Deployed ARMOR Protection: The Application of a Game Theoretic Model for Security at the Los Angeles International Airport

James Pita, Manish Jain, Janusz Marecki, Fernando Ordóñez, Christopher Portway,
Milind Tambe, Craig Western, Praveen Paruchuri*, Sarit Kraus**
University of Southern California, Los Angeles, CA 90089
*Intelligent Automation, Inc., RockVille, MD 20855
**Bar-Ilan University, Ramat-Gan 52900, Israel and
Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742

ABSTRACT

Security at major locations of economic or political importance is a key concern around the world, particularly given the threat of terrorism. Limited security resources prevent full security coverage at all times, which allows adversaries to observe and exploit patterns in selective patrolling or monitoring, e.g. they can plan an attack avoiding existing patrols. Hence, randomized patrolling or monitoring is important, but randomization must provide distinct weights to different actions based on their complex costs and benefits. To this end, this paper describes a promising transition of the latest in multi-agent algorithms – in fact, an algorithm that represents a culmination of research presented at AAMAS – into a deployed application. In particular, it describes a software assistant agent called ARMOR (Assistant for Randomized Monitoring over Routes) that casts this patrolling/monitoring problem as a Bayesian Stackelberg game, allowing the agent to appropriately weigh the different actions in randomization, as well as uncertainty over adversary types. ARMOR combines three key features: (i) It uses the fastest known solver for Bayesian Stackelberg games called DOBSS, where the dominant mixed strategies enable randomization; (ii) Its mixed-initiative based interface allows users to occasionally adjust or override the automated schedule based on their local constraints; (iii) It alerts the users if mixed-initiative overrides appear to degrade the overall desired randomization. ARMOR has been successfully deployed since August 2007 at the Los Angeles International Airport (LAX) to randomize checkpoints on the roadways entering the airport and canine patrol routes within the airport terminals. This paper examines the information, design choices, challenges, and evaluation that went into designing ARMOR.

1. INTRODUCTION

Protecting national infrastructure such as airports, historical landmarks, or locations of political or economic importance is a challenging task for police and security agencies around the world; a challenge that is exacerbated by the threat of terrorism. The protection of important locations includes tasks such as monitoring all entrances or inbound roads and checking inbound traffic. However, limited resources imply that it is typically impossible to provide full security coverage at all times. Furthermore, adversaries can observe security arrangements over time and exploit any predictable patterns to their advantage. Randomizing schedules for patrolling, checking, or monitoring is thus an important tool in the police arsenal to avoid the vulnerability that comes with predictability. Even beyond protecting infrastructure, randomized patrolling is important in tasks varying from security on university campuses to normal police beats, to border or maritime security [2, 9, 13, 16].

This paper focuses on a deployed software assistant agent that can aid police or other security agencies in randomizing their security schedules. We face at least three key challenges in building such a software assistant. First, the assistant must provide quality guarantees in randomization by appropriately weighing the costs and benefits of the different options available. For example, if an attack on one part of an infrastructure will cause economic damage while an attack on another could potentially cost human lives, we must weigh the two options differently — giving higher weight (probability) to guarding the latter. Second, the assistant must address the uncertainty in information that security forces have about the adversary. Third, the assistant must enable a mixed-initiative interaction with potential users rather than dictating a schedule; the assistant may be unaware of users' real-world constraints and hence users must be able to shape the schedule development.

We have addressed these challenges in a software assistant agent called ARMOR (Assistant for Randomized Monitoring over Routes). Based on game-theoretic principles, ARMOR combines three key features to address each of the challenges outlined above. Game theory is a well-established foundational principle within multi-agent systems to reason about multiple agents each pursuing their own interests [6]. We build on these game theoretic foundations to reason about two agents – the police force and their adversary – in providing a method of randomization. In particular, the main contribution of our paper is mapping the problem of security scheduling as a Bayesian Stackelberg game [4] and solving it via the fastest optimal algorithm for such games, addressing the first two challenges. While a Bayesian game allows us to address uncertainty over adversary types, by optimally solving such Bayesian Stackelberg games (we obtain optimal randomized strategies as solutions), ARMOR provides quality guarantees on the schedules generated. The algorithm used builds on several years of research reported in the AAMAS conference main track and AAMAS workshops [14, 15, 13]. Indeed, we reported on an approximate algorithm called ASAP for solving Bayesian Stackeberg games at AAMAS 2007 [13]. ARMOR employs an algorithm that is a logical culmination of this line of research; in particular, ARMOR relies on an algo-

Padgham, Parkes, Mueller and Parsons (eds.): Proc. of AAMAS 2008, Estoril, May, 12-16., 2008, Portugal, pp. XXX-XXX. **Cite as:** Title, Author(s), Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systens, pp. XXX-XXX, 2008.

rithm called DOBSS (Decomposed Optimal Bayesian Stackelberg Solver) [12], which is superior to its competitors including ASAP, and it provides optimal solutions rather than approximations. The third challenge is addresed by ARMOR's use of a mixed-initiative based interface, where users are allowed to graphically enter different constraints to shape the schedule generated. ARMOR is thus a collaborative assistant that iterates over generated schedules rather than a rigid one-shot scheduler. ARMOR also alerts users in case overrides deteriorate the schedule quality below a given threshold. This can include repeatedly scheduling an action which has a low probability in the optimal mixed strategy, or repeatedly forbidding an action which has been assigned a high probabiliity.

ARMOR thus represents a very promising transition of multi-agent research into a deployed application. ARMOR has been successfully deployed on a trial basis since August 2007 [11] at the Los Angeles International Airport (LAX) to assist the Los Angeles World Airport (LAWA) police in randomized scheduling of checkpoints, and since November 2007 for generating randomized patrolling schedules for canine units. In particular, it assists police in determining where to randomly set up checkpoints and to randomly allocate canines to terminals.

2. RELATED WORK

The key contribution of this paper is the development of a game theoretic security scheduler, named ARMOR, for improving security at the Los Angeles International Airport. The novelty of our work lies in modeling the security problem as a Bayesian Stackelberg game [6, 13] and applying an efficient algorithm named DOBSS to find the optimal security schedule. In previous work, it has been shown that finding an optimal solution for Bayesian Stackelberg game with multiple follower types is NP-hard [4]. Two different approaches have been presented previously to find solutions to the Bayesian Stackelberg games efficiently. The first is an exact approach named the multiple LPs method [4]. This approach needs the conversion of the Bayesian game into a normal-form game using the Harsanyi transformation [7]; thus, it loses its compact structure. The second approach, named ASAP, does not need the Harsanyi transformation [13], but it provides an approximate solution. DOBSS outperforms the multiple LPs method because it does not need the Harsanyi tranformation thus gaining exponential speedups. DOBSS is also superior to the ASAP approach as it provides an exact solution by optimally solving the problem at hand. We provide an experimental comparison of these algorithms in Section 6; note that these algorithms had earlier been investigated without the context of a specific infrastructure security application as used in this paper. In contrast, Brown et al. [3] do specifically apply Stackelberg games for defending critical infrastructure. However, they consider a single adversary type (not a Bayesian game) and with ARMOR we have taken the extra step of actually solving and deploying the solutions to the created Bayesian Stackelberg games at LAX.

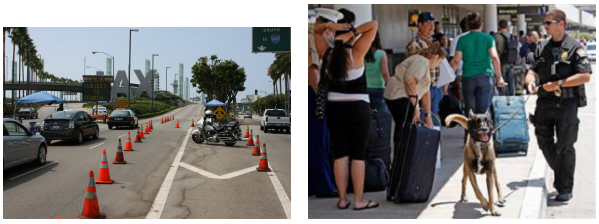
The patrolling problem itself has received significant attention in multi-agent literature due to its wide variety of applications ranging from robot patrol to border patrolling of large areas [16, 9, 2]. The key idea behind the policies provided by these techniques is randomization, which decreases the amount of information given to an adversary. However, no specific algorithm/procedure has been provided for the generation of randomized policies; hence, they can lead to highly suboptimal policies. One exception is Paruchuri et al and their early work [15], which provides algorithms for analyzing randomization-reward trade offs, but they do not model any adversaries. On the other hand, DOBSS provides policies whose randomization is determined by the payoff structure of game matrices; thus DOBSS provides us optimal randomized policies while accounting for multiple adversary models.

While ARMOR is a game theoretic security scheduler, there are many other competing non-game theoretic tools in use for related applications. For example, the "Hypercube Queueing Model" [8] based on queuing theory depicts the detailed spatial operation of urban police departments and emergency medical services and has found application in police beat design, allocation of patrolling time, etc. However, this model does not take specific adversary models into account; ARMOR, however, tailors policies to combat various potential adversaries. SIMCORE's airport simulation software [10] provides simulation and analysis of activities at airports like baggage handling, passenger flow, etc. However, this simulation does not focus on security scheduling, unlike ARMOR. Many other AI based techniques are making their way for making airports safer, but many of these are based on deployment of new hardware such as cameras that automatically detect intruders [5], while our work focuses on making efficient use of existing resources such as number of police personnel, canine units etc.

3. SECURITY DOMAIN DESCRIPTION

We will now describe the specific challenges in the security problems faced by the LAWA police. LAX is the fifth busiest airport in the United States, the largest destination airport in the United States, and serves 60-70 million passengers per year [1, 17]. LAX is unfortunately also suspected to be a prime terrorist target on the west coast of the United States, with multiple arrests of plotters attempting to attack LAX [17]. To protect LAX, LAWA police has designed a security system that utilizes multiple rings of protection. As is evident to anyone traveling through an airport, these rings include such things as vehicular checkpoints, police units patrolling the roads to the terminals and inside the terminals (with canines) and security screening and bag checks for passengers. There are unfortunately not enough resources (police officers) to monitor every single event at the airport; given its size and number of passengers served, such a level of screening would require considerably more personnel and cause greater delays to travelers. Thus, assuming that all checkpoints and terminals are not being monitored at all times, setting up available checkpoints, canine units or other patrols on deterministic schedules allows adversaries to learn the schedules and plot an attack that avoids the police checkpoints and patrols, which makes deterministic schedules ineffective.

Randomization offers a solution here. In particular, from among all the security measures that randomization could be applied to, LAWA police have so far posed two crucial problems to us. First, given that there are many roads leading into LAX, where and when they should set up checkpoints to check cars driving into LAX. For example, Figure 1(a) shows a vehicular checkpoint set up on a road inbound towards LAX. Police officers examine cars that drive by, and if any car appears suspicious, they do a more detailed inspection of that car. LAWA police wished to obtain a randomized schedule for such checkpoints for a particular time frame. For example, if we are to set up two checkpoints, and the timeframe of interest is 8 AM to 11 AM, then a candidate schedule may suggest to the police that on Monday, checkpoints should be placed on route 1 and route 2, whereas on Tuesday during the same time slot, they should be on route 1 and 3, and so on. Second, LAWA police wished to obtain an assignment of canines to patrol routes through the terminals inside LAX. For example, if there are three canine units available, a possible assignment may be to place canines on terminals 1, 3, and 6 on the first day, but terminal 2, 4, and 6 on another day and so on based on the available information. Figure 1(b) illustrates a canine unit on patrol at LAX.



(a) LAX Checkpoint (b) Canine Patrol

Figure 1: LAX Security

Given these problems, our analysis revealed the following key challenges: (i) potential attackers can observe security forces' schedules over time and then choose their attack strategy — the fact that the adversary acts with knowledge of the security forces' schedule makes deterministic schedules highly susceptible to attack; (ii) there is unknown and uncertain information regarding the types of adversary we may face; (iii) although randomization helps eliminate deterministic patterns, it must also account for the different costs and benefits associated with particular targets.

4. APPROACH

We modeled the decisions of setting checkpoints or canine patrol routes at the LAX airport as Bayesian Stackelberg games. These games allow us accomplish three important tasks, meeting the challenges outlined in the previous section: (i) they model the fact that an adversary acts with knowledge of security forces' schedules, and thus randomize schedules appropriately; (ii) they allow us to define multiple adversary types, meeting the challenge of our uncertain information about our adversaries; (iii) they enable us to weigh the significance of different targets differently. Since Bayesian Stackelberg games address the challenges posed by our domain, they are at the heart of generating meaningfully randomized schedules. From this point we will explain what a Bayesian Stackelberg game consists of and how we use DOBSS to optimally solve the problem at hand. We then explain how an LAX security problem can be mapped on to Bayesian Stackelberg games.

4.1 Bayesian Stackelberg game

In a Stackelberg game, a leader commits to a strategy first, and then a follower selfishly optimizes its reward, *considering the action chosen by the leader.* 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. The leader is the row player and the follower is the column player. 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, 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.

	C	D
A	2,1	4,0
B	1,0	3,2

Table 1: Payoff table for example normal form game.

We now explain a Bayesian Stackelberg game. 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 the set of possible types for follower. For the security games of interest in this paper, we assume that there is only one leader type (e.g. only one police force), although there are multiple follower types (e.g. multiple adversary types trying to infiltrate security). 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.

4.2 DOBSS

We now briefly describe the DOBSS algorithm to solve Bayesian Stackelberg games. We note that a more detailed description of this algorithm is the subject of a submitted parallel paper within the main track. Here we provide a brief description of the model and how it relates to a security domain. As mentioned earlier, the concrete novel contribution of this paper is mapping our real-world security problem into a Bayesian Stackelberg game, applying DOBSS to this real-world airport security domain, and finally embedding DOBSS in the overall ARMOR system which provides many features to allow smooth operationalization.

One key advantage of the DOBSS approach is that it operates directly on the Bayesian representation, without requiring the Harsanyi transformation. In particular, DOBSS obtains a decomposition scheme by exploiting the property that follower types are independent of each other. The key to the DOBSS decomposition 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.

We first present DOBSS in its most intuitive form as a Mixed-Integer Quadratic Program (MIQP); we then present a linearized equivalent Mixed-Integer Linear Program (MILP). The model we propose explicitly represents the actions by leader and the optimal actions for the follower types in the problem solved by the agent. Note that we need to consider only the reward-maximizing pure strategies of the follower types, since for a given fixed mixed 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 support of that mixed strategy.

We denote by x the leader's policy, which consists of a vector of probability distributions over the leader's pure strategies. Hence, the value x_i is the proportion of times in which pure strategy i is used in the policy. We denote by q^l the vector of strategies of follower type $l \in L$. 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. Let M be a large positive number. Given *a priori* probabilities p^l, with $l \in L$, of facing each follower type the leader solves the following:

$$
\begin{array}{ll}
\max_{x,q,a} & \sum_{i \in X} \sum_{l \in L} \sum_{j \in Q} p^l R^l_{ij} x_i q^l_j \\
\text{s.t.} & \sum_{i \in X} x_i = 1 \\
& \sum_{j \in Q} q^l_j = 1 \\
& 0 \leq (a^l - \sum_{i \in X} C^l_{ij} x_i) \leq (1 - q^l_j) M \\
& x_i \in [0 \ldots 1] \\
& q^l_j \in \{0, 1\} \\
& a \in \Re
\end{array}
\qquad (1)
$$

Here for a set of leader's actions x and actions for each follower q^l, the objective represents the expected reward for the agent considering the a-priori distribution over different follower types p^l. The first and the fourth constraints define the set of feasible solutions x as a probability distribution over the set of actions X. Constraints 2 and 5 limit the vector of actions of follower type l, q^l to be a pure distribution over the set Q (that is each q^l has exactly one coordinate equal to one and the rest equal to zero). The two inequalities in constraint 3 ensure that $q_j^l = 1$ only for a strategy j that is optimal for follower type l. Indeed this is a linearized form of the optimality conditions for the linear programming problem solved by each follower type. We explain these constraints as follows: note that the leftmost inequality ensures that for all $j \in Q$, $a^l \geq \sum_{i \in X} C_{ij}^l x_i$. This means that given the leader's vector x, a^l is an upper bound on follower type l's reward for any action. The rightmost inequality is inactive for every action where $q_j^l = 0$, since M is a large positive quantity. For the action that has $q_j^l = 1$ this inequality states that the adversary's payoff for this action must be $\geq a^l$, which combined with the previous inequality shows that this action must be optimal for follower type l.

We can linearize the quadratic programming problem 1 through the change of variables $z_{ij}^l = x_i q_j^l$, thus obtaining the following mixed integer linear programming problem:

$$\begin{array}{ll} \max_{q,z,a} & \sum_{i \in X} \sum_{l \in L} \sum_{j \in Q} p^l R_{ij}^l z_{ij}^l \\ \text{s.t.} & \sum_{i \in X} \sum_{j \in Q} z_{ij}^l = 1 \\ & \sum_{j \in Q} z_{ij}^l \leq 1 \\ & q_j^l \leq \sum_{i \in X} z_{ij}^l \leq 1 \\ & \sum_{j \in Q} q_j^l = 1 \\ & 0 \leq (a^l - \sum_{i \in X} C_{ij}^l (\sum_{h \in Q} z_{ih}^l)) \leq (1 - q_j^l) M \\ & \sum_{j \in Q} z_{ij}^l = \sum_{j \in Q} z_{ij}^1 \\ & z_{ij}^l \in [0 \dots 1] \\ & q_j^l \in \{0, 1\} \\ & a \in \Re \end{array} \tag{2}$$

DOBSS refers to this equivalent mixed integer linear program, which can be solved with efficient integer programming packages.

4.3 Bayesian Stackelberg Game for the Los Angeles International Airport

We now illustrate how the security problems set forth by LAWA police, i.e. where and when to deploy checkpoints and canines, can be cast in terms of a Bayesian Stackelberg game. We focus on the checkpoint problem for illustration, but the case of the canine problem is similar. At LAX, there are a specific number of inbound roads on which to set up checkpoints, say roads 1 through n, and LAWA police have to pick a subset of those roads to place checkpoints on prior to adversaries selecting which roads to attack. We assume that there are m different types of adversaries, each with different attack capabilities, planning constraints, and financial ability. Each adversary type observes the LAWA-police checkpoint policy and then decides where to attack. Since adversaries can observe the LAWA police policy before deciding their actions, this situation can be modeled via a Stackelberg game with the police as the leader.

In this setting the set X of possible actions for LAWA police is the set of possible checkpoint combinations. If, for instance, LAWA police were setting up one checkpoint then $X = \{1, \dots, n\}$. If LAWA police were setting up a combination of two checkpoints, then $X = \{(1,2), (1,3) ... (n-1, n)\}$, i.e. all combinations of two checkpoints. Each adversary type $l \in L = \{1, \dots, m\}$ can decide to attack one of the n roads or maybe not attack at all (none), so its set of actions is $Q = \{1, \dots, n, none\}$. If LAWA police select road i to place a checkpoint on and adversary type $l \in L$ selects road j to attack then the agent receives a reward R_{ij}^l and the adversary receives a reward C_{ij}^l. These reward values vary based on three considerations: (i) the chance that the LAWA police checkpoint will catch the adversary on a particular inbound road; (ii) the damage the adversary will cause if it attacks via a particular inbound road; (iii) type of adversary, i.e. adversary capability. If LAWA police catch the adversary when $i = j$ we make R_{ij}^l a large positive value and C_{ij}^l a large negative value. However, the probability of catching the adversary at a checkpoint is based on the volume of traffic through the checkpoint (significant traffic will increase the difficulty of catching the adversary), which is an input to the system. If the LAWA police are unable to catch the adversary, then the adversary may succeed, i.e. we make R_{ij}^l a large negative value and C_{ij}^l a large positive value. Certainly, if the adversary attacks via an inbound road where no checkpoint was set up, there is no chance that the police will catch the adversary. The magnitude of R_{ij}^l and C_{ij}^l vary based on the adversary's potential target, given the road from which the adversary attacks. Some roads lead to higher valued targets for the adversary than others. The game is not a zero sum game however, as even if the adversary is caught, the adversary may benefit due to publicity.

The reason we consider a Bayesian Stackelberg game is because LAWA police face multiple adversary types. Thus, differing values of the reward matrices across the different adversary types $l \in L$ represent the different objectives and valuations of the different attackers (e.g. smugglers, criminals, terrorists). For example, a hard-core, well-financed adversary could inflict significant damage on LAX; thus, the negative rewards to the LAWA police are much higher in magnitude than an amatuer attacker who may not have sufficient resources to carry out a large-scale attack. Currently we model two types of adversary LAWA may face. A 20-80 split of probability implies that while there is a 20% chance that the LAWA police face the former type of adversary, there is an 80% chance that they face an amatuer attacker. Our experimental data provides detailed results about the sensitivity of our algorithms to the probability distributions over different adversary types. Given these two adversary types the largest game we have constructed, which was done for canine deployment, consisted of 784 actions for the LAWA police (when multiple canine units were active) for the eight possible terminals within the airport and 9 actions per adversary type (one for a possible attack on each terminal, and one for $none$).

5. SYSTEM ARCHITECTURE

There are two separate versions of ARMOR, ARMOR-checkpoint and ARMOR-canine. While in the following we focus on ARMOR-checkpoint for illustration, both these versions use the same underlying architecture with different inputs. As shown in Figure 2, this architecture consists of a front-end and a back-end, integrating four key components: (i) a front-end interface for user interaction; (ii) a method for creating Bayesian Stackelberg game matrices; (iii) an implementation of DOBSS; (iv) a method for producing suggested schedules for the user. They also contain two major forms of external input. First, they allow for direct user input into the system through the interface. Second, they allow for file input of relevant information for checkpoints or canines, such as traffic/passenger volume by time of day, which can greatly affect the security measures taken and the values of certain actions. At this point we will discuss in detail what each component consists of and how they interact with each other.

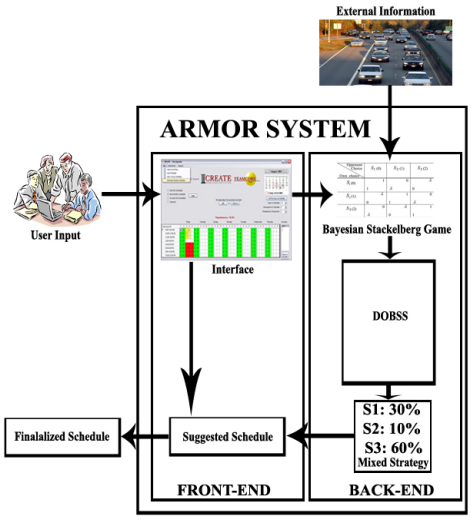


Figure 2: ARMOR System Flow Diagram

5.1 Interface

The ARMOR interface, seen in Figure 3, consists of a file menu, options for local constraints, options to alter the action space, a monthly calendar and a main spreadsheet to view any day(s) from the calendar. Together these components create a working interface that meets all the requirements set forth by LAWA officers for checkpoint and canine deployment at LAX.

The base of the interface is designed around six possible adjustable options; three of them alter the action space and three impose local constraints. The three options to alter the action space are the following: (i) number of checkpoints allowed during a particular timeslot; (ii) time interval of each timeslot; (iii) number of days to schedule over. For each given timeslot, we construct a new game. As discussed in Section 4.4, given knowledge of the total number of inbound roads, the number of checkpoints allowed during that timeslot determines the available actions for the LAWA police, whereas the action space of the adversary is determined as discussed in Section 4.4 by the number of inbound roads. Thus, we can set up the foundation for the Bayesian Stackelberg game by providing all the actions possible in the game. Once the action space has been generated, it can be sent to the back-end to be set up as a Bayesian Stackelberg game, solved, and returned as a suggested schedule, which is displayed to the user via the spreadsheet. The third option determines how many iterations of the game will be played (as it determines the number of days to schedule over).

Once the game is solved, there are three options that serve to restrict certain actions in the generated schedule: (i) forced checkpoint; (ii) forbidden checkpoint; (iii) at least one checkpoint. These constraints are intended to be used sparingly to accommodate situations where a user, faced with exceptional circumstances and extra knowledge, wishes to modify the output of the game. The user may impose these restrictions by forcing specific actions in the schedule. In particular, the "forced checkpoint" option schedules a checkpoint at a specific time on a specific day. The "forbidden checkpoint" option designates a specific time on a specific day when a checkpoint should not be scheduled. Finally, the "at least one checkpoint" option designates a set of time slots and ensures that a checkpoint is scheduled in at least one of the slots. We will return to these constraints in Section 5.3.

The spreadsheet in the interface serves as the main mechanism

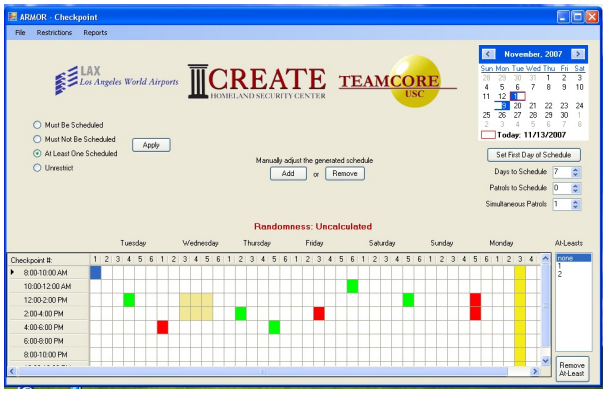


Figure 3: ARMOR Interface

for viewing, altering, and constraining schedules. The columns correspond to the possible checkpoints, and the rows correspond to the time frames in which to schedule them. Up to a full week can be viewed within the spreadsheet at a single time with each day being marked as seen in Figure 3. Once a particular day is in view, the user can assign to that day any constraints that they desire. Each constraint is represented by a specific color within the spreadsheet, namely green, red, and yellow for forced, forbidden, and at least constraints respectively.

5.2 Matrix Generation and DOBSS

Given the submitted user information, the system must create a meaningful Bayesian Stackelberg game matrix as suggested in Section 4.3. The previous section illustrates the generation of the action space in this game. Based on the pre-specified rewards as discussed in Section 4.3, we can provide the rewards for the LAWA police and the adversaries to generate a game matrix for each adversary type. After the final game matrices are constructed for each adversary type, they are sent to the DOBSS implementation, which chooses the optimal mixed strategy over the current action space.

To demonstrate the process, assume there are three possible inbound roads or checkpoint locations (A, B, C), one possible timeslot to schedule over, and two checkpoints available for scheduling. Given this scenario, the unique combinations possible include scheduling checkpoints A and B, A and C, and B and C, over the given time frame. We will assume that checkpoints A and B are highly valuable while C, although not completely invaluable, has a very low value. Based on this information, a likely mixed strategy generated by DOBSS would be to assign a high probability to choosing action A and B, say seventy percent, and a low probability to both the other actions, say fifteen percent each. Whatever the mixed strategy actually comes out to be, it is the optimal strategy a user could take to maximize security based on the given information. This mixed strategy is then stored and used for the actual schedule generation.

5.3 Mixed Strategy and Schedule Generation

Once an optimal mixed strategy has been chosen by DOBSS and stored within the system, a particular combination of actions must be chosen to be displayed to the user. Consider our example from the previous section involving three possibilities (checkpoints A and B, A and C, B and C)) and their probabilities of 70%, 15% and 15%. Knowing this probability distribution, we can formulate a method to randomly select between the combinations with the given probabilities. Each time a selection is made, that combination is sent to the user interface to be reviewed by the user as

necessary. So, if for instance combination one was chosen, the user would see checkpoint A and B as scheduled for the given timeslot.

In rare cases, as mentioned in Section 5.1, a user may have forbidden a checkpoint, or required a checkpoint. ARMOR accommodates such user directives when creating its schedule, e.g. if checkpoint C is forbidden, then all the probability in our example shifts to the combination A and B. Unfortunately, by using this capability frequently (e.g. frequent use of forbidden and required checkpoints), a user can completely alter the mixed strategy produced as the output of DOBSS, defeating DOBSS's guarantee of optimality. To avoid such a possibility, ARMOR incorporates certain alerts (warnings) to encourage non-interference in its schedule generation. For example, if a combination has zero or very low probability of being chosen and the user has forced that checkpoint combination to occur, ARMOR will alert the user. Similarly, if a combination has a very high likelihood and the user has forbidden that event, ARMOR will again alert the user. However, ARMOR only alerts the user; it does not autonomously remove the user's constraints. Resolving more subtle interactions between the user imposed constraints and DOBSS's output strategy remains an issue for future work.

When a schedule is presented to the user with alerts as mentioned above, the user may alter the schedule by altering the forbidden/required checkpoints, or possibly by directly altering the schedule. Both possibilities are accomodated in ARMOR. If the user simply adds or removes constraints, ARMOR can create a new schedule. Once the schedule is finalized, it can be saved for actual use, thus completing the system cycle. This full process was designed to specifically meet the requirements at LAX for checkpoint and canine allocation.

6. DESIGN CHALLENGES

Designing and deploying the ARMOR software on a trial basis at LAX posed numerous challenges and problems to our research group. We outline some key lessons learned during the design and deployment of ARMOR:

- *Importance of tools for randomization:* There is a critical need for randomization in security operations. Security officials are aware that requiring humans to generate randomized schedules is unsatisfactory because as psychological studies have often shown [18], humans have difficulty randomizing, and also they can fall into predictable patterns. Instead, mathematical randomization that appropriately weighs the costs and benefits of different action, and randomizes with appropriate weights leads to improved results. Security officials were hence extremely enthusiastic in their reception of our research, and eager to apply it to their domain.

- *Importance of manual schedule overrides:* While ARMOR incorporates all the knowledge that we could obtain from LAWA police and provides the best output possible, it may not be aware of dynamic developments on the ground. For example, police officers may have very specific intelligence for requiring a checkpoint on a particular inbound road. Hence, it was crucial to allow LAWA police officers (in rare instances when it is necessary) to manually override the schedule provided.

- *Importance of providing police officers with operational flexibility:* When initially generating schedules for canine patrols, we created a very detailed schedule, micro-managing the patrols. This did not get as positive a reception from the officers. Instead, an abstract schedule that afforded the officers some flexibility to respond to dynamic situation on the ground was better received.

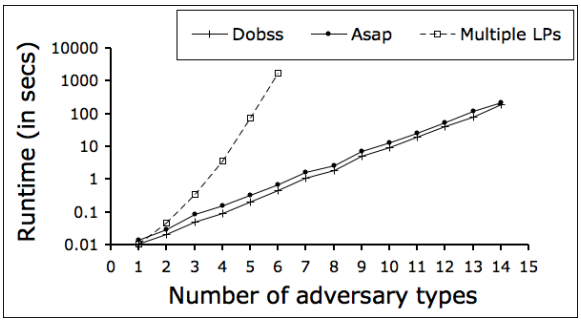


Figure 4: Runtimes: DOBSS, ASAP and multiple-LP methods

7. EXPERIMENTAL RESULTS

Our experimental results explore the runtime efficiency of DOBSS (in Section 7.1) and evaluate the solution quality and implementation of the ARMOR system (in Section 7.2).

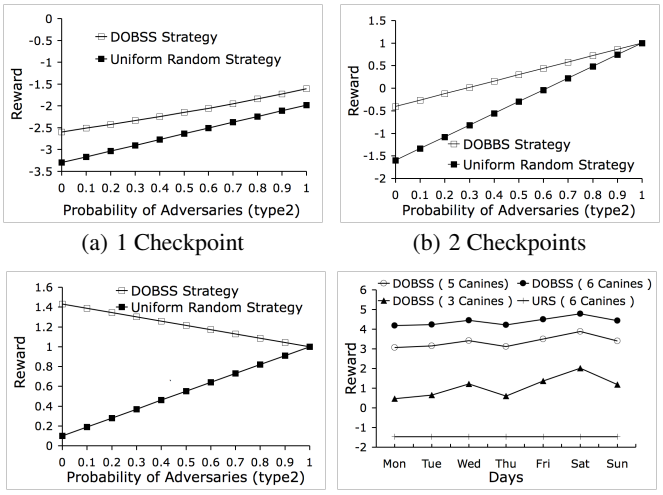
7.1 Runtime Analysis

Here we compare the runtime results of the various procedures available for solving Bayesian Stackelberg games on a benchmarking domain. The aim of this analysis is to show that DOBSS is indeed the most suitable procedure for application to real domains such as the LAX canine and checkpoint allocation. To that end, we used the set of readymade suite of problem instances presented in [13]. These problem instances are based on a patrolling domain that allows for the flexibility of having multiple adversary types and scalability of agent actions. It is thus similar to the challenges we face in our problems at LAX, but it allows for easy scalability to understand the impact of scale on performance.

In Figure 4 we summarize the runtime results for our Bayesian games using DOBSS and the two other competing techniques, i.e. ASAP and Multiple LPs, described in Section 2. We tested our results on Bayesian games with number of adversary types varying between 1 to 14. Each game between the agent and one adversary type is modeled as a normal form game. Thus, there are 14 normal form games designed for the game between an agent and the various adversary types for the base case. The size of each of these normal form games is (12,4) corresponding to 12 strategies for the agent and 4 for the adversary. We then created 19 randomly generated instances of this base case to obtain averaged results.

The x-axis in Figure 4 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 graph we summarize that DOBSS and ASAP methods outperform the multiple-LPs method by an exponential margin. In the graph, while multiple-LPs could solve the problem only till 5 adversary types, the DOBSS and ASAP could solve till fourteen adversary types within 400s for DOBSS and 500s for ASAP.

Hence the conclusion that ASAP and DOBSS are faster than the multiple LPs. In comparing DOBSS vs ASAP, DOBSS is seen to be faster than ASAP albeit with a small speedup. More importantly, a key realization is that as we scale up the number of actions, ASAP is unable to obtain a feasible solution. Its approximations result in



(a) 1 Checkpoint (b) 2 Checkpoints

(c) 3 Checkpoints (d) Canines

Figure 5: DOBSS Strategy v/s Uniformly Random Strategy

ASAP incorrectly labeling problems to be infeasible, returning no answer. For example, in the same domain as tested in Figure 4, when numbers of leader actions were scaled up from 12 to 30 or higher, ASAP returns an infeasible answer in at least 30% of the cases. Consequently, we conclude that DOBSS is the algorithm of choice for Bayesian Stackelberg games [12].

7.2 Evaluation of ARMOR

We now evaluate the solution quality obtained when DOBSS is applied to the LAX security domain. We offer three types of evaluation. While our first evaluation is "in the lab," ARMOR is a deployed assistant, and hence our remaining two evaluations are of its deployment "in the field." With respect to our first evaluation, we conducted four experiments. The first three compared ARMOR's randomization with other randomization techniques, in particular a uniform randomization technique that does not use ARMOR's weights in randomization. The uniformly random strategy gives equal probabilities to all possible actions.

The results of the first experiment are shown in Figures 5(a), 5(b) and 5(c). The *x*-axis represents the probabilities of occurrence type 1 and type 2 adversaries. Recall we had mentioned the use of two adversary types to construct our Bayesian Stackelberg games for LAX in Section 4.3. The *x*-axis shows the probability p of adversary type 2 (the probability of adversary type 1 is then obtained on 1-p). The *y*-axis represents the reward obtained. Figure 5(a) shows the comparison when one checkpoint is placed. For example, when adversary of type 1 occurs with a probability of 0.1 and type 2 occurs with a probability of 0.9, the reward obtained by the DOBSS strategy is -1.72 whereas the reward obtained by a uniform random strategy is -2.112. It is important to note that the reward of the DOBSS strategy is strictly greater than the reward of the uniform random strategy for all probabilities of occurrence of the adversaries.

Figure 5(b) also has the probability distribution on the *x*-axis and the reward obtained on the *y*-axis. It shows the difference in the obtained reward when 2 checkpoints are placed. Here also the reward in the case of DOBSS strategy is greater than the reward of the uniform random strategy. When we have 2 checkpoints, the type 2 adversary chooses the action $none$ (to not attack). This leads to the observation that the rewards of the DOBSS strategy and the reward of the uniform strategy are the same when only the type 2 adversary is present. Figure 5(c) presents the case of 3 checkpoints. Here the reward values obtained by DOBSS in the 3 checkpoint case are always positive — this is because the chances of catching the adversary of type 1 improve significantly with 3 checkpoints. This also leads to the reward of DOBSS decreasing with the decrease in the probability of occurrence of the adversary of type 1. Note that the type 2 adversary as with the case of 2 checkpoints, decides $none$ and hence the reward of the DOBSS strategy and the uniformly random strategy are the same when only type 2 adversary is present.

Table 2: Variation in Usage Percentage

Checkpoint Number	1	2	3	4	5
Week 1	33.33	4.76	33.33	0	28.57
Week 2	19.04	23.80	23.80	14.28	19.05

The three experiments reported above allow us to conclude that DOBSS weighted randomization provides significant improvements over uniform randomization in the same domain, thus illustrating the utility of our algorithms. We continue these results in the following fourth experiment, focusing now on canine units. Figure 5(d) shows the comparison of the reward obtained between scheduling canine units with DOBSS and scheduling them with a uniform random strategy (denoted *URS*). In the uniform random strategy, canines are randomly assigned to terminals with equal probability. The *x*-axis represents the weekday and the *y*-axis represents the reward obtained. We can see that DOBSS performs better even with 3 canine units as compared to 6 canine units being scheduled using the uniform random strategy. For example, on Friday, the reward of uniform random strategy with 6 canine units is -1.47 whereas the reward of 3, 5 and 6 canines with DOBSS is 1.37, 3.50 and 4.50 respectively. These results show that DOBSS weighted randomization with even 3 canines provides better results against uniform randomization in the same domain with 6 canines. Thus our algorithm provides better rewards and can help in reducing the cost of resources needed.

Now we analyze the performance of ARMOR as it is deployed in the field. In the next evaluation, we examine ARMOR's setting of checkpoints at LAX. The first experiment examines the change in checkpoint deployment during a fixed shift (i.e. keeping the time fixed) over two weeks. The results are shown in Table 2. The numbers 1 to 5 in the table denote the checkpoint number (we have assigned arbitrary identification numbers to all checkpoints for the purpose of this experiment) and the values of the table show the percentage of times this checkpoint was used. For example, in week 1, checkpoint 2 was used just less than 5% of times, while checkpoint 2 was used about 25% of the times in week 2. We can make two observations from these two weeks: (i) we do not have uniform randomization of these checkpoints, i.e. there is great variance in the percentage of times checkpoints are deployed; (ii) the checkpoint deployment varies from week to week, e.g. checkpoint 4 was not used in week 1, but it was used 15% of the times in week 2.

The goal of the next experiment was to provide results on the sensitivity analysis, specifically, how the probabilities of different actions will change if we change the proportion of adversary types. Figure 6 shows the variation in strategy for placing two checkpoints together when the probability of occurrence of the adversary changes. The *x*-axis shows the variation in the probability of occurrence of the adversary types, whereas the *y*-axis shows the variation in the probabilities in the DOBSS's strategy. For example, when

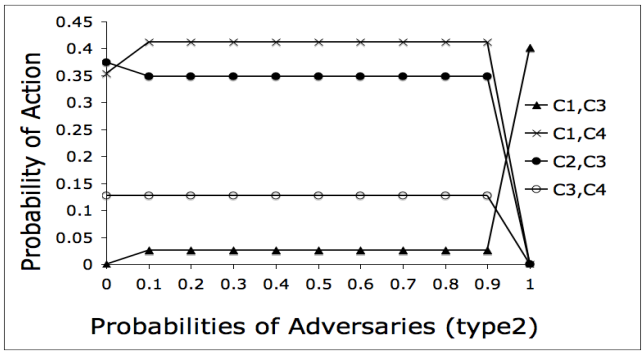


Figure 6: Sensitivy Analysis

adversary of type 1 occurs with a probability of 1, the probability of placing both checkpoints 1 and 4 is 0.353, when the adversaries 1 and 2 occur with probabilities 0.4 and 0.6 respectively, then the probability of placing checkpoints 3 and 4 is 0.127. We can observe that there is very little to no variation in the probabilities in the DOBSS strategies when the probabilities of occurrence of the two adversary types vary from .1 to .9. This indicates that our results are not particularly sensitive to variations in probabilities of opponents except at the very extremes.

Our final evaluation is a more informal evaluation based on feedback from the LAWA police. First, they have provided very positive feedback about the deployment. They suggest that the technique they had previously used was not one of randomization, but one of alternating checkpoints (e.g. if checkpoint 1 was active today, it would be inactive tomorrow); such a routine can bring about determinism in the scheduling which we have avoided. Second, ARMOR has eliminated the burden for creating schedules, thus reducing routine work and allowing LAWA police to focus on more important tasks. Third, several arrests have been made at checkpoints scheduled by ARMOR: typically these involved cars attempting to carry weapons into LAX. This does not necessarily suggest that ARMOR's schedule was responsible because this is not a controlled experiment per se. Nonetheless, it illustrates that the first line of defense at the outer airport perimeter is helping alleviate the threat of violence at the airport.

8. SUMMARY

Establishing security around airports, ports, or other infrastructure of economic or political importance is a challenge that is faced today by police forces around the world. While randomized monitoring (patrolling, checking, searching) is important — as adversaries can observe and exploit any predictability in launching an attack — randomization must use different weighing functions to reflect the complex costs and benefits of different police actions. This paper describes a *deployed agent assistant* called ARMOR that casts the monitoring problem as a Bayesian Stackelberg game, where randomized schedule generation for police forces can appropriately weigh the costs and benefits as well as uncertainty over adversary types. ARMOR combines three key features: (i) it uses the fastest known solver for Bayesian Stackelberg games called DOBSS, where the dominant mixed strategies provide schedule randomization; (ii) its mixed-initiative based interface allows users to occasionally adjust or override the automated schedule based on their local constraints; (iii) it alerts the users in case mixed-initiative overrides appear to degrade the overall desired randomization. ARMOR has been successfully deployed at the Los Angeles International Airport, randomizing allocation of checkpoints since August 2007 and canine deployment since November 2007. ARMOR thus represents a successful transition of multi-agent algorithmic advances that represent the culmination of research published in AAMAS [15, 13] for the past two years into the real-world.

9. REFERENCES

[1] General Description: Just the Facts. *http://www.lawa.org/lax/justTheFact.cfm*, 2007.

[2] N. Billante. The Beat Goes On: Policing for Crime Prevention. *http://www.cis.org.au/IssueAnalysis/ia38/ia38.htm*, 2003.

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

[4] V. Conitzer and T. Sandholm. Computing the Optimal Strategy to Commit to. In *EC*, 2006.

[5] B. Editors. Logan International Airport Deploys Advanced Perimeter Secuirty System; Vistascape and FLIR Systems Provide Software and Cameras for Pilot Program. *Business Wire*, May 13, 2003.

[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] R. C. Larson. A Hypercube Queuing Model for Facility Location and Redistricting in Urban Emergency Services. *Computer and OR*, 1(1):67–95, 1974.

[9] P. J. Lewis, M. R. Torrie, and P. M. Omilon. Applications Suitable for Unmanned and Autonomous Missions Utilizing the Tactical Amphibious Ground Support (TAGS) Platform. *http://www.autonomoussolutions.com/Press/SPIE%20TAGS.html*, 2005.

[10] P. Monier. Simcore: Airport Operations Simulation, Scheduling, Optimisation and Planning Software. *http://www.airport-technology.com/contractors /consult/simcore*, 2007.

[11] A. Murr. The Element of Surprise. *Newsweek National News*, http://www.msnbc.msn.com/id/21035785/site/newsweek/page/0/, 28 September 2007.

[12] P. Paruchuri. *Keep the Adversary Guessing: Agent Security by Policy Randomization*. PhD thesis, 2007.

[13] P. Paruchuri, J. P. Pearce, M. Tambe, F. Ordonez, and S. Kraus. An Efficient Heuristic Approach for Security Against Multiple Adversaries. In *AAMAS*, 2007.

[14] P. Paruchuri, M. Tabme, F. Ordonez, and S. Kraus. Safety in Multiagent Systems by Policy Randomization. In *SASEMAS*, 2005.

[15] P. Paruchuri, M. Tambe, F. Ordonez, and S. Kraus. Security in Multiagent Systems by Policy Randomization. In *AAMAS*, 2006.

[16] 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.

[17] D. Stevens and et. al. Implementing Security Improvement Options at Los Angeles International Airport. *http://www.rand.org/pubs/documented*$_b rie f ings/2006/RAND_D B499-1.pdf$, 2006.

[18] W. A. Wagenaar. Generation of Random Sequences by Human Subjects: A Critical Survey of Literature. 1972.