computational Geometry*, 2005.

[5] V. Conitzer and T. Sandholm. Computing the optimal strategy to commit to. In *EC*, 2006.

[6] D. Fudenberg and J. Tirole. *Game Theory*. MIT Press, 1991.

[7] J. C. Harsanyi and R. Selten. A generalized Nash solution for two-person bargaining games with incomplete information. *Management Science*, 18(5):80–106, 1972.

[8] D. Koller, N. Megiddo, and B. von Stengel. Fast algorithms for finding randomized strategies in game trees. In *26th ACM Symposium on Theory of Computing*, 1994.

[9] D. Koller and A. Pfeffer. Representations and solutions for game-theoretic problems. *AI*, 94(1-2):167–215, 1997.

[10] Y. A. Korilis, A. A. Lazar, and A. Orda. Achieving network optima using stackelberg routing strategies. In *IEEE/ACM Transactions on Networking*, 1997.

[11] A. Murr. Random checks. In *Newsweek National News. http://www.newsweek.com/id/43401*, 28 Sept. 2007.

[12] P. Paruchuri, J. P. Pearce, M. Tambe, F. Ordonez, and S. Kraus. An efficient heuristic approach for security against multiple adversaries. In *AAMAS*, 2007.

[13] J. Pita, M. Jain, J. Marecki, F. Ordonez, C. Portway, M. Tambe, C. Western, P. Paruchuri, and S. Kraus. Deployed armor protection: The application of a game theoretic model for security at the los angeles internation airport. In *AAMAS Industry Track*, 2008.

[14] S. Ruan, C. Meirina, F. Yu, K. R. Pattipati, and R. L. Popp. Patrolling in a stochastic environment. In *10th Intl. Command and Control Research and Tech. Symp.*, 2005.

[15] T. Sandholm, A. Gilpin, and V. Conitzer. Mixed-integer programming methods for finding nash equilibria. In *AAAI*, 2005.

[16] L. A. Wolsey. *Integer Programming*. John Wiley, 1998.