Using Game Theory for Los Angeles Airport Security

James Pita, Manish Jain, Fernando Ordóñez, Christopher Portway, Milind Tambe, Craig Western
University of Southern California, Los Angeles, CA 90089

Praveen Paruchuri
Intelligent Automation, Inc., RockVille, MD 20855

Sarit Kraus
Bar-llan University, Ramat-Gan 52900, Israel
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 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 two 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. 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.

Introduction

Protecting national infrastructure such as airports, historical landmarks, or a location 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. Such 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 ranging from security on university campuses to normal police beats to border or maritime security (Billante 2003; Paruchuri *et al.* 2007; Ruan *et al.* 2005).

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 (Fudenberg & Tirole 1991). 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 (Conitzer & Sandholm 2006) and solving it via the fastest optimal algorithm for such games (Paruchuri *et al.* 2008), addressing the first two challenges. The algorithm used builds on several years of research regarding multi-agent systems and security (Paruchuri *et al.* 2005; 2006; 2007). In particular,

ARMOR relies on an optimal algorithm called DOBSS (Decomposed Optimal Bayesian Stackelberg Solver) (Paruchuri *et al.* 2008).

While a Bayesian game allows us to address uncertainty over adversary types, by optimally solving such Bayesian Stackelberg games (which yield optimal randomized strategies as solutions), ARMOR provides quality guarantees on the schedules generated. These quality guarantees obviously do not imply that ARMOR provides perfect security; instead, ARMOR guarantees optimality in the utilization of fixed security resources (number of police or canine units) assuming the rewards are accurately modeled. In other words, given a specific number of security resources and areas to protect, ARMOR creates a schedule that randomizes over the possible deployment of those resources in a fashion that optimizes the expected reward obtained in protecting LAX.

The third challenge is addressed 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 may potentially deteriorate schedule quality.

ARMOR thus represents a very promising transition of multi-agent research into a deployed application. ARMOR has been successfully deployed since August 2007 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 where to randomly allocate canines to terminals. Indeed, February 2008 marked the successful end of the six month trial period of ARMOR deployment at LAX. The feedback from police at the end of this six month period was extremely positive; ARMOR will continue to be deployed at LAX and expand to other police activities at LAX.

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 and the largest destination airport in the United States, serving 60-70 million passengers per year (Airport 2007; Stevens *et al.* 2006). 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 (Stevens *et al.* 2006). To protect LAX, LAWA police have 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 to which randomization could be applied, 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 on terminals 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) LAX 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 — this fact 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.

In summarizing the domain requirements, we emphasize the following key points. First, it is LAWA police, as domain experts, who expressed a requirement for randomization, leading us to design ARMOR. Second, there exists different rings of security (including canines and checkpoints

that ARMOR schedules), which are not static and therefore may change independently of the other rings different times. The end result of such shifting randomized security rings is that adversary costs and uncertainty increase particularly for well planned attacks which in turn may help deter and prevent attacks.

Approach

We modeled the decisions of setting checkpoints or canine patrol routes at the LAX airport as Bayesian Stackelberg games. These games allow us to accomplish three important tasks: (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, how an LAX security problem can be mapped onto Bayesian Stackelberg games, some of the previous methods for solving Bayesian Stackelberg games, and how we use DOBSS to optimally solve the problem at hand.

Bayesian Stackelberg Games

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.* For example, given our security domain, the police force (leader) must first commit to a mixed strategy for placing checkpoints on roads in order to be unpredictable to the adversaries (followers), where a mixed strategy implies a probability distribution over the actions of setting checkpoints. The adversaries, after observing checkpoints over time, can then choose their own strategy of attacking a specific road. To see the advantage of being the leader in a Stackelberg game, consider a simple game with the payoff table as shown in Figure 2. The leader is the row player and the follower is the column player. Given a simultaneous move game, i.e. the leader and follower now act at the same time, 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. However, 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* in order to maximize its payoff, leading to a payoff for the leader of 3.5. Thus, by committing to a mixed strategy first, the leader is able to obtain a higher payoff than could be obtained in a simultaneous move situation.

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

Figure 2: Payoff table for example normal form game.

The Bayesian form of such a game then, implies that each agent must be of a given set of types. For our security domain, we have two agents, the police force and the adversary. While there is only one police force type, there are many different adversary types, such as serious terrorists, drug smugglers and petty criminals, denoted by L. During the game, the adversary knows its type, but the police do not know the adversary's type, this is an incomplete information game. For each agent (the police force and the adversary) i, there is a set of strategies σ_i and a utility function $u_i : L \times \sigma_1 \times \sigma_2 \rightarrow \Re$. Figure 3 shows a Bayesian Stackelberg game with two follower types. We also assume knowing *a-priori* probability p^l, where l represents the type of adversary (1, 2, etc.), of the different follower types (i.e. $l \in L$). Our goal is *to find the optimal mixed strategy* for the leader to commit to, given that the follower may know the leader's mixed strategy when choosing its strategy and that the leader will not know the follower's type in advance.

Follower Type 1

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

Follower Type 2

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

Figure 3: Security Agent vs Followers 1 and 2

Techniques for Solving Stackelberg Games

In previous work it has been shown that finding an optimal solution to a Bayesian Stackelberg game with multiple follower types is NP-hard (Conitzer & Sandholm 2006). Researchers in the past have identified an approach, which we will refer to as the Multiple-LPs method, to solve Stackelberg games (Conitzer & Sandholm 2006), and this can be used to solve Bayesian Stackelberg games. This approach, however, requires transforming a Bayesian game into a normal form game using the Harsanyi transformation (Harsanyi & Selten 1972). Similarly one may apply efficient algorithms for finding Nash equilibria (Sandholm, Gilpin, & Conitzer 2005), but they require the same Harsanyi transformation. Since our research crucially differs in its non-use of the Harsanyi transformation, it is important to understand this transformation and its impact.

Harsanyi Transformation The first step in solving Bayesian games for previous methods is to apply the Harsanyi transformation (Harsanyi & Selten 1972) that converts the incomplete information game into a normal-form game. Given that the Harsanyi transformation is a standard concept in game theory, we explain it briefly through a simple example without introducing the mathematical formulations. Consider the case of the two follower types 1 and 2 as shown in Figure 3. Follower type 1 will be active with probability α, and follower type 2 will be active with probability $1 - \alpha$. Performing the Harsanyi transformation involves introducing a chance node that determines the follower's type, thus transforming the leader's incomplete information regarding the follower into an imperfect information game. The transformed, normal-form game is shown in Figure 4. In the transformed game, the leader still

has two strategies while there is a single follower type with four (2*2) strategies. For example, consider the situation in the transformed game where the leader takes action *a* and the follower takes action *cc'*. The leader's payoff in the new game is calculated as a weighted sum of its payoffs from the two tables in Figure 3 i.e., α times payoff of leader when follower type 1 takes action c plus $1-\alpha$ times payoff of leader when follower type 2 takes action c'. All the other entries in the new table, both for the leader and the follower, are derived in a similar fashion. In general, for *n* follower types with *k* strategies per follower type, the transformation results in a game with k^n strategies for the follower, thus causing an exponential blowup losing compactness.

Methods such as (Conitzer & Sandholm 2006; Sandholm, Gilpin, & Conitzer 2005) must use this Harsanyi transformation, which implies the game loses its compact structure. Nonetheless, the solutions their methods obtain can be transformed back into the original game.

	cc'	cd'	dc'	dd'
a	2α+(1-α), 1	2, α	4α+(1-α), (1-α)	4α+2(1-α), 0
b	α, (1-α)	α+3(1-α), 2(1-α)	3α, 2α+(1-α)	3,2

Figure 4: Harsanyi Transformed Payoff Table

DOBSS

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 illustrate how it may be transformed into a linearized equivalent Mixed-Integer Linear Program (MILP). While a more detailed discussion of the MILP is available in (Paruchuri *et al.* 2008), the current section may at least serve to explain at a high level the key idea of the decomposition used in this 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. We denote by x the leader's policy (mixed strategy), 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:

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

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. Constraints with free indices mean they are repeated for all values of the index. For example, the fourth constraint means $x_i \in [0\dots 1]$ for all $i \in X$. The first and the fourth constraints define the set of feasible solutions x as a probability distribution over the set of actions X. The second and fifth constraints 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). 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.

The two inequalities in the third constraint ensure that $q^l_j = 1$ 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 $j \in Q$, $a^l \ge \sum_{i\in X} C^l_{ij} x_i$. 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 $q^l_j = 0$, since M is a large positive quantity. For the action that has $q^l_j = 1$ this inequality states that the adversary's payoff for this action must be $\ge a^l$, which combined with the previous inequality shows that this action must be optimal for follower type l. Notice that Problem 1 is a decomposed MIQP in the sense that it does not utilize a full-blown Harsanyi transformation; instead it solves multiple smaller problems using individual adversaries' payoffs (indexed by l). Furthermore, this decomposition does not cause any suboptimality (Paruchuri *et al.* 2008).

We can linearize the quadratic programming problem 1 through the change of variables $z^l_{ij} = x_i q^l_j$. The substitution of this one variable allows us to create an MILP. The details of this transformation and its equivalence to problem 1 are presented in (Paruchuri *et al.* 2008). DOBSS refers to this equivalent mixed-integer linear program, which can be solved with efficient integer programming packages. Although DOBSS still remains as an exponential solution to solving Bayesian Stackelberg games, by avoiding the Harsanyi transformation it obtains significant speedups over the previous approaches as shown in the experimental results and proofs in (Paruchuri *et al.* 2008)

Bayesian Stackelberg Game for the Los Angeles International Airport

We now illustrate how the security problems set forth by LAWA police 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. Given the checkpoint problem our game consists of two players: the LAWA police (the leader) and the adversary (the follower) in a situation consisting of a specific number of inbound roads on which to set up checkpoints, say roads 1 through k. LAWA police's set of pure strategies consist of a particular subset of those roads to place checkpoints on prior to adversaries selecting which roads to attack. LAWA police can choose a mixed strategy so that the adversary will be unsure of exactly where the checkpoints may be set up, but the adversary will know the mixed strategy LAWA police have chosen. 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 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 $X = \{1, \ldots, k\}$. If LAWA police were setting up a combination of two checkpoints, then $X = \{(1,2),(1,3)...(k-1,k)\}$, i.e. all combinations of two checkpoints. Each adversary type $l \in L = \{1, \ldots, m\}$ can decide to attack one of the k roads or maybe not attack at all (none), so its set of actions is $Q = \{1, \ldots, k, none\}$. If LAWA police select road i to place a checkpoint on and adversary type $l \in L$ selects road j to attack then the police receive a reward R^l_{ij} and the adversary receives a reward C^l_{ij}. 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^l_{ij} a large positive value and C^l_{ij} 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^l_{ij} a large negative value and C^l_{ij} 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^l_{ij} and C^l_{ij} 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 amateur attacker who may not have sufficient resources to carry out a large-scale attack. If these are the only two types of adversaries faced, then 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 amateur attacker. Our experimental data provides initial results about the sensitivity of our algorithms to the probability distributions over these two different adversary types. While the number of adversary types has varied based on inputs from LAWA police, for any one adversary type the largest game that has been 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 8 actions per adversary type (one for a possible attack on each terminal).

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 5, 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.

Interface

The ARMOR interface, seen in Figure 6, 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 key 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, the system constructs a new game.

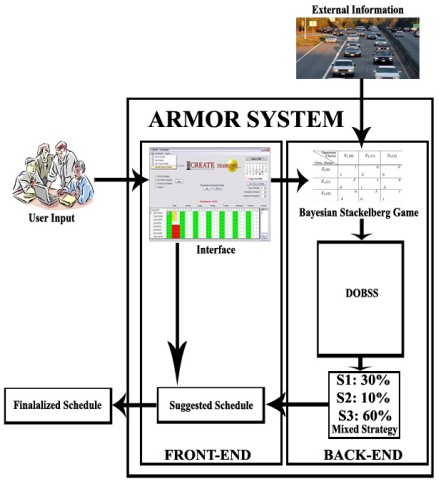


Figure 5: ARMOR System Flow Diagram

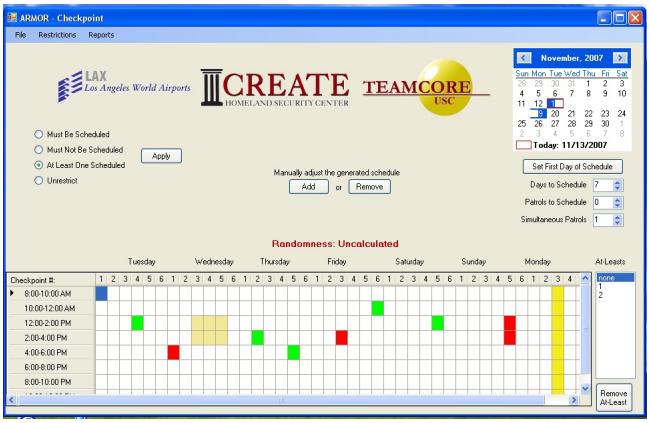


Figure 6: ARMOR Interface

As discussed previously, given knowledge of the total number of inbound roads, and 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 previously by the number of inbound roads. Thus, the system 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.

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.

The spreadsheet in the interface serves as the main mechanism 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 at a single time as seen in Figure 6. 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.

Matrix Generation and DOBSS

Given the submitted user information, the system must create a meaningful Bayesian Stackelberg game matrix. Previously we illustrated the generation of the action space in this game. Based on the pre-specified rewards as discussed earlier, 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 calculates the optimal mixed strategy over the current action space.

To demonstrate the process, assume there are three possible 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.

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, the system 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 previously, 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 constraining the schedule frequently, 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's imposed constraints and DOBSS's output strategy remains an issue for future work.

When a schedule is presented to the user with alerts, the user may alter the schedule by altering the forbidden/required checkpoints, or possibly by directly altering the schedule. Both possibilities are accommodated 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.

Design Challenges

Designing and deploying the ARMOR software on a trial basis at LAX posed numerous challenges and problems to our research group. Here 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 (Wagenaar 1972), humans have difficulty randomizing. Instead, mathematical randomization that appropriately weighs the costs and benefits of different actions, and randomizes accordingly leads to improved results. Security officials were hence extremely enthusiastic in their reception of our research and eager to apply it to their domain. In addition, these officials have indicated that obtaining schedules automatically reduces their burden of having to construct such schedules manually taking all the relevant factors into account.
- *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 selectively override the schedule provided.
- *Importance of providing police officers with operational flexibility:* When initially generating schedules for canine patrols, the system 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.

Experimental Results

Our experimental results explore the runtime efficiency of DOBSS and evaluate the solution quality and implementation of the ARMOR system.

Runtime Analysis

It has been shown in (Paruchuri *et al.* 2008) that DOBSS significantly outperforms its competitors, which include MIP-Nash (Sandholm, Gilpin, & Conitzer 2005) and Multiple-LPs (Conitzer & Sandholm 2006), in an experimental domain that involves a security agent patrolling a world consisting of m houses, $1 \ldots m$ and a robber trying to rob these houses. These results show that even for a world as small as 3 houses, MIP-Nash and Multiple-LPs are unable to converge on a solution within the allowed time of 30 minutes when there are 8 or more adversary types. DOBSS, however, is able to achieve a solution in less than 10 seconds for up to 14 adversary types. Also, as the number of houses increases up to 5, Paruchuri et. al. find that MIP-Nash and Multiple-LPs are unable to converge on a solution within 30 minutes for even low numbers of adversary types (Paruchuri *et al.* 2008).

Our runtime analysis adds to the above results, focusing specifically on the current security domain for which this work has been applied. For this reason we compare the runtime results of DOBSS versus Multiple LPs, described previously, given the specific domain used for canine deployment at LAX. MIP-Nash (Sandholm, Gilpin, & Conitzer 2005) has not been included in this analysis of runtimes as it only provides the best Bayes-Nash equilibrium as opposed to the optimal mixed strategies provided by the Multiple LPs method and the DOBSS method. 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 data from a full week of canine deployment to analyze the time necessary to generate a schedule given the DOBSS method and the Multiple LPs method. For completeness we show the results given one to four adversary types where four adversary types is the minimum amount LAWA has set forth as necessary.

In Figure 7 we summarize the runtime results for our Bayesian games using DOBSS and Multiple LPs. We tested our results on the Bayesian games provided from the canine domain with number of adversary types varying between one to four. Each game between LAWA and one adversary type is modeled as a normal form game. Thus, there are four normal form games designed for the game between LAWA

and the various adversary types for the base case. The size of each of these normal form games is (784,8) corresponding to 784 strategies for LAWA and 8 for the adversary. We then used the 7 generated instances, taken from an arbitrary week of canine deployment, of this base case to obtain averaged results.

The x-axis in Figure 7 shows the number of follower types the leader faces starting, and the y-axis of the graph shows the runtime in seconds. All the experiments that were not concluded in 20 minutes (1200 seconds) were cut off. From the graph we summarize that DOBSS outperforms the multiple-LPs method by a significant margin given our real canine domain. In the graph, while multiple-LPs could solve the problem only for up to 2 adversary types, DOBSS could solve for all four adversary types within 80s.

Hence we see that the DOBSS method is faster than the multiple-LPs method. Consequently, we conclude that DOBSS is the algorithm of choice for Bayesian Stackelberg games (Paruchuri *et al.* 2008), especially given the particular games created by real security domains such as the canine patrolling problem presented in this paper.

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 a uniform randomization technique that does not use ARMOR's weights in randomization.

The results of the first experiment are shown in Figures 8(a), 8(b) and 8(c). The *x*-axis represents the probabilities of occurrence of the two adversary types we chose to focus on. Since the actual number of adversary types used for LAX is secure information, we use 2 adversary types for simplicity in this analysis. 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 by LAWA. This reward represents the expected reward LAWA would obtain given the optimal adversary response to the strategy adopted by LAWA. Figure 8(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

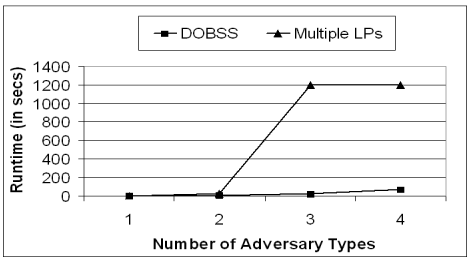
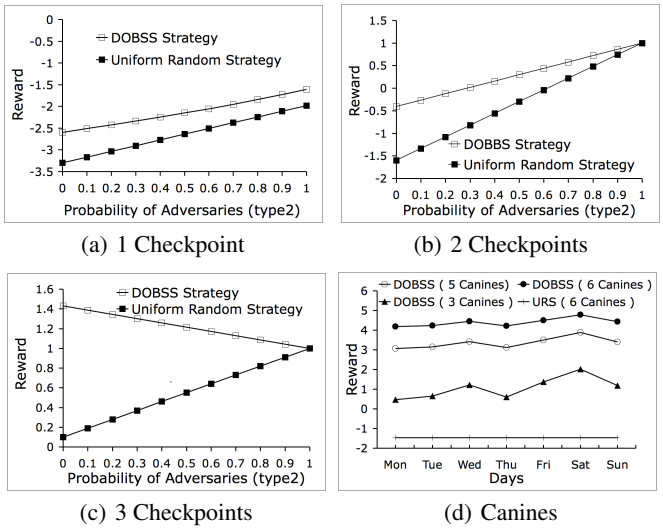


Figure 7: Runtimes: DOBSS and Multiple-LP methods



(a) 1 Checkpoint (b) 2 Checkpoints

(c) 3 Checkpoints (d) Canines

Figure 8: DOBSS Strategy v/s Uniformly Random Strategy

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 adversary types.

Figure 8(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 the 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 reward of the DOBSS strategy and the reward of the uniform strategy are the same when only the type 2 adversary is present. Figure 8(c) presents the case of 3 checkpoints. Here the reward values obtained by DOBSS 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.

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 8(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

Table 1: 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

6 canine units being scheduled using the uniform random strategy. For example, on Friday, the reward of a uniformly 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.

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 1. 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 appear to have uniform use 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.

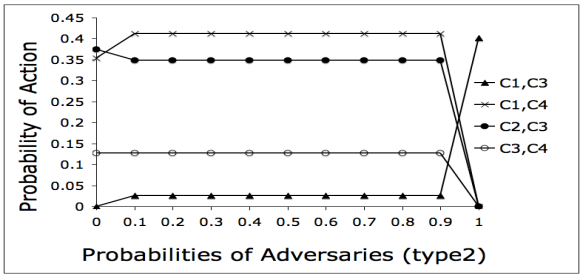


Figure 9: Sensitivity Analysis

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 9 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 strategy. For example, when adversary of type 1 occurs with a probability of 1, the probability of placing both checkpoints 1 and 4 is 0.353, when 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 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 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; such a routine can bring about determinism in the scheduling which we have avoided. Second, ARMOR has reduced routine work in scheduling, which allows 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. Finally, Director James Butts of LAX police has commented that the new random placement "makes travelers safer" and even gives them "a greater feeling of police presence" by making the police appear more numerous(Murr 2007). Also Chief Erroll Southers in a testimony to a congressional committee has commented "LAX is safer today than it was eighteen months ago" citing ARMOR as one of the key factors (US House 2008). 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.

Related Work and Summary

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 (Ruan *et al.* 2005; Billante 2003). 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. Two exceptions are Paruchuri et al and their early work (Paruchuri *et al.* 2006), which provides algorithms for analyzing randomization-reward trade offs, and Agmon et al and their recent work (Agmon, Kraus, & Kamink 2008), which provides algorithms for reducing the probability of penetration. However, unlike our work, neither model any adversaries or adversary types.

Finally, the sequence from (Koller & Pfeffer 1997) provides an alternative compact representation to normal form representation. However, representing commitment to a mixed strategy, as required in our Stackelberg games is difficult in this representation, making its use difficult. Furthermore, (Koller & Pfeffer 1997) have not focused on computing optimal response in Stackelberg games, but rather in only finding equilibria.

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 Queuing Model" (Larson 1974) 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, on the other hand, tailors policies to combat various potential adversaries.

Two different approaches have been presented previously to find solutions to Bayesian Stackelberg games efficiently. One of the approaches, named ASAP (Paruchuri *et al.* 2007), is able to operate on the Bayesian form of Stackelberg games, but it provides an approximate solution. The second approach, the Multiple-LPs method, requires a Bayesian game to be transformed into a normal-form game using the Harsanyi transformation (Harsanyi & Selten 1972). DOBSS is superior to ASAP in that it provides exact solutions and as shown it also outperforms the Multiple-LPs method for our domain of interest.

In summary, establishing security around infrastructure of economic or political importance is a challenge that is faced today by police forces around the world. While randomized monitoring is important — as adversaries can observe and exploit any predictability — 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 can appropriately weigh the costs and benefits as well as uncertainty over adversary types. ARMOR combines two 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. 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 (Paruchuri *et al.* 2006; 2007; 2008) for the past two years into the real-world.

Acknowledgements

ARMOR's deployment at LAX has only been possible due to the exceptional effort by LAWA police to strike a collaboration. This research was supported by the United States Department of Homeland Security through the Center for Risk and Economic Analysis of Terrorism Events (CREATE) under grant number 2007-ST-061-000001. However, any opinions, findings, and conclusions or recommendations in this document are those of the authors and do not necessarily reflect views of the United States Department of Homeland Security. We would also like to thank the National Science Foundation for their contributions under grant number IS0705587.

References

Agmon, N.; Kraus, S.; and Kamink, G. A. 2008. Multi-Robot Perimeter Patrol in Adversarial Settings. In *Proceedings of ICRA*.

Airport, L. A. I. 2007. General Description: Just the Facts. *http://www.lawa.org/lax/justTheFact.cfm*.

Billante, N. 2003. The Beat Goes On: Policing for Crime Prevention. *http://www.cis.org.au/issueanalysis/IA38/IA38.HTM*.

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

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

Harsanyi, J. C., and Selten, R. 1972. A Generalized Nash Solution for Two-Person Bargaining Games with Incomplete Information. *Management Science* 18(5):80–106.

Koller, D., and Pfeffer, A. 1997. Representations and Solutions for Game-Theoretic Problems. *Artificial Intelligence* 94(1-2):167–215.

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

Murr, A. 2007. The Element of Surprise. *Newsweek National News* http://teamcore.usc.edu/ARMOR/newsweek.pdf.

Paruchuri, P.; Tambe, M.; Ordóñez, F.; and Kraus, S. 2005. Safety in Multiagent Systems by Policy Randomization. In *SASEMAS*.

Paruchuri, P.; Tambe, M.; Ordóñez, F.; and Kraus, S. 2006. Security in Multiagent Systems by Policy Randomization. In *AAMAS*.

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

Paruchuri, P.; Pearce, J. P.; Marecki, J.; Tambe, M.; Ordóñez, F.; and Kraus, S. 2008. Playing Games for Security: An Efficient Exact Algorithm for Solving Bayesian Stackelberg Games. In *AAMAS*.

Ruan, S.; Meirina, C.; Yu, F.; Pattipati, K. R.; and Popp, R. L. 2005. Patrolling in a Stochastic Environment. In *10th Intl. Command and Control Research and Tech. Symp.*

Sandholm, T.; Gilpin, A.; and Conitzer, V. 2005. Mixed-Integer Programming Methods for Finding Nash Equilibria. In *AAAI*.

Stevens, D.; Hamilton, T.; Schaffer, M.; Dunham-Scott, D.; Medby, J. J.; Chan, E.; Gibson, J.; Eisman, M.; Mesic, R.; Kelley, C.; Kim, J.; LaTourrette, T.; and Riley, K. J. 2006. Implementing Security Improvement Options at Los Angeles International Airport. *http://www.rand.org/pubs/documented_briefings/2006/RAND_DB499-1.pdf*.

US House, Committee on Homeland Security, F. C. 2008. The Resilient Homeland - Broadening the Homeland Security Strategy.

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