2-2016

Get me to my GATE on time: Efficiently solving general-sum bayesian threat screening games

Aaron SCHLENKER

Matthew BROWN

Arunesh SINHA
Singapore Management University, aruneshs@smu.edu.sg

Milind TAMBE

Ruta MEHTA

Get Me to My GATE On Time: Efficiently Solving General-Sum Bayesian Threat Screening Games

Aaron Schlenker and **Matthew Brown** and **Arunesh Sinha** and **Milind Tambe**[1] and **Ruta Mehta**[2]

Abstract. Threat Screening Games (TSGs) are used in domains where there is a set of individuals or objects to screen with a limited amount of screening resources available to screen them. TSGs are broadly applicable to domains like airport passenger screening, stadium screening, cargo container screening, etc. Previous work on TSGs focused only on the Bayesian zero-sum case and provided the MGA algorithm to solve these games. In this paper, we solve Bayesian general-sum TSGs which we prove are NP-hard even when exploiting a compact marginal representation. We also present an algorithm based upon a adversary type hierarchical tree decomposition and an efficient branch-and-bound search to solve Bayesian general-sum TSGs. With this we provide four contributions: (1) GATE, the first algorithm for solving Bayesian general-sum TSGs, which uses hierarchical type trees and a novel branch-and-bound search, (2) the Branch-and-Guide approach which combines branch-and-bound search with the MGA algorithm for the first time, (3) heuristics based on properties of TSGs for accelerated computation of GATE, and (4) experimental results showing the scalability of GATE needed for real-world domains.

1 Introduction

Screening for threats represents a significant security challenge, whether it is preventing an attack at a sports complex, interdicting illicit cargo shipping, or enforcing border protection. Such domains require finding the best way to allocate limited screening resources so as to reduce the expected consequences of a possible attack. This challenge is especially pronounced in settings where both screening efficiency and screening effectiveness are central objectives. As an example, the Transportation Security Administration (TSA) in the United States has launched the Dynamic Aviation Risk Management Solution (DARMS) initiative where risk-based approaches are being proposed to improve vital aspects of aviation security such as passenger screening [1].

Threat Screening Games (TSGs) [4] have been introduced to model screening domains where the screener must process a set of people or objects coming into an secure area, while an adversary tries to sneak in an attack through screening. An example is a terrorist trying to pass through airport screening with plastic explosives in their carry-on luggage to attack a flight from New York to Los Angeles. Indeed, airport threat screening by itself is a vast worldwide problem, emphasizing the tremendous importance of research on the TSG model; particularly given that the TSA in the United States appears to be moving forward with this model [4]. However, even beyond airport screening the TSG model is applicable for screening of cargo in ports (again an important worldwide challenge), the screening of stadium patrons or protecting against illegal trafficking across borders. In TSGs, the screener uses different types of screening resources where each resource: (i) detects a possible attack method with different levels of effectiveness; and (ii) has a capacity limit in terms of the number of screenees that can be processed by that resource in a certain time period. The goal of the screener is to find the most effective way to allocate screenees to screening resources while being constrained by the capacity limits on the use of those resources. Further complicating the optimization is that screening resources can be combined into screening teams. The presence of screening teams introduce complex capacity constraints into the optimization problem as all teams containing the same resource must satisfy the original resource capacity constraints on that resource. This creates a difficult challenge as the most effective teams can only be used a limited number of times and selecting the passengers to apply the highest scrutiny to becomes an important problem.

The focus of this paper is on solving Bayesian general-sum TSGs. Previous research in TSGs [4] solves the game with the zero-sum assumption - a restrictive case that does not hold in many real world domains. By using the zero-sum assumption and a compact *marginal representation* the previous work[4] solves the game in polynomial time using an LP. Unfortunately, Bayesian general-sum games are NP-hard to solve even when exploiting a compact marginal representation for the screener's strategy. Hence, the problem we are solving is fundamentally more difficult than what is considered in previous research and in order for the TSG model to be extended to real world domains an approach to solve the general-sum case is necessary. Beyond [4], there has been work done on Bayesian general-sum security games [6, 8, 13] but as discussed in Section 2, TSGs differ from security games in crucial aspects. Thus, for large scale TSGs novel techniques and efficient solution algorithms need to be developed to apply the Bayesian general-sum TSG model to real world domains.

To solve Bayesian general-sum TSGs we present an algorithm that uses hierarchical adversary type trees to break a TSG down into smaller, restricted games containing a subset of the adversary types. These restricted games are solved using an efficient branch-and-bound search tree combined with MGA [4]; the solution information from these 'child' nodes are then passed up to the 'parent' nodes in the hierarchical tree where the information provides (i) infeasible strategy information for pruning, (ii) tighter bounds, and (iii) branching heuristics which provide faster computation at the parent nodes. We also provide heuristics based upon the properties of TSGs that increase the computational speed of the algorithm even further.

Thus, by improving upon techniques for solving Bayesian general-

[1] University of Southern California, United States, email: {aschlenk,mattheab,aruneshs,tambe}@usc.edu
[2] University of Illinois at Urbana-Champaign, United States, email: rutameh ta@cs.illinois.edu

sum Stackelberg games and exploiting a compact TSG representation, we provide the following contributions: (1) GATE, the first algorithm for solving Bayesian general-sum TSGs, which uses hierarchical type trees and a novel branch-and-bound search, (2) the Branch-and-Guide approach which combines branch-and-bound search with MGA for the first time, (3) heuristics based on the properties of TSGs for accelerated computation of GATE by reducing the adversary strategy space that needs to be explored in the hierarchical type tree, and (4) solution quality and experimental results showing the scalability of GATE for large scale TSG instances.

2 Related Work

The problem of threat screening has been explored extensively in literature. However, it has been pointed out [4] that the TSG approach (which is inspired by security games) is much better than prior non-game-theoretic models in domains such as, screening for shipping containers [2], stadium patrons [15], and airport passengers [11, 12] or simple game-based models such as [19]. Specifically, previous non-game-theoretic approaches fails to model a rational adversary aiming to take advantage of vulnerabilities in the screening strategies whereas TSGs take this into account when devising screening strategies.

Furthermore, previous solution methods for solving security games fail to apply directly to our domain. This happens because TSGs (i) include a group of non-player screenees that all must be screened while a single adversary tries to pass through screening undetected, (ii) model screening resources with different efficacies and capacities that can be combined to work in teams and (iii) do not have an explicitly modeled set of targets. These are fundamental differences from traditional security games [9, 14, 16, 18] where the defender protects a set of targets against a single adversary. Previous research in TSGs provides an efficient solution method for the zero-sum case where our techniques are not helpful, but as mentioned in the introduction these techniques are not applicable to Bayesian general-sum TSGs.

In terms of solving Bayesian Stackelberg games there are three main approaches in previous literature: the Multiple LPs approach [5], the DOBSS algorithm [13], and the HSBA algorithm [8]. Multiple LPs fails to scale up for Bayesian Stackelberg games as it requires solving an exponential number of LPs [5]. DOBSS is an algorithm that exactly solves Bayesian Stackelberg games. However, it uses the normal form representation of the game which in our case is infeasible as our pure strategy space is exponentially larger than the space they were considering. HSBA represents the closest alternative solution method for our problem. HSBA breaks down the Bayesian game into smaller restricted games and solves the restricted games using an efficient branch and bound search with column generation. The solution information from these restricted games is then used to solve the original game more effectively. Unfortunately, [4] shows that column generation is not a scalable approach even for solving large scale Bayesian zero-sum TSGs. Still, in this paper we make use of some techniques in HSBA, namely breaking the game down into smaller restricted games, which we describe in more depth in Section 5 along with significant innovations for solving general-sum TSGs in Sections 5 and 6.

3 Threat Screening Game Model

A *threat screening game* (TSG) is a Stackelberg game played between the screener (leader) and an adversary (follower) in the presence of a set of non-player screenees that pass through a screening checkpoint operated by the screener. While TSGs are applicable to many domains, given that readers may be familiar with passenger screening at airports, we use examples from that domain. A TSG is composed of the following:

- *Time windows*: These represent the temporal dimension of screening as screenees arrive over time. This is modeled by slicing the game into a set of time windows W. The superscript w is used to indicate a specific time window.
- *Screenee categories*: Screenees have implicit characteristics, e.g., risk level and flight, which can be used to aggregate them into screenee categories $c \in C$ as they are functionally indistinguishable within the context of the game. The total number of screenees in each category c is N_c and the number of screenees in c that arrive at the screening checkpoint during time window w is N_c^w. A constant arrival rate is assumed for screenees within each screenee category and time window. All screenees in a category arriving in the same time window are screened equally in expectation according to the randomized screening strategy.
- *Adversary actions:* The adversary chooses a time window $w \in W$ to go through screening, a screenee category $c \in C$ to pose as during screening, and an attack method $m \in M$. An example adversary action is w = *8:00AM-9:00AM*, c = {*HighRisk, Flight1*}, and m = *on-body explosives*.
- *Adversary types*: The adversary has implicit characteristics that cannot be chosen, e.g., TSA-assigned risk level. We consider *adversary types* $\theta \in \Theta$, which restrict the adversary to select from screenee categories $C_\theta \subset C$. The adversary knows their own type, but the screener only knows a prior distribution $\mathbf{z}$ over the adversary types.
- *Resource types*: The set of screening resource types is R and all resources of type $r \in R$, e.g., all walk-through metal detectors, can be used to screen a combined total of at most L_r^w screenees during time window w.
- *Team types*: Screenees are screened by one or more resources types, e.g., walk through metal detector *and* x-ray machine. Each unique combination of resources types constitutes a screening team type t. The set of all valid team types, denoted by T, is given a priori.
- *Team type effectiveness*: Team types vary in their ability to detect different attack methods. For team type t, E_m^t is the probability of detection against attack method m.

Pure strategy A pure strategy P for the screener can be represented by $|W| \times |C| \times |T|$ non-negative *integer-valued* numbers $P_{c,t}^w$, where each $P_{c,t}^w$ is the number of screenees in c assigned to be screened by team type t during time window w. Pure strategy P must assign every screenee to a team type while satisfying the resource type capacity constraints for each time window, via the following constraints:

$$\sum_{t \in T} I_r^t \sum_{c \in C} P_{c,t}^w \leq L_r^w \quad \forall w, \forall r \tag{1}$$

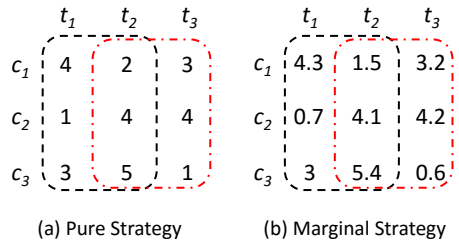
$$\sum_{t \in T} P_{c,t}^w = N_c^w \quad \forall w, \forall c \tag{2}$$

where I_r^t is an indicator function returning 1 if team type t contains resource type r and 0 otherwise. We denote the set of all valid pure strategies as $\hat{P}$ and we assume $\hat{P} \neq \emptyset$, i.e., it is possible to assign every screenee to a team type.

For example, consider a game with one time window w_1 and two screening resources r_1, r_2 with capacity constraints $L_{r_1,r_2} = 20$, respectively. The resources are combined into three screening teams $t_1 = \{r_1\}$, $t_2 = \{r_1, r_2\}$ and $t_3 = \{r_3\}$. There are three categories

c_1, c_2 and c_3 and each has $N_c = 9$ passengers. Figure 1(a) shows an example of a pure strategy allocation in this game.

The pure strategies for the adversary types are denoted as $a^{\theta,w}_{c,m}$ which specifies that adversary type θ selects time window w, screenee category c, and attack method m.



(a) Pure Strategy (b) Marginal Strategy

Figure 1. TSG Strategies

In the security games literature, two approaches are commonly used to handle scale-up: marginal strategies [9, 10] and column generation [6, 20]. To solve large-scale zero-sum TSGs [4] employed a marginal-based approach, showing that such an approach significantly outperformed the use of column generation. Hence we continue with the use of marginal strategies.

Marginal strategy A marginal strategy $\mathbf{n}$ for the screener can be represented by $|W| \times |C| \times |T|$ non-negative *real-valued* numbers $n^w_{c,t}$, where $n^w_{c,t}$ is the number of screenees in c assigned to be screened by team type t during time window w. We would want this marginal strategy to be a valid mixed strategy (*implementable*), i.e., there should exist a probability distribution over $\hat{P}$ given by q_P (i.e., $\sum_{P\in\hat{P}} q_P = 1$, $0 \le q_P \le 1$) such that $n^w_{c,t} = \sum_P q_P P^w_{c,t}$. An example marginal screener strategy is shown in Figure 1(b) with the same game parameters described previously.

Utilities Since all screenees in category c are screened equally in expectation, we can interpret $n^w_{c,t}/N^w_c$ as the probability that a screenee in category c arriving during time window w will be screened by team type t. Then, the probability of detecting an adversary type in category c during time window w using attack method m is given by $x^w_{c,m} = \sum_t E^t_m n^w_{c,t}/N^w_c$. The payoffs for the screener are given in terms of whether adversary type θ chooses screenee category c and is either detected during screening, denoted as $U^d_{s,c}$, or is undetected during screening, denoted as $U^u_{s,c}$. Similarly, the payoffs for adversary type θ are given in terms of whether θ chooses screenee category c and is either detected during screening, denoted as $U^d_{\theta,c}$, or is undetected during screening, denoted as $U^u_{\theta,c}$. Given adversary type θ pure strategy $a^{\theta,w}_{c,m}$, the screener's expected utility is given by $U_s(a^{\theta,w}_{c,m}) = x^w_{c,m}U^d_{s,c} + (1 - x^w_{c,m})U^u_{s,c}$ and the expected utility for adversary type θ is given by $U_\theta(a^{\theta,w}_{c,m}) = x^w_{c,m}U^d_{\theta,c} + (1-x^w_{c,m})U^u_{\theta,c}$.

4 Approach

While it has been shown that Bayesian Stackelberg games are hard to solve [5], it is interesting to observe that Bayesian general-sum TSGs are hard to solve even in the marginal strategy space as we prove next. This result shows that finding the optimal marginal strategy n for Bayesian general-sum TSGs is fundamentally more complex than the zero-sum case which can be solved in polynomial time as an LP. Thus, it is not surprising that the solution approaches used in [4] are not directly applicable to the general-sum case.

Theorem 1. *Finding the optimal solution in Bayesian general-sum TSGs is NP-hard even in the relaxed marginal strategy space.*

Proof. We reduce from the knapsack problem to our problem. Assume n items with weights w_i and value v_i with a sack of capacity K. Wlog, assume w_i and K are integers. Construct a game with n adversary types $|\Theta| = n$. Each type of adversary has two flights to board: f_0, f_1. Thus, $C = \{(\theta_i, f_j) \mid 0 \le i \le n, j \in \{0, 1\}\}$. Choose the other parameters of the game as follows: two resources r_1,r_2 with capacities $L_{r_1} = K$ and $L_{r_2} = \infty$. We have two teams $t_1 = \{r_1\}$ and $t_2 = \{r_2\}$. There is only one attack method m_1. For this game $E^{t_1}_{m_1} = 1$ and $E^{t_2}_{m_1} = 0$, i.e., t_2 is not effective at all at detecting m_1. The number of screenees for each screenee category $N_{(\theta_i,f_0)} = w_i$ and $N_{(\theta_i,f_1)} = 1$ for all $\theta_i \in \Theta$. Each type θ_i occurs with probability $\frac{v_i}{\sum_{\theta_i} v_i}$.

The utilities for the screener are $U^d_{s,(\theta_i,f_0)} = 0, U^u_{s,(\theta_i,f_0)} = 0$, $\forall\ \theta_i$ and $U^d_{s,(\theta_i,f_1)} = 2, U^u_{s,(\theta_i,f_1)} = 1, \forall\ \theta_i$. Thus, the screener strictly prefers the adversary of every type to choose f_1. The utilities for the adversary are set as follows: $U^d_{\Theta,(\theta_i,f_0)} = 1, U^u_{\Theta,(\theta_i,f_0)} = 2$, $\forall\ \theta_i$ and $U^d_{\Theta,(\theta_i,f_1)} = 0, U^u_{\Theta,(\theta_i,f_1)} = 1, \forall\ \theta_i$. Thus, the adversary will choose (θ_i, f_1) only when the probability of detection $x_{(\theta_i,f_0)} = 1, x_{(\theta_i,f_1)} = 0$ (breaking ties in favor of the screener). This happens only when all of the screenees $N_{(\theta_i,f_0)} = w_i$ are screened by t_1. Thus, we have from the capacity constraints that $\sum_{\theta_i}$ chooses $_{f_1}\ w_i \le K$. Therefore, for the choice of f_1 by adversary of type θ_i, the screener earns $\frac{v_i}{\sum_{\theta_i} v_i}$ and otherwise the screener earns 0. Given, the optimization problem maximizes this utility: $\sum_{\theta_i}$ chooses $_{f_1}\ \frac{v_i}{\sum_{\theta_i} v_i}$, the optimization provides a solution for the knapsack problem.

The resulting TSG instance from the knapsack problem has two flights, two resources, two teams, and as many adversary types as the number of items n in the knapsack problem. The screenee types assigned to t_1 gives the knapsack solution. Therefore, the reduction from the knapsack problem is overall polynomial in the number of items n. □

The optimal marginal strategy for a Bayesian general-sum TSG can be obtained by solving the mixed integer linear program *MarginalStrategyMILP*, provided below.

$$\max_{\mathbf{n},\mathbf{s},\mathbf{x},\mathbf{a}} \quad \sum_{\theta\in\Theta} z_\theta s_\theta \tag{3}$$

$$s_\theta - U_s(a^{\theta,w}_{c,m}) \le (1 - a^{\theta,w}_{c,m}) \cdot Z \qquad \forall\theta, w, c, m \tag{4}$$

$$0 \le k_\theta - U_\theta(a^{\theta,w}_{c,m}) \le (1 - a^{\theta,w}_{c,m}) \cdot Z \qquad \forall\theta, w, c, m \tag{5}$$

$$x^w_{c,m} = \sum_{t\in T} E^t_m \frac{n^w_{c,t}}{N^w_c} \qquad \forall w, c, m \tag{6}$$

$$\sum_{t\in T} I^t_r \sum_{c\in C} n^w_{c,t} \le L^w_r \qquad \forall w, r \tag{7}$$

$$\sum_{t\in T} n^w_{c,t} = N^w_c \qquad \forall w, c \tag{8}$$

$$n^w_{c,t} \ge 0 \qquad \forall w, c, t \tag{9}$$

$$a^{\theta,w}_{c,m} \in \{0, 1\} \qquad \forall\theta, w, c, m \tag{10}$$

$$\sum_{w,c,m} a^{\theta,w}_{c,m} = 1 \qquad \forall\theta \tag{11}$$

Equations 10 and 11 force each adversary type θ to choose a pure strategy. Equation 3 is the objective function which maximizes the screener expected utility as a weighted summation of screener expected utility against adversary type θ, s_θ, multiplied by the probability of encountering adversary type θ, z_θ. Equation 4 defines the

screener's expected payoff against each adversary type, contingent on the choice of pure strategies by the adversary types. The constraint places an upper bound of $U_s(a_{c,m}^{\theta,w})$ on s_θ, but only if $a_{c,m}^{\theta,w} = 1$, as Z denotes an arbitarily large constant. For all other pure strategies of adversary type θ, the RHS is arbitrarily large. Similarly, Equation 5 places an upper bound of $U_\theta(a_{c,m}^{\theta,w})$ on the utility of adversary type θ, k_θ, for $a_{c,m}^{\theta,w} = 1$. Additionally, Equation 5 also lower bounds k_θ by the largest $U_\theta(a_{c,m}^{\theta,w})$ over all $a_{c,m}^{\theta,w}$. Taken together these upper and lower bounds ensure that the pure strategy selected by adversary type θ is a best response to the screener marginal strategy $\mathbf{n}$. Equations 7-9 requires that $\mathbf{n}$ satisfies the resource type capacity constraints (i.e., Equation 1) and screenee category assignment constraints (i.e., Equation 2).

Even though *MarginalStrategyMILP* represents a NP-hard problem, solving it does not necessarily produce an implementable marginal screener strategy, i.e., the marginal strategy may not map to a probability distribution over pure strategies. The issue of implementability was addressed in [4] for TSGs by introducing the Marginal Guided Algorithm (MGA), which uses the (potentially non-implementable) marginal strategy to additionally restrict the constraints of a TSG to obtain a provably implementable marginal strategy though it can possibly lose solution quality in the process.

5 GATE: Solving Bayesian General-Sum TSG

Despite operating in the marginal screener strategy space, solving the *MarginalStrategyMILP* is computationally expensive due to the presence of the integer variables that encode the adversary strategy space. Therefore, we instead seek to solve Bayesian general-sum TSGs using an algorithmic approach for exploring the adversary strategy space. An example of the adversary strategy space for two adversary types and two actions per type is shown in Figure 2(a). The leaf nodes in this tree represent all possible joint pure strategy combinations for the two adversary types.

As mentioned previously a standard algorithmic approach that could be used to solve Bayesian general-sum TSGs is Multiple LPs [5]. This technique exploits the fact that, by fixing the joint adversary pure strategy, the underlying optimization problem is converted from a MILP to an LP. Thus, an LP can be solved for each leaf node in the adversary strategy tree to obtain the best marginal screener strategy which induces that joint adversary pure strategy as a best response. The marginal strategy with the highest screener utility is returned as the solution for the TSG. However, the number of joint adversary pure strategies is exponential in the number of adversary types Θ and thus Multiple LPs is not scalable for large problem instances. Therefore, we propose the General-sum Algorithm for Threat screening game Equilibria (GATE) and in this section we provide an intuitive, high level description. In Section 6 we provide a detailed algorithm as our experimental results show GATE does not scale leading to the need for heuristics to further speed up computation.

5.1 Hierarchical Type Trees

At a high level GATE seeks to exploit the structure of TSGs and reduce the number of joint adversary pure strategies that need to be evaluated. GATE achieves this reduction by building off intuition from the HBSA algorithm [8], which involves constructing a hierarchical type tree. Such a tree decomposes the game with each node in the tree corresponding to a restricted game over a subset of adversary types. The idea is to solve these smaller, restricted games to efficiently obtain (1) infeasibility information to eliminate large sets of joint adversary pure strategies, and (2) utility upper bound information that can be used to terminate the evaluation of joint adversary pure strategies.

GATE operates on restricted TSGs, where we define $TSG(\Theta^{'})$ to be a TSG with a subset of adversary types $\Theta^{'} \subset \Theta$. It is important to note that, despite not including all adversary types, $TSG(\Theta^{'})$ does not ignore the screenee categories associated with adversary types $\Theta \backslash \Theta^{'}$. Indeed, $TSG(\Theta^{'})$ must still satisfy the constraint that all screenees in each category must be assigned to a screening team type, i.e., Equation 8. By continuing to enforce these constraints, the upper bounds generated will be tighter as the screener cannot focus all the screening resources on just a subset of screenee categories, helping to improve the ability of GATE to prune out joint adversary pure strategies.

The subsets of adversary types are decomposed such that each level in the hierarchical type tree forms a partition over Θ satisfying $\Theta_i^{'} \cap \Theta_j^{'} = \emptyset, \forall i, \forall j, i \neq j$ as well as $\cup_i \Theta_i^{'} = \Theta$. Additionally, the set of adversary types in each parent node is the union of the sets of adversary types of all of its children, with the root node of the hierarchical type tree corresponding to the full problem. Figure 2(b) shows an example of a hierarchical tree structure with full binary partitioning for a game with four adversary types. The root node is the parent to two restricted games with two adversary types, each of which is a parent to two restricted games for individual adversary types.

The evaluation of the hierarchical tree starts at the leaf nodes and works up the tree such that all child nodes are evaluated before the parent nodes are evaluated. Every node is processed by evaluating the pure strategies of the restricted game and propagating up only the feasible pure strategies (i.e., pure strategies inducible as an adversary best response). [8] proved that if a pure strategy $a_{\theta'}$ can never be a best response for adversary type θ' in a restricted game $TSG(\Theta')$ with $\Theta' = \{\theta'\}$ then any joint pure strategy containing $a_{\theta'}$ can never be a best response in any $TSG(\Theta'')$ with $\Theta' \subset \Theta''$. Thus, at a given node, it is only necessary to consider joint pure strategies in the cross product of the sets of feasible pure strategies passed up from the child nodes.

For each pure strategy to be propagated, the corresponding utility with respect to the restricted game is also passed up. [8] also proved that it is possible to upper bound the screener utility for joint adversary pure strategy a_Θ in $TSG(\Theta)$ by $\sum_{\theta \in \Theta} z_\theta \beta(a_\theta)$ where z_θ is the normalized probability of adversary type θ in $TSG(\Theta')$ and $\beta(a_\theta)$ is the upper bound on the screener utility for adversary pure strategy a_θ in the restricted game $TSG(\{\theta\})$. These upper bounds can be used to determine the order to evaluate joint pure strategies as well as when it no longer necessary to evaluate joint pure strategies. This propagation of pure strategies and upper bounds continues until the root node is solved to obtain the best solution for the game.

Example Consider a game with four adversary types $\Theta = \{\theta_1, \theta_2, \theta_3, \theta_4\}$, like in Figure 2(b), where each adversary type has two actions available, $a_{\theta_i}^1, a_{\theta_i}^2$. The full game is broken down into the restricted games and we solve the leaf nodes in the hierarchical tree first. Suppose that after we solve all of the leaf nodes we get the following action sets for the adversaries: $\theta_1 = \{a_{\theta_1}^1\}, \theta_2 = \{a_{\theta_2}^1, a_{\theta_2}^1\}$, $\theta_3 = \{a_{\theta_3}^1\}, \theta_4 = \{a_{\theta_4}^1\}$ (as the other actions are found to be infeasible). Then, in the node $\{\theta_1, \theta_2\}$ we evaluate $[a_{\theta_1}^1, a_{\theta_2}^2]$ and $[a_{\theta_1}^1, a_{\theta_2}^1]$ while for node $\{\theta_3, \theta_4\}$ we evaluate $[a_{\theta_3}^2, a_{\theta_4}^1]$. Now, at the root node (Θ) we only have to evaluate two joint adversary pure strategies (i.e., $[a_{\theta_1}^1, a_{\theta_2}^1, a_{\theta_3}^2, a_{\theta_4}^1]$ and $[a_{\theta_1}^1, a_{\theta_2}^2, a_{\theta_3}^2, a_{\theta_4}^1]$) instead of 16 (cross product of all adversary type strategy sets) in order to find the optimal strategy for the game.

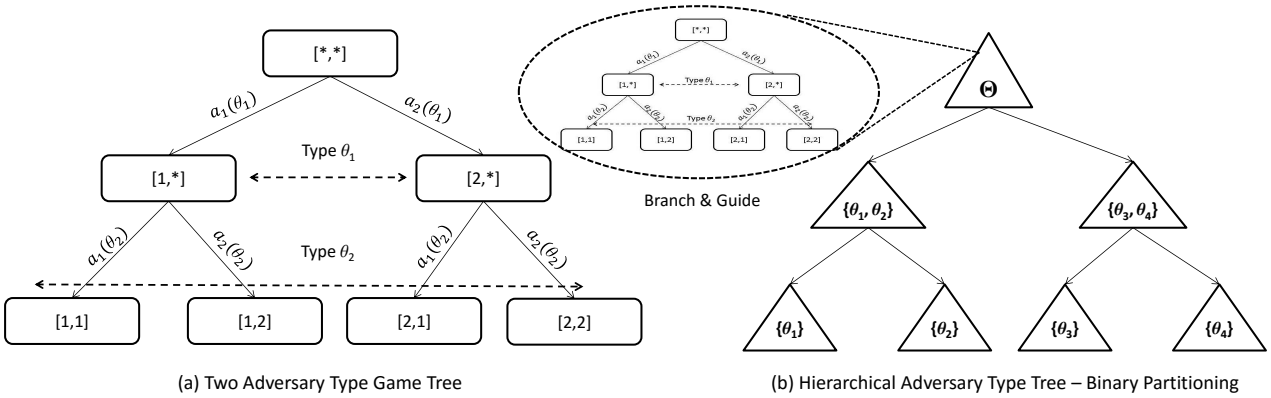


Figure 2. Bayesian Adversary Strategy Space

5.2 Advantages of GATE

As mentioned earlier, security games techniques do not apply to TSGs directly, and thus, HBSA is not well suited for our problem. In particular, HBSA utilizes a Branch-and-Price framework [3] which requires running column generation (given the large number of defender strategies in complex domains) to evaluate every single adversary joint pure strategy in the hierarchical type tree. While Branch-and-Price is a general approach frequently used for Bayesian Stackelberg games, column generation has been shown to be incapable of scaling for large-scale TSGs even in the zero-sum case due to the massive number of screener pure strategies [4]. Thus, having to repeatedly run column generation for the Bayesian general-sum TSGs is a non-starter.

To efficiently evaluate the joint adversary pure strategies in the nodes of the hierarchical tree, we introduce Branch-and-Guide, which combines branch-and-bound search with MGA to simultaneously mitigate the challenges of both scalability and implementability when solving Bayesian general-sum TSGs. Branch-and-Guide may be run at the root node of the hierarchical type tree, so that given a large of joint adversary pure strategies, we may be able to prune out a large number of them using upper-bounds, speeding up our computation. Furthermore, Branch-and-Guide exploits the fact that for a fixed joint adversary pure strategy, an implementable marginal screener strategy can be obtained quickly (even if it not necessarily optimal) and thus can avoid having to rely on column generation.

An example of the adversary strategy tree explored in Branch-and-Guide is shown in Figure 3, with the size and ordering of the tree based on the feasible joint adversary pure strategies and corresponding upper bounds propagated up by the child nodes. Branches to the left fix the joint adversary pure strategy, converting *MarginalStrategyMILP* into a linear program which can be solved efficiently. However, the resulting marginal strategy may not be implementable and thus we run MGA on the marginal while ensuring that the selected joint adversary pure strategy is still a best response. This two-step process produces an implementable marginal strategy that gives a lower bound on the overall solution quality. Branches to the right represent the upper bound on the screener utility for the next best joint adversary pure strategy which is calculated using the solution quality information passed up from the child nodes. If the screener utility for the best solution found thus far is higher than the upper bound than the next best joint adversary pure strategy, then the execution of Branch-and-Guide can be terminated without exploring the remaining joint adversary pure strategies.

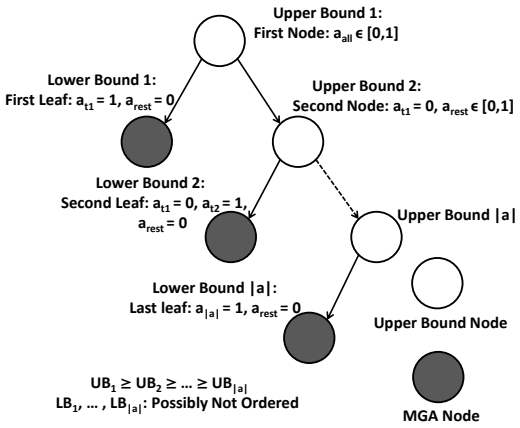


Figure 3. Branch-and-Guide Tree

6 Scaling Up GATE

While GATE incorporates state-of-the-art techniques to solve *MarginalStrategyMILP*, it fails to scale up to real world problem sizes for TSGs (see comparison in Evaluation). Thus, we employ intuitive heuristics that further narrows down the search space for GATE, thereby enabling up to 10X run time improvement with only 5-10% solution quality loss. There are two distinct steps in GATE where additional heuristics can help: the processing step at the leafs of the hierarchical type tree and the processing step at the intermediate and root nodes. In this section we first present GATE-H (GATE with heuristics) formally and then describe the heuristics used to speed up the computation.

6.1 GATE-H: GATE with Heuristics

GATE-H solves TSGs efficiently by limiting both the number of adversary pure strategies passed up the hierarchical adversary type tree from restricted games and by limiting the number of adversary strategies evaluated in the individual nodes. Each node in the hierarchical tree is solved using Algorithm 1, beginning at the leaf nodes. The feasible adversary pure strategy set, denoted as A', is passed up to the parent nodes as each child is solved. Notice that not all strategies need be evaluated at a given node for the computation to terminate

Algorithm 1: $\mathsf{GATE-H-NODE}(\Theta, \mathsf{A}^{\mathsf{i}}_{\Theta}, \mathsf{B}^{\mathsf{i}}, \mathsf{U_s}, \mathsf{U}_{\Theta}, \mathsf{K})$

//A^i_Θ: Pruned feasible pure strategy set for all adversary types
1. $A^{''}$:= all-Joint-Adversary-Pure-Strategies()
2. $B^{'}(a_\Theta)$:= getBound(a_Θ, B^i) $\forall a_\Theta \in \prod_\Theta A^i_\Theta$
3. sort($A^{''}, B^{'}(a_\Theta)$) //sort a_Θ in descending order of $B^{'}(a_\Theta)$
4. $a_\Theta := [A^1_\Theta(1), A^2_\Theta(1), ..., A^{|\Theta|}_\Theta(1)]$
5. $r^*, r^{'} = -\infty$ //Save best and iterative solutions
repeat
  6. (*feasible*, n, r) := MGA(a_Θ)
  if *feasible* **then**
    7. $A^{'} := A^{'} \cup a_\Theta$
    if $r > r^*$ **then**
      8a. $r^* := r$
      8b. $n^* := n$
    9. $B^{'}(a_\Theta) := r$
  else
    10. $A^{''} := A^{''} \backslash a_\Theta$
  11. **Every** K iterations
  if $r^{'} = r^* \neq -\infty$ **then**
    12. break
  else
    13. $r^{'} := r^*$
  14. a_Θ := getNextStrategy($a_\Theta, r^*, A^i_\Theta, B^i$)
until a_Θ = NULL
return $(n^*, r^*, A^{'}, B^{'})$

as either the Branch-and-Guide heuristic or K cutoff heuristic, both introduced later in this section, can end the computation early.

When solving a given node in GATE-H we take in the adversary set (Θ) and the screener's and adversary's utilities (U_s and U_Θ, respectively), with the feasible strategies (A^i_Θ) and bound information (B^i) is acquired from the children of that node as shown in Algorithm 1. In the case of solving a leaf node in the binary tree we enumerate all of that adversary type's strategies and get bound information by solving an upper bound LP, described in Section 6.2. After constructing the joint adversary pure strategy set (Line 1), we order the set by their upper bound values (Line 3) and evaluate each strategy one by one (main loop starts after Line 5). Heuristics are used inside of this loop, leading to GATE-H.

Algorithm 2: $\mathsf{getNextStrategy}(\mathsf{a}_\Theta, \mathsf{r}^*, \mathsf{A}^{\mathsf{i}}_\Theta, \mathsf{B}^{\mathsf{i}})$

for $i = |\Theta|$ to 1 **Step**-1 **do**
  j := index-of(a_Θ,A^i_Θ)
  //Set each adversary type strategy equal to left most leaf
  $a^i_\Theta := [A^1_\Theta(1), ..., A^{|\Theta|-1}_\Theta(1), A^{|\Theta|}_\Theta(j+1)]$
  if $r^* <$ getBound(a_Θ, B) **then**
    return a_Θ
return NULL

The first of the two heuristics used is the Branch-and-Guide approach (Line 14 - getNextStrategy()) which ends the computation early if the value of the current best strategy is greater than the next highest upper bound. The second heuristic, discussed in more detail in Section 6.3, is the K cutoff (Line 11) which ends the computation early if the current best solution is not improving after K iterations.

Algorithm 2 describes the *getNextStrategy* function in detail. Essentially the function builds the next strategy to be evaluated by iterating through all of the adversary types and grabbing the highest valued strategy in their respective pure strategy lists.

6.2 Tuning Leaf Node Computation

At the leaf nodes in our hierarchical type tree we solve the restricted game for each adversary type. For GATE to be exact, we must return all feasible adversary strategies from each leaf node. If we do not return all feasible pure strategies, we have a heuristic approach, which may run well in practice but is not guaranteed to be optimal. Nonetheless, in some cases even for optimality it might suffice for us it might suffice for us to return only some promising strategies. Below we note a special condition under which it is optimal to just consider one adversary strategy at each leaf in the hierarchical tree.

Lemma 1. *Let $\mathbf{n}_\theta$ represent the optimal allocation against type θ at the leaf. Let $\mathbf{n}_\theta[C_\theta]$ be the part of the allocation that is assigned to screenees in screenee category C_θ. If the strategy $\mathbf{n}$ formed by putting together all $\mathbf{n}_\theta[C_\theta]$: $\mathbf{n} = \sum_\theta \mathbf{n}_\theta[C_\theta]$ is feasible then $\mathbf{n}$ is the optimal defender strategy and the single adversary best response for each single type is the adversary best response in the overall game.*

Proof Sketch. First, note that the strategy $\mathbf{n}$ achieves the payoff $\sum_\theta z_\theta s^*_\theta$, where s^*_θ is the defender utility in the restricted game with just the type θ. Also, clearly s^*_θ is an upper bound on the defender utility for the restricted game. By the result from [8], the upper bound on the defender utility is $\sum_\theta z_\theta s^*_\theta$ which is achieved by $\mathbf{n}$. □

We can use Branch-and-Guide as described earlier to return fewer promising adversary strategies, but the question that arises when using Branch-and-Guide at the leaf nodes is how to compute the upper bounds for the nodes on the right of the tree (Recall for non-leaf nodes this upper bound is computed from the upper bounds that are propagated up from each child). One approach to compute this upper bound is to adapt the ORIGAMI [9] approach; ORIGAMI is the fastest technique for solving non-Bayesian general-sum security games without resource scheduling constraints. The underlying idea is to solve an LP to minimize the utility of the adversary by inducing the largest possible attack set (set of targets that are equally and most attractive for the attacker) and [9] shows this provides the optimal defender utility in games without scheduling constraints. However, even for a TSG with a single time window and a single adversary type, ORIGAMI may not provide the optimal solution.

Therefore, ORIGAMI cannot be applied directly to find upper bounds on the adversary pure strategies at the leaf nodes. Thus, we provide *UpperBoundLP*, shown below, to calculate the upper bound at the leaf nodes in the hierarchical tree.

$$\min_{\mathbf{n},\mathbf{q},\mathbf{s},\mathbf{x}} \quad k_{\theta'} \tag{12}$$

$$s.t. \quad k_{\theta'} \geq x^w_{c,m} U^d_{\theta',c} + (1 - x^w_{c,m}) U^u_{\theta',c} \;\forall w, \forall c, \forall m \tag{13}$$

$$x^w_{c,m} = \textstyle\sum_{t \in T} E^t_m \frac{n^w_{c,t}}{N^w_c} \qquad \forall w, \forall c, \forall m \tag{14}$$

$$\textstyle\sum_{t \in T} I^t_r \sum_{c \in C} n^w_{c,t} \leq L^w_r \qquad \forall r, \forall w \tag{15}$$

$$\textstyle\sum_{t \in T} n^w_{c,t} \leq N^w_c, \quad n^w_{c,t} \geq 0 \qquad \forall c, \forall w \tag{16}$$

The objective function 12 minimizes the attacker's utility for the adversary type $\theta^{'}$. Equation 13 enforces that the adversary payoff be the maximal payoff for the adversary given a marginal strategy $\mathbf{n}$.

Equations 14-16 enforce the resource constraints from the original game. This LP uses a slightly modified formulation when enforcing the capacity constraints. Namely, replace Equation 7 with 16, i.e., we relax the constraint that every passenger in a given category c for a time window w must be screened.

Theorem 2. *UpperBoundLP provides an upper bound on the screener utility $d_{\theta'}$ for a non-Bayesian TSG.*

Proof. By relaxing constraint 7 to constraint 16, we can only expand the attack set of the adversary from the original formulation. This happens as Equation 16 allows for an adversary to not be screened which can only increase their utility for targets, thus possibly increasing their attack set. Since the adversary breaks ties in the screener's favor this can only increase the screener's possible utility. □

The above approach serves two purposes: we enormously reduce the set of strategies that are sent up to the parent even if all the adversary strategies are evaluated in the Branch-and-Guide tree, as the attack set is much smaller than the set of all feasible strategies. The running time is also reduced, given the small attack set and the efficient LP to obtain this attack set and upper bounds.

6.3 Tuning Non-leaf Node Computation

Section 6.2 focused on reducing the number of adversary pure strategies returned from the leaf nodes. However, another very important area to prune the search space is at the interior nodes in the binary tree. One way to approach this problem is using Branch-and-Guide in these nodes and stopping the search once the current best solution is better than the next highest upper bound. This can possibly provide significant speed-ups in terms of the computation speed of GATE but it is not quite enough. Unfortunately, in our experiments it turned out that most of the joint adversary pure strategies were evaluated in the interior nodes by MGA as the stopping condition for Branch-and-Guide was almost never met.

Thus, we take inspiration from column generation and security games literature [17] where column generation is stopped if the solution quality does not change much with new columns being added. This heuristic almost always provides a very good approximation. We adopt this same principle so that when evaluating adversary strategies in the Branch-and-Guide approach if the current solution quality does not change over the next K strategies then we can stop the computation and declare the current solution as the final solution with the adversary strategies evaluated so far propagated to the parent. This approach can also be adopted at the root. Here K is a parameter that we can vary, but we use $K = 30$ for the experiments as it seems to work the best for our problem. This approach serves two purposes: (1) it reduces the run time at each intermediate node and the root node and (2) it reduces the number of adversary pure strategies propagated up the tree.

GATE-H then allows us to solve large-scale Bayesian general-sum TSGs. Further, empirically these heuristics maintain high solution quality while decreasing the runtime by an order of magnitude.

7 Evaluation

We evaluate GATE-H using synthetic examples from the passenger screening domain as real world data is not available. We solved the LPs and MILPs using CPLEX 12.5 with the barrier method, as this was found to work the best, on USC's HPC Linux cluster limited to one Hewlett-Packard SL230 node with 2 processors. The adversary and screener payoffs are generated uniformly at random with $U_a^u \in [2, 11]$ and $U_s^u \in [-1, -10]$. For both the adversary and the screener we set $U^d = 0$. The default values for the experiments

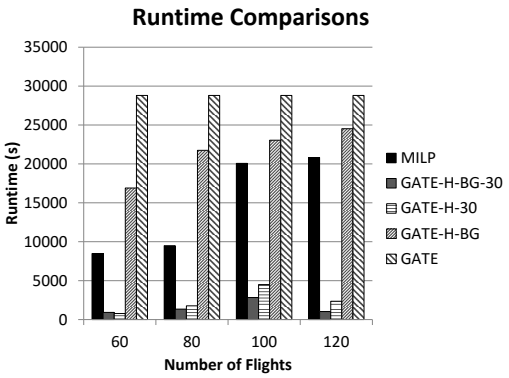


Figure 4. Runtime Comparison - MILP and GATE

are 6 adversary risk levels, 5 screening resource types, 10 screening teams, 2 attack methods and 3 time windows unless otherwise specified. For all experiments the capacity resource constraints also remain constant. All results are averaged over 20 randomly generated game instances.

Scaling Up and Solution Quality The first experiment tests the scalability of each approach and provides solution quality information for GATE-H relative to the *MarginalStrategyMILP*. This experiment provides information about the trade-off between runtime and solution quality for our heuristic algorithm. The four different variations of GATE that were tested are: (1) GATE which evaluates all adversary pure strategies in each of the restricted games and only uses Branch-and-Guide at the root, (2) GATE-H-BG which uses the Branch-and-Guide heuristic in all nodes, (3) GATE-H-K which uses the $K = 30$ cutoff in all nodes and (4) GATE-H-BG-K which uses both Branch-and-Guide and the K cutoff in all nodes. In Figure 4 we show the runtime results for all of the algorithms. On the x-axis we vary the number of flights from 60 to 120 in increments of 20. On the y-axis is the runtime in seconds. For example, for 80 flights *MarginalStrategyMILP* takes almost 10,000 seconds to finish. GATE did not finish in any of the instances showing it cannot scale to large TSG instances. GATE-H-BG also does not perform well as it fails to beat the average runtime of the MILP over all of the instances. As can be seen GATE-H-BG-30 and GATE-H-30 significantly reduce the runtime with an average of a 10 fold improvement over all cases and GATE-H-BG-30 providing a 20 fold speed up at 120 flights.

In Figure 5 we compare the solution quality of the MILP with GATE-H-BG-30 and GATE-H-30. We do not include GATE and GATE-H-BG as they do not finish in a majority of instances making it difficult to compare the solution quality. On the x-axis we again vary the number of flights from 60 to 120 in increments of 20. On the y-axis is the screener's utility. For instance, for 60 flights GATE-H-BG-30 returns an average screener utility of -0.5974. As the graph shows the average solution quality loss over these game instances is always less than .0411 for both GATE-H-BG-30 and GATE-H-30 compared to the MILP. These results show that both GATE-H-BG-30 and GATE-H-30 provide good approximations for large scale TSGs.

Our next experiment aimed to test the ability of GATE-H-BG-30 and GATE-H-30 to scale up to much larger TSG instances. The re-

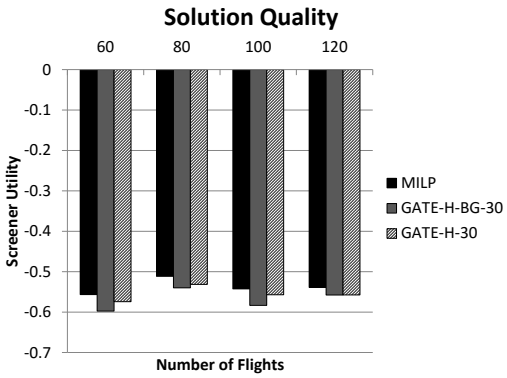


Figure 5. Solution Quality - MILP and GATE

sults are shown in Figure 6. On the x-axis we increase the number of flights from 160 to 220 in increments of 20. On the y-axis we show the average runtime in seconds to solve each of the TSG instances. For example, GATE-H-BG-30 took on average 2,396 seconds to solve a game with 200 flights. An interesting trend here is that the runtime peaks at 180 flights and starts to decrease afterward. This trend could be related to the resource saturation problem as seen in other security games [7], where the observation is that resource optimization is easiest when the resources available are comparatively small or equal to the number of targets and becomes difficult when this is not the case.

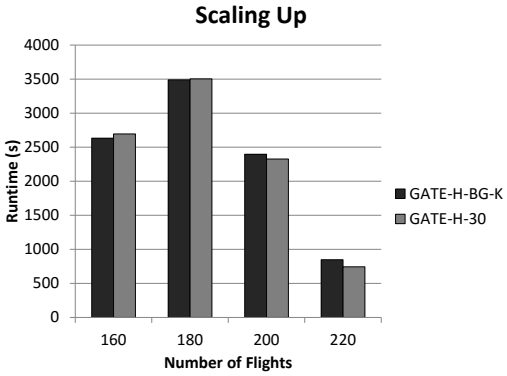


Figure 6. Scaling Up to Larger TSG Instances

Moving Towards Zero Sum The last experiment aimed to test what happens to the solution quality of GATE-H-BG-30 and GATE-H-30 as the game payoffs move toward zero-sum. For this experiment, we use 40 flights and have only one time window as the MILP does not scale in these instances. We vary an r-coefficient from 0 to -0.9 in increments of -0.1, where $r = 0$ means there is no correlation between the attacker's and screener's payoffs and $r = -1$ means the game is zero-sum (Note: -1 is not tested as there is a specialized algorithm to deal with that case where our techniques are not useful). In previous experiments we do not explicitly set a correlation between the adversary's and screener's payoffs although it may be the case. On the x-axis is the r-coefficient and the y-axis shows the screener's utility. For example, when $r = 0$ the *MarginalStrategyMILP* returns a screener utility -0.889, GATE-H-BG-30 returns a screener utility -0.917 and GATE-H-30 returns a solution quality of -0.931. In this

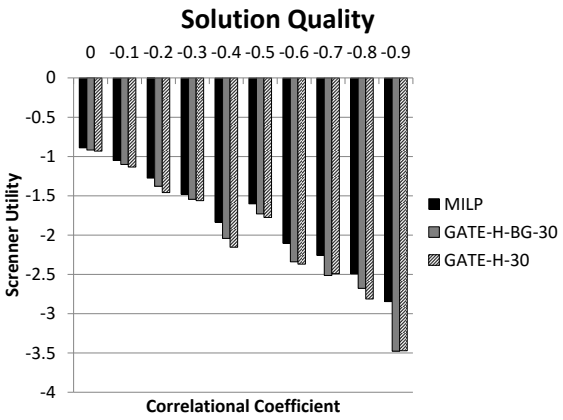


Figure 7. Solution Quality - Moving to Zero Sum

experiment an interesting trend appears. As we move toward zero-sum games the relative performance of both GATE-H-BG-30 and GATE-H-30 progressively worsens. However, until the game payoffs are nearly zero-sum ($r = -0.9$) both GATE-H variations do have a solution quality loss greater than 11.385%. This experiment again shows that both GATE-H-BG-30 and GATE-H-30 provide good approximations in general-sum TSGs. (A careful reader might notice in Figures 5 and 5 that there are slight differences in solution quality between GATE-H-BG-30 and GATE-H-30, however these are not statistically significant.)

8 Summary

The TSG model provides an extensible and adaptable model for game-theoretic screening in the real world. It improves upon previous models in security games that fail to capture important properties of the screening domain, e.g., the presence of non-player screenees in the game and complex team capacity constraints. The model also improves on work done on threat screening, such as screening stadium patrons [15], cargo container screening [2], and screening airport passengers [12, 11].

Previous work done on TSGs [4] focused on the Bayesian zero-sum case and in this paper we extend TSGs to the Bayesian general-sum case. We provide four contributions to accomplish this task: (1) the GATE algorithm which efficiently solves large scale Bayesian general-sum TSGs, (2) the Branch-and-Guide approach which combines branch-and-bound search and MGA in order to efficiently solve nodes in the hierarchical tree, (3) heuristics that speed up the computation of GATE, and (4) experimental evaluation of GATE showing the scalability of our algorithm.

Using the aforementioned contributions this paper presents a practical approach for solving Bayesian general-sum TSGs that scales up to problem sizes encountered in the real world. Thus, with this paper we hope to increase the applicability of TSGs by providing techniques for solving large scale Bayesian general-sum TSGs.

Acknowledgments: This research was supported by the United States Department of Homeland Security through the National Center for Risk and Economic Analysis of Terrorism Events (CREATE) under award number 2010-ST-061-RE0001 and by MURI grant W911NF-11-1-0332

REFERENCES

[1] AAAE, 'Transportation security policy', Technical report, American Association of Airport Executives, (2014).

[2] Saket Anand, David Madigan, Richard Mammone, Saumitr Pathak, and Fred Roberts, 'Experimental analysis of sequential decision making algorithms for port of entry inspection procedures', in *Intelligence and Security Informatics*, 319–330, Springer, (2006).

[3] Cynthia Barnhart, Ellis L Johnson, George L Nemhauser, Martin WP Savelsbergh, and Pamela H Vance, 'Branch-and-price: Column generation for solving huge integer programs', *Operations research*, **46**(3), 316–329, (1998).

[4] Matthew Brown, Arunesh Sinha, Aaron Schlenker, and Milind Tambe, 'One size does not fit all: A game-theoretic approach for dynamically and effectively screening for threats', in *AAAI conference on Artificial Intelligence (AAAI)*, (2016).

[5] Vincent Conitzer and Tuomas Sandholm, 'Computing the optimal strategy to commit to', in *Proceedings of the 7th ACM conference on Electronic commerce*, pp. 82–90. ACM, (2006).

[6] Manish Jain, Erim Kardes, Christopher Kiekintveld, Fernando Ordónez, and Milind Tambe, 'Security games with arbitrary schedules: A branch and price approach.', in *AAAI*, (2010).

[7] Manish Jain, Kevin Leyton-Brown, and Milind Tambe, 'The deployment-to-saturation ratio in security games', *Target*, **1**(5), 5, (2012).

[8] Manish Jain, Milind Tambe, and Christopher Kiekintveld, 'Quality-bounded solutions for finite bayesian stackelberg games: Scaling up', in *International Conference on Autonomous Agents and Multiagent Systems*, (2011).

[9] Christopher Kiekintveld, Manish Jain, Jason Tsai, James Pita, Fernando Ordez, and Milind Tambe, 'Computing optimal randomized resource allocations for massive security games', in *The Eighth International Conference on Autonomous Agents and Multiagent Systems*, (2009).

[10] Joshua Letchford and Vincent Conitzer, 'Solving security games on graphs via marginal probabilities.', in *AAAI*. Citeseer, (2013).

[11] Laura A McLay, Adrian J Lee, and Sheldon H Jacobson, 'Risk-based policies for airport security checkpoint screening', *Transportation science*, **44**(3), 333–349, (2010).

[12] Xiaofeng Nie, Rajan Batta, Colin G Drury, and Li Lin, 'Passenger grouping with risk levels in an airport security system', *European Journal of Operational Research*, **194**(2), 574–584, (2009).

[13] Praveen Paruchuri, Jonathan P Pearce, Janusz Marecki, Milind Tambe, Fernando Ordonez, and Sarit Kraus, 'Playing games for security: An efficient exact algorithm for solving bayesian stackelberg games', in *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2*, pp. 895–902. International Foundation for Autonomous Agents and Multiagent Systems, (2008).

[14] James Pita, Manish Jain, Janusz Marecki, Fernando Ordóñez, Christopher Portway, Milind Tambe, Craig Western, Praveen Paruchuri, and Sarit Kraus, 'Deployed armor protection: the application of a game theoretic model for security at the los angeles international airport', in *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems: industrial track*, pp. 125–132. International Foundation for Autonomous Agents and Multiagent Systems, (2008).

[15] Brian C Ricks, Brian Nakamura, Alper Almaz, Robert DeMarco, Cindy Hui, Paul Kantor, Alisa Matlin, Christie Nelson, Holly Powell, Fred Roberts, et al., 'Modeling the impact of patron screening at an nfl stadium', in *IIE Annual Conference. Proceedings*, p. 3086. Institute of Industrial Engineers-Publisher, (2014).

[16] Eric Shieh, Bo An, Rong Yang, Milind Tambe, Craig Baldwin, Joseph DiRenzo, Ben Maule, and Garrett Meyer, 'Protect: A deployed game theoretic system to protect the ports of the united states', in *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pp. 13–20. International Foundation for Autonomous Agents and Multiagent Systems, (2012).

[17] Eric Shieh, Albert Xin Jiang, Amulya Yadav, Pradeep Varakantham, and Milind Tambe, 'Unleashing dec-mdps in security games: Enabling effective defender teamwork', in *European Conference on Artificial Intelligence (ECAI)*, (2014).

[18] Jason Tsai, Christopher Kiekintveld, Fernando Ordonez, Milind Tambe, and Shyamsunder Rathi, 'Iris-a tool for strategic security allocation in transportation networks', (2009).

[19] Xiaowen Wang, Cen Song, and Jun Zhuang, 'Simulating a multi-stage screening network: A queueing theory and game theory application', in *Game Theoretic Analysis of Congestion, Safety and Security*, 55–80, Springer, (2015).

[20] Rong Yang, Albert Xin Jiang, Milind Tambe, and Fernando Ordóñez, 'Scaling-up security games with boundedly rational adversaries: A cutting-plane approach', in *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*. AAAI Press, (2013).