Noname manuscript No.
(will be inserted by the editor)

Efficient Solutions for Joint Activity Based Security Games: Fast Algorithms, Results and a Field Experiment on a Transit System[1]

Francesco M. Delle Fave[2] · Eric Shieh[2] · Manish Jain · Albert Xin Jiang · Heather Rosoff · Milind Tambe · John P. Sullivan

Received: date / Accepted: date

Abstract In recent years, several security agencies have been deploying scheduling systems based on algorithmic advances in Stackelberg security games (SSGs). Unfortunately, none of the existing algorithms can scale up to domains where benefits are accrued from multiple defender resources performing jointly coordinated activities. Yet in many domains, including port patrolling where SSGs are in use, enabling multiple defender resources to perform jointly coordinated activities would significantly enhance the effectiveness of the patrols.

To address this challenge, this paper presents four contributions. First, we present SMART (*Security games with Multiple coordinated Activities and Resources that are Time-dependent*), a novel SSG model that *explicitly* represents jointly coordinated activities between defender's resources. Second,

[1] An initial version of the work presented in this article has previously appeared in [34]. In this work, we extend this initial version with the following contributions: (i) we present a large scale real-world experiment, describing in detail how a real-world security deployment problem could be modeled using our SMART framework; (ii) within this experiment, we provide the first head-to-head comparison between game-theoretic schedules (generated by our algorithm SMART_H) and human-generated schedules, presenting results showing some of the benefits that game-theoretic scheduling can provide; (iii) we present new simulations where we analyze the performance of TSP ordering heuristic developed to scale up SMART_H and (iv) where we evaluate the effectiveness of using ORIGAMIP to prune nodes of the branch-and-price tree; (v) finally, we provide additional detailed examples and discuss significant new related work and future work.

[2] This article is the results of the joint effort of Francesco Maria Delle Fave and Eric Shieh. Both of them should be considered first authors of this work.

Francesco M. Delle Fave, Eric Shieh, Heather Rosoff, Manish Jain, Albert Xin Jiang, Milind Tambe
University of Southern California, CA 90089
E-mail: {dellefav,eshieh,rosoff,manishja,jiangx,tambe}@usc.edu

John P. Sullivan
Los Angeles County Sheriff's Department
E-mail: jpsulliv@lasd.org

we present two branch-and-price algorithms, SMART_O—an optimal algorithm, and SMART_H— a heuristic approach, to solve SMART instances. The two algorithms present three novel features: (i) a novel approach to generate individual defender strategies by ordering the search space during column generation using insights from the Traveling Salesman Problem(TSP); (ii) exploitation of iterative modification of rewards of multiple defender resources to generate coordinated strategies and (iii) generation of tight upper bounds for pruning using the structure of the problem. Third, we present an extensive empirical and theoretical analysis of both SMART_O and SMART_H. Fourth, we describe a large scale real-world experiment whereby we run the first head-to-head comparison between game-theoretic schedules generated using SMART_H against schedules generated by humans on a one-day patrol exercise over one train line of the Los Angeles Metro System. Our results show that game-theoretic schedules were evaluated to be superior to ones generated by humans.

Keywords Game theory, Security, Stackelberg Games, Simulations, Field Evaluation

1 Introduction

In recent years, research in Stackelberg Security Games (SSGs) has led to the development of several decision aids which assist real-world security agencies in security resource allocation, e.g., to deploy patrols and checkpoints to protect targets from terrorists and criminals [5, 11, 15, 31, 38, 41, 43]. Examples include ARMOR and GUARDS for airport security [31, 32], IRIS for allocating security personnel to international flights of US carriers [18], PROTECT for randomizing the patrols for security of ports and passenger ferries in ports such as New York, Boston, and Los Angeles [14, 33], and TRUSTS for patrolling Metro trains in Los Angeles [44].

In all these applications, a SSG is used to model a two-player game between a defender (the security agency) and an adversary which may be of multiple types.[1] The defender commits to a mixed strategy— a randomized resource allocation specified by a probability distribution over deterministic schedules— which takes into account the adversary's best response to his observation of the mixed strategy.

Most of the applications discussed above require generating mixed strategies defined over an *exponential* number of defender schedules. Hence, generating an effective allocation of security resources is often a computationally expensive task. For this reason, most applications depend heavily on the significant advances in fast algorithms for solving SSGs [17, 38]. For example, IRIS requires the use of a branch-and-price approach to handle the exponentially-many possible allocations of security resources to international flights of US carriers [17]. Similarly, TRUSTS solves the patrol problem as a network flow

[1] By convention in security games literature, the defender is referred to as "she" and the adversary as "he".

optimization problem, using a transition graph, to scale up to the number of possible schedules to patrol a train line [20, 44].

However, despite all such significant advances, scaling up remains a significant issue in advancing the scope of SSGs. A major drawback of the current algorithms is their failure to scale up to SSGs where multiple defender resources explicitly perform joint activities, i.e., games where coordination in space and time will provide the defender with additional benefits [17, 30, 41]. To date, neither general purpose SSG algorithms [12, 30], nor special purpose SSG algorithms [17] can scale up to handle these joint activities. Yet, joint activites are an important aspect of real-world security. For example, the algorithm used in PROTECT [33] only focuses on one boat patrols. Yet, if a single boat is, perhaps, 50% effective in detecting (and hence stopping) a potential adversary attack, a second boat may increase the effectiveness significantly to, perhaps, 80% as the adversary may be forced to react to this second boat. PROTECT is unable to handle such coordination over multiple boats. Similarly, when patrolling a train line, security resources such as explosive detective canine (EK9) teams often patrol train lines in cooperation with other resources. By doing so, their effectiveness is increased. In essence, the key problem for most of the algorithms, discussed above, is that representing this type of joint activity space significantly accelerates the growth in the number of pure strategies, which severely impacts their ability to scale up in several security domains (e.g., port security).

Furthermore, a key question raised for deployed applications of SSGs is the evaluation of their performance in the field. Despite earlier attempts, the actual evaluation of the deployed SSGs-applications in the field is still a major open challenge [33]. A significant number of practical constraints (e.g., time to train the officers, availability of personnel to organize, run and evaluate the experiment) limits the ability of researchers to conduct head-to-head comparisons between SSGs-applications and human schedulers. Hence, a systematic study that evaluates the benefits using a head-to-head comparison is still missing from the literature.

To address these shortcomings, this paper presents four contributions. The first contribution is SMART, *Security games with Multiple coordinated Activities and Resources that are Time-dependent*, a model extending the framework of security games to *explicitly represent* jointly coordinated activities. The second contribution consists of two algorithms. We present SMART_O, an optimal algorithm to compute optimal defender strategies for SMART problems and SMART_H, a heuristic iterative procedure to achieve further speed-up over SMART_O. Both SMART_O and SMART_H use a branch-and-price algorithm – an algorithm composed of branch-and-bound and column generation – to deal with the large strategy space of the domain [4]. These algorithms exploit the structure of the joint coordinated activities to gain speed up, based on the following key ideas: (i) use of insights from the Traveling Salesman Problem (TSP) to order the search space during column generation, especially in SMART_H, while maintaining coordination; (ii) efficient greedy computation of patrols per resource via iterative modification of rewards to generate a joint

patrol, during column generation, and (iii) generation of tight upper-bounds within the branch-and-bound component by exploiting scheduling constraints to allow pruning of the search space based on the sub-modular property of joint activities. Our third contribution is the analysis of the performance of both $\textsc{Smart}_{\mathrm{O}}$ and $\textsc{Smart}_{\mathrm{H}}$ in solving instances of Smart. We analyze the quality of the solutions generated by $\textsc{Smart}_{\mathrm{H}}$ and evaluate both algorithms in simulation comparing their runtime and solution quality.

Finally, our fourth contribution is the real-world evaluation of the game-theoretic schedules generated using $\textsc{Smart}_{\mathrm{H}}$. This evaluation constitutes the largest scale experiment evaluating the performance of SSGs in the field. We present results from a massive transit full-scale exercise (FSE), a real-world experiment whereby 80 security officers coordinated their activities to patrol 10 stations of a metro line for 12 hours. The purpose of the exercise was a head-to-head comparison between SSG-based schedules, generated using $\textsc{Smart}_{\mathrm{H}}$, against human-generated schedules. We were able to evaluate the schedule generation process, as well as provide a thorough evaluation of the performance of both schedules as conducted by a number of security experts located at each of the ten stations during the entire length of the exercise. Our results show that our game-theoretic approach, based on $\textsc{Smart}_{\mathrm{H}}$, was able to significantly cut the schedule generation effort for humans compared to manual scheduling. Yet, game-theoretic scheduling was able to generate schedules similar to the ones generated by humans in terms of number of joint activities. In addition, game-theoretic schedules were able to address the *comprehensive* security of the train line by having the different teams patrol *all* the different levels of the stations (e.g., the platform level, the street level and the mezzaning level). Finally, the game-theoretic schedules allocated the more effective teams to the more important stations. These last two factors, which were missing from the human-generated schedules, led security experts to concur that the game-theoretic schedules were more effective in providing security than the human-generated schedules.

The overall conclusion from this real-world exercise is that our game-theoretic schedules, generated by $\textsc{Smart}_{\mathrm{H}}$ were able to perform at least equivalently to (and in fact better than those) generated by human schedulers. This indicates that we could save precious time so security experts could focus on maintaining security rather than on generating schedules. Overall, the data that we were able to collect constitutes a source of information which can be used for evaluating the current status of research in SSGs, and to understand new directions where to take such research.

The remainder of this paper is organized as follows. Section 3 presents the Smart model for incorporating jointly coordinated activities into SSGs. Section 4 presents $\textsc{Smart}_{\mathrm{O}}$, the optimal algorithm to solve Smart problem instances. Section 5 then presents $\textsc{Smart}_{\mathrm{H}}$ the heuristic that we developed to increase the scalability over $\textsc{Smart}_{\mathrm{O}}$. Next, Section 6 discusses both our simulation and our real-world experiment. Finally, Section 7 summarizes the paper and introduces some future and ongoing work.

2 Related Work

Stackelberg security games (SSGs) have gathered significant attention in literature [5, 13, 22, 23, 24, 25, 26]. As discussed in the previous section, SSG models and algorithms have been used to build decision aids including ARMOR [31], IRIS [39], GUARDS [32], PROTECT [33], TRUSTS [44] and RaPtoR [42]. All these decision aids have been developed to assist the security of transportation infrastructure including ports, airports and train lines. However, most of these decision aids do not model joint coordinated activities. The IRIS system models some of this coordination by assigning a negative infinite weight to the joint action of two Federal Air Marshals (FAMS) taking the same flight, explicitly restricting the maximum number of FAMS on any flight to one (see the work of Jain et al. [17] for more details). This type of solution, however, does not model more complex forms of joint effectiveness as we will do in the rest of this paper. In fact, most decision aids, including ARMOR, GUARDS, PROTECT, TRUSTS and RaPtoR, do not account for jointly coordinated activities. They allocate security resources without considering the benefits that could be accrued by different resources combining their effort. As a consequence, they are generating schedules that are not as effective as they could be. Indeed, recent work by Jiang et al. [19], analyzes the loss incurred by miscoordination between defender's resources and derives theoretical bounds indicating that such loss may be extremely high in several scenarios.

Current literature presents two gaps that limit the ability of the existing SSG models and algorithms to address jointly coordinated activities between defender's resources. First, existing SSG models assume that a single resource is 100% effective in protecting any single target [17, 21]. They do not consider varying effectiveness of joint activities, i.e., they all assume that no benefits can be accrued if the resources coordinate their activities. Second, from an algorithmic perspective, even algorithms that are intended to achieve scale-up in presence of constraints, such as ASPEN [17], cannot scale up to SSGs incorporating joint activities due to the significant increase in the strategy space.

Similarly, the challenge of deploying game-theoretic schedules in the field has not been addressed by research in SSGs. Despite some initial evaluation of the PROTECT system [33], a head-to-head comparison between game-theoretic schedules and human generated schedules, the way in which most security agencies allocate their resources, is still missing from literature. However, this type of study would be extremely useful to advance the state-of-the-art of game-theoretic scheduling, because it would allow us to measure, for the first time, the actual performance of such schedules when deployed in the real world. From this perspective, our work shares many ideas with literature on game theory in the field. This line of research has focused on showing equilibrium concepts in the human and animal activities [8, 29]. Our work shares their enthusiasm of taking game theory to the field, but fundamentally focuses on *algorithmic deployments* and the impact of such algorithms. In addition, this work shares some similarities with literature in crime prevention [9, 10].

Our idea is to evaluate a new form of policing of a transportation network based on game-theoretic scheduling. We measure the performance of strategic recommendations in the form of schedules that can be used to mitigate crime and terrorism within areas such as ports or transit systems.

In addition to decision aids and security allocation, research in SSGs has also addressed problems of multi-robot patrolling. More specifically, research has developed the multi-robot adversarial patrolling games (MAPG) framework, a restricted type of SSG, which considers the problem of coordinated patrols of multiple robots around a closed area with the existence of an adversary attempting to penetrate into the area [1, 2]. The penetration requires time and the defender should identify the attacker during his attempt. Similarly, the work from Sless et al. [36] requires the robots to physically inspect penetration attempts for a given time period. More specifically, Sless et al. [36] investigate the problem of coordinated attacks, in which the adversary initiates two attacks in order to maximize its chances of successful penetration, assuming a robot from the team will be sent to examine a penetration attempt. Such MAPGs patrols are frequency-based patrols in adversarial environments, but do not consider targets of varying importance and the impact of *joint activities* [3, 27, 40].

3 The Smart Problem

A Smart problem is an instance of a SSG. A SSG, as discussed in detail in the work of Kiekintveld et al. [21], is a two-player game involving a defender d and an attacker a competing over a set of targets T. The defender has a limited number of resources R and needs to select which targets to protect considering that the attacker is going to conduct a thorough surveillance to exploit any predictable pattern in the defender resource allocation. In a SSG, each target $t \in T$ is assigned a reward $U_d^c(t)$ and a penalty $U_d^u(t)$ to the defender if t is covered and uncovered, respectively, by a defender's resource. Similarly, each target is assigned a reward $U_a^c(t)$ and a penalty $U_a^u(t)$ to the attacker. As discussed by Kiekintveld et al. [21], the payoffs are defined such that $U_d^u(t) < U_d^c(t)$ and $U_a^c(t) < U_a^u(t)\ \forall\ t \in T$. The purpose of each player then is to maximize their expected payoffs defined in equations 4 and 5. In an optimal solution of a SSG, the defender plays a mixed strategy, i.e., a probability distribution over the different targets, which intelligently allocate the defender resources given the importance of each target and considering the behavior of the attacker [12].

In a Smart problem instance, each resource chooses an activity from the set $\mathcal{A} = \{\alpha_1, \alpha_2, \dots \alpha_K\}$ for each target $t \in T$. Each resource $r \in R$ is assigned a graph $G_r = (T, E_r)$, where the set of vertices T represents the set of targets to patrol and the set of edges E_r represents the connectivity between such targets. Each edge $e \in E_r$ is assigned a time value $\tau(e)$ representing the time that it takes for one defender resource r to traverse e. Each graph encodes the motion of different resources. For example, security assistants patrol the

R	Number of defender resources, subscripted by r
$G_r = (T, E_r)$	Graph of the input problem instance
T	Set of targets
t_b	Home base
$E_r : \{e(t_i, t_j)\}$	Set of edges
$\tau(e(t_i, t_j))$	Time required to traverse the edge e
$\tau(\alpha)$	Time required to conduct activity α
$\texttt{eff}(\alpha)$	Effectiveness of activity α
$\texttt{eff}(\alpha_i, \alpha_j)$	Effectiveness of joint activity $\langle \alpha_i, \alpha_j \rangle$
$\mathbf{P}$	Set of pure strategies of the defender
$\omega_t(\mathbf{P}_i)$	Effective coverage of t in $\mathbf{P}_i$
Γ_r	Maximum time allowed for an individual patrol for resource r
W	Time window for a joint activity
$\mathcal{X}^r$	The set of pure strategies for resource r

Table 1 Notation Table

stations of a train line by taking the trains. In contrast, sheriffs and EK9 units use a car. They can go from one end to the other of a train line without having to cross each station. Similarly, aerial patrols can move freely and reach any area of a port, whereas boat patrols might be constrained to take certain routes. The notation used in SMART is described in Table 1.

The attacker's pure strategy space is the set of all targets, T. A pure strategy for the defender is a set of routes, one route X_i for each resource. Formally, each patrol route is defined as an ordered list of 3-tuples $X_i = [X_i^1, \ldots, X_i^j, \ldots]$. The jth 3-tuple $X_i^j = (t, \alpha, \gamma)$ represents a time-action segment for defender resource i: she conducts and completes activity α at target t at time γ. Each time-action segment is different since different activities might require different amounts of time and have a different effect on the target to protect (as discussed below).

Each patrol route starts and ends at the same, pre-defined, home base $t_b \in T$, i.e., $X_i^1.t = t_b$ and $X_i^{|X_i|}.t = t_b$. The total route length of each resource's patrol is upper bounded by a specific value Γ_r as follows:

$$\overbrace{\sum_{j=1}^{|X_i|-1} \tau(e(X_i^j.t, X_i^{j+1}.t))}^{\text{traversal time}} + \overbrace{\sum_{j=1}^{|X_i|} \tau(X_i^j.\alpha)}^{\text{time for activities}} \leq \Gamma_r \ \forall \ X_i \tag{1}$$

$\mathcal{X}^r$ is defined as the set of pure strategies for resource r and the set of joint pure strategies $\mathbf{P}$ is given by the cross-product of pure strategies for each resource, i.e., $\mathbf{P} = \prod_{r=1}^{R} \{\mathcal{X}^r\}$.

SMART is unique since it explicitly models *joint activities*, or activities coordinated in space and time between multiple defender resources. The defender is said to conduct a joint activity $\langle \alpha_i, \alpha_j \rangle$ in her pure strategy if there exists at least two tuples $(t_i, \alpha_i, \gamma_i) \in X_i$ and $(t_j, \alpha_j, \gamma_j) \in X_j$ in the defender's pure strategy such that $t_i = t_j$ and $|\gamma_i - \gamma_j| \leq W$. In other words, i.e., the two

activities are on the same target and are within a time window of width W. Here, the time width W represents the minimum interval of time within which two different activities have a joint effect. For instance, if one aerial and one boat patrol explore the same area one after the other within a time frame of 10 minutes, their effectiveness will be much larger then if they were patrolling one after the other but within a time frame of 30 minutes. In the former case, it can be assumed that they were conducting a joint patrol action. In contrast, in the second case, given the large temporal distance, the two actions can be considered individually.

For each activity α_i, $\texttt{eff}(\alpha_i)$ represents the individual effectiveness[2] of the activity α_i. This effectiveness ranges from 0% to 100%, and measures the probability that the defender will be able to successfully prevent an attack on target t if such an attack overlaps with the activity α_i at t that the defender is conducting . This is similar to what was done in PROTECT [33]. We define the effectiveness of the joint activity $\langle \alpha_i, \alpha_j \rangle$ as $\texttt{eff}(\alpha_i, \alpha_j)$. While a joint activity may be composed of two *or more* resources – and our experimental results show the benefits of n-ary joint activities in Section 6.1 – in this section we focus on joint activities composed of two resources for simplicity of explanation. In this case, a joint activity composed of two resources receives the maximum effectiveness and any additional resource visiting target t in the time window will have no additional benefit.Thus, it is possible to define a total order relation $\geq$ on $\mathcal{A}$ such that $\alpha_i \geq \alpha_j$ if and only if (1) $\texttt{eff}(\alpha_i) \geq \texttt{eff}(\alpha_j)$ and (2) $\texttt{eff}(\alpha_i, \alpha_k) \geq \texttt{eff}(\alpha_j, \alpha_k), \forall \alpha_k$. In other words α_i provides a greater effectiveness than α_j.

Given a set of activities $S = \{\alpha_i\}_{i=1\ldots k}$ on a target within the same time window, labeled so that $\alpha_i \geq \alpha_j$ for all $i > j$, we extend the notation of $\texttt{eff}$ such that $\texttt{eff}(\{\emptyset\}) = 0$, $\texttt{eff}(S) = \texttt{eff}(\alpha_1)$ if $S = \{\alpha_1\}$, i.e., $|S| = 1$, and $\texttt{eff}(S) = \texttt{eff}(\alpha_1, \alpha_2)$ if $S = \{\alpha_i\}_{i=1\ldots k}$, i.e., $|S| > 1$. $\texttt{eff}(S)$ represents the maximum effectiveness of an individual or a joint activity over a set S of activities performed at a target within the same time window. One interesting aspect to understand about the operator $\texttt{eff}()$ is whether it is submodular or not. We define $\texttt{eff}()$ as *submodular* if for all $S_1 \subseteq S_2$ and all α_i the following condition holds:

$$\texttt{eff}(S_1 \cup \{\alpha_i\}) - \texttt{eff}(S_1) \geq \texttt{eff}(S_2 \cup \{\alpha_i\}) - \texttt{eff}(S_2) \tag{2}$$

This means that each additional activity performed has diminishing gains in effectiveness. The reason why we are interested in submodularity is that whenever it holds it becomes possible to define an approximate greedy algorithm, $\textsc{Smart}_{\text{H}}$, which provides performance guarantees on the quality of the solution that it calculates. More specifically, in Section 5, we formally demonstrate that whenever $\texttt{eff}()$ is submodular, the solutions generated by $\textsc{Smart}_{\text{H}}$ are upper bounds to the optimal solutions of the problem (see Equation 26).

Therefore, the submodularity property is crucial from a practical perspective. Whenever it holds in a real-world domain, we can provide theoretical

[2] We associate effectiveness with activities and not with targets, assuming that each activity is equally effective at all targets.

guarantees on the performance of the deployed officers. One example of such domain is port security, if one boat from the US Coast Guard is exploring a specific area of a port, any additional boat is unlikely to provide additional benefit in terms of deterrence effect or ability to capture criminals. In contrast, submodularity will not hold in domains where the defender has two different resources that only provide benefit when they are acting together. As we will see in Section 6.2, this is the case of the train domain, where some security resources (e.g., the EK 9) will be characterized by a null individual effectiveness but a non-zero joint effectiveness. If the submodularity property does not hold, the SMART_H algorithm is still able to solve and generate an approximate solution of the problem, however nothing can be said about such solution's quality.

The expected utilities $U_d(\mathbf{P}_i, t)$ and $U_a(\mathbf{P}_i, t)$ for both players, when the defender is conducting pure strategy $\mathbf{P}_i$ (defined as a joint pure strategy for multiple defender resources), and when the attacker chooses to attack target t is given as follows:

$$\omega_t(\mathbf{P}_i) = \max_{\substack{(t,\alpha,\gamma)\in\mathbf{P}_i \\ \{(t,\alpha_l,\gamma_l),(t,\alpha_m,\gamma_m)\}\subseteq\mathbf{P}_i, |\gamma_l-\gamma_m|\leq W}} \{\texttt{eff}(\alpha), \texttt{eff}(\alpha_l, \alpha_m)\} \tag{3}$$

$$U_d(\mathbf{P}_i, t) = \omega_t(\mathbf{P}_i)U_d^c(t) + (1 - \omega_t(\mathbf{P}_i))U_d^u(t) \tag{4}$$

$$U_a(\mathbf{P}_i, t) = \omega_t(\mathbf{P}_i)U_a^c(t) + (1 - \omega_t(\mathbf{P}_i))U_a^u(t) \tag{5}$$

Here $\omega_t(\mathbf{P}_i)$ defined in Equation (3) represents the *effective coverage* of the defender on target t when executing pure strategy $\mathbf{P}_i$. This is computed by taking the maximum effectiveness of either a single or joint activity performed at target t at any time during the defender's patrols. The justification here is that in many domains the time that it takes to prepare and carry out a complex attack on a target, is often longer than the time required to patrol. Hence, we can safely assume that the attacker only cares about the maximum effective activity or joint activity (nonetheless, the formulation could be extended to situations involving shorter attack durations by dividing a patrol based multiple attack periods, a topic we leave for future work). Once the effectiveness $\omega_t(\mathbf{P}_i)$ is computed from the pure strategy $\mathbf{P}_i$, the defender and attacker expected utilities $U_d(\mathbf{P}_i, t)$ and $U_a(\mathbf{P}_i, t)$ are calculated as defined in Equation (4) and (5). The following example illustrates how the effectiveness of each pure strategy is calculated.

Example 1 Consider the problem composed of 5 targets as given by the graph in Figure 1. The travel time on the edges between targets t_i and t_j is denoted as τ_{ij}. Assume that the home base, $t_b = t_1$. Furthermore, consider that the defender has 2 resources, each of which could conduct activities $\{\alpha_1, \alpha_2, \alpha_3\}$, such that α_1 is a thorough inspection of the target premises, α_2 is waiting at the target looking for suspicious behavior and α_3 just transiting through the target area. Tables 2 and 3 give the time required for activity α and the effectiveness `eff` for each individual as well as joint activity. The time at each tuple is computed by summing the distance between the targets and the activity time in Table 2.

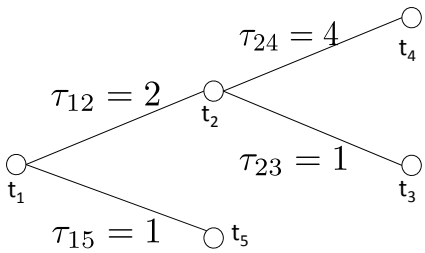


Fig. 1 Example graph of targets and edges

Activity	α_1	α_2	α_3
Effectiveness, $\texttt{eff}(\alpha_i)$	0.5	0.4	0.1
Time	2	1	0

Table 2 Effectiveness of a single activity at any target.

	α_1	α_2	α_3
α_1	0.8	0.7	0.58
α_2	0.7	0.55	0.45
α_3	0.58	0.45	0.11

Table 3 Joint Effectiveness of each joint activity at any target.

An example of a pure strategy for the first defender resource is: $X_1 = [(t_1, \alpha_3, 0), (t_5, \alpha_1, 3), (t_1, \alpha_1, 6)]$. In words, the strategy describes a patrol whereby the first defender resource start at the home base target, t_1, performs activity α_3, go to t_5, performs activity α_1 and then returns back to the base, t_1 to perform activity α_1.

An example of a pure strategy of the second defender resource is: $X_2 = [(t_1, \alpha_3, 0), (t_2, \alpha_3, 2), (t_3, \alpha_3, 3), (t_2, \alpha_3, 4), (t_1, \alpha_2, 7)]$. This strategy describes a patrol where the second defender resource starts at t_1 and travels to t_2, t_3, t_2, and then back to t_1 performing activity α_3 at all targets except at the second visit target t_1, performing activity α_2.

The pure defender strategy considering all defender resources is defined as $\mathbf{P}_1 = (X_1, X_2)$. Assuming that the time window $W = 2$, then the effectiveness of the defender's pure strategy is computed by first determining the most effective single activity for this target, which is α_1 with $\texttt{eff}(\alpha_1) = 0.5$ for target t_1, as shown in Table 2. Looking at the schedule of the two defender resources, we then find the maximum coordinated joint activity that are within the time window $W = 2$. For t_1, this is determined to be $(t_1, \alpha_1, 6)$ for resource 1 and $(t_1, \alpha_2, 7)$ for resource 2, which gives $\texttt{eff}(\alpha_1, \alpha_2) = 0.7$, from Table 3. Thus, $\omega_{t_1}(\mathbf{P}_1) = \max(0.7, 0.5) = 0.7$, as defined in Equation (3). Similarly, $\omega_{t_2}(\mathbf{P}_1) = 0.1$, $\omega_{t_3}(\mathbf{P}_1) = 0.1$, $\omega_{t_4}(\mathbf{P}_1) = 0.0$, $\omega_{t_5}(\mathbf{P}_1) = 0.5$.

Given pure strategy $\mathbf{P}_1$, the defender's expected utility for target t_1 is computed using Equation (4), and is equal to $0.7 \cdot U_d^c(t_1) + 0.3 \cdot U_d^u(t_1)$. The attacker's expected utility for target t_1 given pure strategy $\mathbf{P}_1$ is computed in a similar fashion.

Problem Statement: The objective of the defender is to maximize her expected utility in the SMART problem by computing the optimal mixed strategy given that the attacker will best respond to the defender's strategy.

4 Smart$_{\mathrm{O}}$: Optimal Branch-and-Price Solver

SMART$_{\mathrm{O}}$ is an optimal algorithm to compute solutions for SMART problem instances. It builds upon the ASPEN algorithm [17], an optimal algorithm to solve SSGs based on the *branch-and-price* framework [4]. The two major novelties of SMART$_{\mathrm{O}}$ over ASPEN are the formulation of a slave component capable of handling joint activities (in Section 4.1) and the improved upper bounds on the quality of the solutions recovered by the algorithm (in Section 4.2). The price SMART$_{\mathrm{O}}$ pays for its optimality is its lack of scalability (as discussed later in Section 6.1). Nonetheless, it is useful to understand SMART$_{\mathrm{O}}$'s functioning because it lays the foundation for SMART$_{\mathrm{H}}$, which is a more scalable algorithm. SMART$_{\mathrm{O}}$ also allows us to measure SMART$_{\mathrm{H}}$'s performance on smaller-sized problems.

4.1 Pricing component

The branch-and-price framework constructs a branch-and-bound tree, where for each leaf of the tree, the attacker's target is fixed to a different t'. The objective of the pricing component is to find the best defender mixed strategy $\mathbf{x}$ at that leaf, such that *the best response of the attacker* to $\mathbf{x}$ is to attack target t'. Due to the exponential number of defender pure strategies, the best defender mixed strategy is determined using *column generation*, which is composed of a *master* and *slave* procedure, where the slave iteratively adds a new column (defender strategy) to the master. Each component is defined as follows:

$$\min_{\mathbf{c},\mathbf{x}} - U_d(t', \mathbf{c}) \tag{6}$$

$$U_a(t', \mathbf{c}) \;\geq\; U_a(t, \mathbf{c}) \;\forall t \neq t' \tag{7}$$

$$c_t - \sum_{j \in J} \omega_t(\mathbf{P}_j) x_j \leq 0 \;\forall t \in T \tag{8}$$

$$\sum_{j \in J} x_j = 1 \tag{9}$$

$$x_j \in [0,1] \;\forall j \in J, c_t \in [0,1] \;\forall t \in T \tag{10}$$

Master: The master LP, given in Equations (6) to (10), finds the optimal defender mixed strategy $\mathbf{x}$ given a set of pure strategies J and assuming that the pure strategy of the attacker is set to t' (determined by the leaf node).[3]

[3] For the sake of computation, we formulate the LP as a minimization problem (Equation 6); this will be explained in detail when we describe the slave procedure.

This is similar in formulation to the ERASER algorithm [21]. $U_d(t', \mathbf{c})$ and $U_a(t', \mathbf{c})$ are the utilities of the defender and the attacker respectively when the defender's effective marginal coverage is $\mathbf{c}$ and the attacker attacks t'. Both $U_d(t', \mathbf{c})$ and $U_a(t', \mathbf{c})$ are defined following Equations 4 and 5, e.g., $U_d(t', \mathbf{c}) = c_{t'} U_d^c(t') + (1 - c_{t'}) U_d^u(t')$. For each pure strategy $\mathbf{P}_j$, $\omega_t(\mathbf{P}_j)$ is the effectiveness on t.

Slave: Once the master LP is solved to optimality, the slave problem receives the values of the *duals* of the master LP. The *reduced cost* $\overline{c}_j$ associated with column $\mathbf{P}_j$ is defined as follows:

$$\overline{c}_j = \sum_t y_t \cdot \omega_t(\mathbf{P}_j) - z, \tag{11}$$

where z is the dual variable of Equation (9) and $\{y_t\}$ are the duals of Equation family (8). The reduced cost of a column gives the potential change in the master's objective function when a candidate pure strategy is added to J. The candidate pure strategy with the minimum reduced cost is likely to improve our objective the most [7], since we are minimizing the master in Equation (6).

The objective for the slave problem is to find the column $\mathbf{P}_j$ with the least reduced cost, to add to the current set of columns. In addition, if the least reduced cost is greater or equal to 0, the current master solution is optimal for the full LP. The best column is identified using a mixed-integer linear program (MILP) formulation over the transition graph defined below, *which captures all the spatio-temporal constraints of the problem for handling joint activities and avoids having to enumerate all pure strategies.*

The transition graph $\mathcal{G}_r = (N'_r, E'_r)$ contains nodes $\mathtt{u} = (t, \gamma)$ for each target t and time instant $\gamma \in [0, \Gamma_r]$ if it is possible for the defender to be at target t at time instant γ (the time interval is discretized). Each edge in E'_r is associated with an activity α. An edge $e(\mathtt{u},\mathtt{v},\alpha)$ from node $\mathtt{u}$ to node $\mathtt{v}$ maps to a defender patrol that starts from target $t_\mathtt{u}$ at time $\gamma_\mathtt{u}$, goes to target $t_\mathtt{v}$ and conducts activity α at target $t_\mathtt{v}$. Therefore, we can calculate $\gamma_\mathtt{v}$ as follows:

$$\gamma_\mathtt{v} = \gamma_\mathtt{u} + \tau(t_\mathtt{u}, t_\mathtt{v}) + \tau(\alpha) \tag{12}$$

where $\tau(t_\mathtt{u}, t_\mathtt{v})$ is the time required to traverse from target $t_\mathtt{u}$ to $t_\mathtt{v}$ and $\tau(\alpha)$ is the time required to conduct activity α. The graph contains virtual `source` and `sink` nodes that contain edges to/from the base target t_b to ensure that patrols start and end at t_b.

Example 2 Figure 2 shows a sample transition graph related to the problem presented in Example 1. Here, $t_b = t_1$ and the source has three edges, one for each activity $\alpha_1 - \alpha_3$. Looking at node $\mathtt{u} = (t_1, 0)$, target t_1 is adjacent to t_2 and t_5, so for each of these targets, three edges are added to represent the travel and corresponding activity at that target. For example, if activity α_2 is then performed at target t_2, then the new vertex would be at time $\gamma = 0 + \tau(\alpha_2) + \tau_{12} = 0 + 1 + 2 = 3$, where $\tau_{12} = 2$, and node $\mathtt{v} = (t_2, 3)$ as shown in Figure 2.

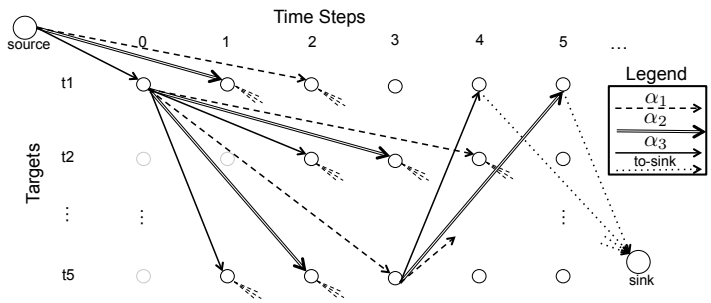


Fig. 2 An Example for the Transition Graph

Slave Problem MILP: We now describe our mixed integer linear programming formulation that identifies the pure strategy $\mathbf{P}_j$. The MILP for the slave problem is given in Equation (13) to (17). This novel MILP component of $\textsc{Smart}_{\text{O}}$ solves for joint activities and generates the optimal defender pure strategy.

$$\min \quad \sum_{t \in T} y_t \cdot \max\{g_t, h_t\} \tag{13}$$

$$\sum_{e \in \texttt{out(u)}} f(e_r) = \sum_{e \in \texttt{in(u)}} f(e_r) \ \forall \ \texttt{u}, r \tag{14}$$

$$g_t = \max_{\substack{e_r \in \texttt{IA}(t), \\ \forall r}} \{f(e_r) \cdot \texttt{eff}(e.\alpha)\} \tag{15}$$

$$h_t = \max_{\substack{e_i, e_j \in \texttt{JA}(r_i, r_j, t), \\ \forall i,j \ \in R}} \{(f(e_i) + f(e_j) - 1) \cdot \texttt{eff}(e_i.\alpha, e_j.\alpha)\} \tag{16}$$

$$f(e_r) \in \{0, 1\} \ \forall e_r, g_t, h_t \ \in [0, 1] \ \forall t \in T \tag{17}$$

This MILP uses one copy of the transition graph for *each* defender resource, where $f(e_r)$ represents the flow on edge e for resource r, and g_t and h_t represent the effectiveness of the defender's individual and joint activities on target t. It only considers the maximum effective activity at target t (Equations (13), (15), and (16)) in accordance with our assumption of the attacker's decision making. In all three equations, the maximum effectiveness is computed using integer variables along with a large constant M. The resulting constraints are similar to the ones used in the DOBSS algorithms [30].

Here, the set $\texttt{IA}(d)$ represents the set of edges in the transition graph such that they represent one resource performing an activity α on target d, and can be represented as:

$$\texttt{IA}(d) = \{\texttt{in}(\texttt{u}_r) | \texttt{u}_r.t = d, \forall \texttt{u}_r \in N'_r, \forall r \in R\}$$

where $\texttt{in}(\texttt{u}_r)$ represents all edges with the target node $\texttt{u}_r$. Similarly, the set $\texttt{JA}(r_i, r_j, d)$ contains pairs of edges $\langle e_i, e_j \rangle$ such that both edges lead to the

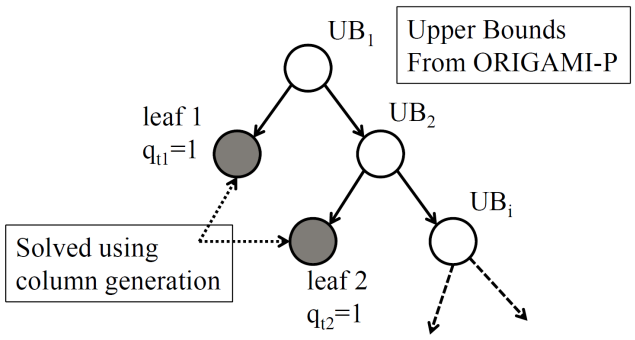


Fig. 3 The structure of branch-and-price

same target d and are separated by a time window no larger than W, corresponding to when resources i and j perform a joint activity on target d. Formally, $\texttt{JA}(r_i, r_j, d) =$

$$\{\langle e_i = (\mathtt{u}, \mathtt{v}), e_j = (\mathtt{u}', \mathtt{v}')\rangle | \mathtt{v}.t = \mathtt{v}'.t = d, |\mathtt{v}.\gamma - \mathtt{v}'.\gamma| \leq W\}.$$

Both sets IA and JA are defined over all transition graphs.

The result from the slave MILP is a set of 0-1 integer flows for each defender resource r. Given these flows, the defender pure strategy $\mathbf{P}_j$ and the effective coverage $\boldsymbol{\omega}(\mathbf{P}_j)$ are generated, and then both are sent back to the master.

4.2 Branch-and-bound component

In our branch-and-price framework, we define a separate branch for each attacker pure strategy, i.e. for each target t. Thus, the leaf node for target $\hat{t}$ has $q_t = 1$ for $t = \hat{t}$ and 0 otherwise. The pricing procedure described earlier is then used to compute the solution for this leaf node. This procedure is repeated for each leaf, after which the best solution obtained thus far is returned as the optimal solution.[4] An example branch-and-bound tree is given in Figure 3. The leaf (*gray*) nodes in the figure represent the pure strategies of the attacker, i.e., where the pricing computation is performed. The non-leaf (*white*) nodes represent the nodes for which upper bounds are obtained using a branch-and-bound heuristic (the branch-and-bound heuristic also determines the ordering of leaves, or attacker's pure strategies, in this tree). The objective of the branch and bound component is (i) to compute upper bounds for each internal node of the tree such that leaf nodes can be pruned thereby requiring less computation, and (ii) to determine an efficient ordering of leaves.

We generate these upper bounds using OrigamiP, a modification of Origami [21] specifically designed to generate tighter upper bounds for Smart problem instances by exploiting the structure of the domain.

[4] Only considering pure-strategies for the attacker is not a limitation; Stackelberg games always exhibit at least one Strong Stackelberg equilibrium where the attacker's best response is a pure strategy [30].

$$\min_{\mathbf{c},\mathbf{f}(\mathbf{e})} \quad k \tag{18}$$

$$0 \leq k - U_a(t, \mathbf{c}) \leq (1 - q_t) \cdot M \ \forall t \in T \tag{19}$$

$$\sum_{e \in \texttt{out(source)}} f(e) = R \tag{20}$$

$$\sum_{e \in \texttt{in(sink)}} f(e) = R \tag{21}$$

$$\sum_{e \in \texttt{out(u)}} f(e) = \sum_{e \in \texttt{in(u)}} f(e) \ \forall \texttt{u} \tag{22}$$

$$c_t \leq \sum_{e=\texttt{(u,v)}|\texttt{v}.t=t} f(e) \cdot \texttt{eff}(\alpha_k) \ \forall t \in T \tag{23}$$

$$c_t \in [0, 1], \ q_t \in \{0, 1\} \ \forall t \in T, \ f(e) \in [0, R] \ \forall e \in E \tag{24}$$

OrigamiP uses the transition graph defined in the slave formulation (Section 4.1). Equations (18)–(19) minimize the attacker's maximum expected utility, $U_a(t, \mathbf{c})$ defined by Equation 5. This utility represents the attacker's utility given the defender's effective marginal coverage is $\mathbf{c}$ and the attacker attacks t. Since, the algorithm is used in the internal nodes of the branch-and-price tree, the attacker's target is fixed to a target t'. Thus, the integer variables q_t representing the attacker's pure strategy are fixed (for target $\hat{t}$, $q_t = 1$ for $t = \hat{t}$ and 0 otherwise) and the original MILP is simplified into an LP. Equations (20)–(22) define the flows of the edges and enforce the flow conservation property. Equation (23) limits the coverage of the defender based on the amount of flow and the respective activity. We can show that OrigamiP satisfies the following proposition:

Proposition 1 OrigamiP *computes upper bounds of the defender expected utility $U_d()$ if* `eff()` *is submodular.*

Proof OrigamiP estimates the effectiveness of a defender patrol on a target as being the sum of the effectiveness of all individual activities on a target. This is an over-estimate of the effectiveness (thereby providing an upper bound on defender utility) if the effectiveness function `eff` is sub-additive, i.e., $\texttt{eff}(\alpha_i) + \texttt{eff}(\alpha_j) \geq \texttt{eff}(\alpha_i, \alpha_j)$, which is the case when `eff` satisfies the submodularity property in (2).

OrigamiP is an LP and therefore solvable in polynomial time. Once the OrigamiP solution has been obtained, the defender's expected utility $U_d(t, \mathbf{c})$ is computed for each target t. The targets are then ordered in *decreasing* order of $U_d(t, \mathbf{c})$. This ordering and computation of upper bounds is then exploited to prune the nodes in the branch-and-price tree.

5 Smart$_{\mathbf{H}}$: Further scaling up Smart

We now present the SMART$_{\mathrm{H}}$ heuristic to further scale up the computation for SMART problem instances. As we will see in the following section, SMART$_{\mathrm{O}}$ fails to scale beyond 4 targets in our computational experiments. Hence, we introduce SMART$_{\mathrm{H}}$ which follows the same branch-and-price procedure discussed before but uses a novel heuristic slave formulation.

In essence, SMART$_{\mathrm{H}}$ is built on two intuitions related to coordination. The first intuition is that joint patrols can be computed by considering individual patrols iteratively, by using *greedy policy optimization* between iterations to reflect the additive benefit of joint activities. The second intuition is that each defender resource would like to visit as many targets as possible, and visiting targets in accordance with an *ordering* based on a solution of the Traveling Salesman Problem is likely to extract maximal benefit out of the resource while still accounting for the spatio-temporal constraints needed for coordination. As a result, the SMART$_{\mathrm{H}}$ slave only needs to solve a set of *linear programs* (as opposed to solving a MILP in SMART$_{\mathrm{O}}$'s slave).

5.1 Iterative Reward Modification

The slave in SMART$_{\mathrm{H}}$ computes the joint patrol $\mathbf{P}_j$ of the defender by iteratively and greedily building up individual patrols X_r for each defender resource r. The additional benefit of joint activities is considered by appropriately shaping the rewards for each resource based on the patrols of other resources. Greedy policy optimization has been used in other reinforcement learning contexts [37]; here we leverage this idea for coordination among multiple resources. This greedy approach allows SMART$_{\mathrm{H}}$ to handle heterogeneous defender resources with each iteration solving for a different resource r.

Algorithm 1 SMART$_{\mathrm{H}}$ Greedy Slave

1: Input: $\mathbf{y}$, $\boldsymbol{\mathcal{G}}$
2: Initialize $\mathbf{P}_j$, $\boldsymbol{\mu}$
3: **for all** $r_i \in R$ **do**
4: $\quad X_i \leftarrow \text{SolveSinglePatrol}(\mathbf{y}, \boldsymbol{\mu}, \mathcal{G}_r)$
5: $\quad \mathbf{P}_j \leftarrow \mathbf{P}_j \cup X_i$
6: $\quad \boldsymbol{\mu} \leftarrow \text{ComputeCostCoef}(\mathbf{P}_j, \mathcal{G}_r)$
7: $\boldsymbol{\omega}(\mathbf{P}_j) \leftarrow \text{ConvertToColumn}(\mathbf{P}_j)$
8: return $\mathbf{P}_j, \boldsymbol{\omega}(\mathbf{P}_j)$

SMART$_{\mathrm{H}}$ uses a greedy algorithm, as outlined in Algorithm 1. This algorithm takes the coefficients y_t (refer Equation (11)) as input and builds $\mathbf{P}_j$ iteratively in Lines 3–6. Line 4 computes the best individual patrol X_r for the defender resource r (described in Section 5.2). X_r is then merged with the rest of the defender's pure strategy $\mathbf{P}_j$ (in Line 5). Line 6 computes $\boldsymbol{\mu}$, the potential effectiveness contribution from one resource to another given the current

pure strategy $\mathbf{P}_j$. This is computed over each edge $e(\mathtt{u},\mathtt{v},\alpha)$ in the transition graph, and measures the added benefit to the defender *if the defender resource was to travel from* $\mathtt{u}.t$ *to* $\mathtt{v}.t$ *at time* $\mathtt{u}.\gamma$ *performing activity* $e.\alpha$ *at target* $\mathtt{v}.t$. These values of $\boldsymbol{\mu}$ are used in the next iteration when computing an individual patrol for the next defender resource.

To understand how close the solution of the greedy algorithm is to the optimal solution, we use some insights from [28], which states that greedy maximization of a non-negative submodular function achieves a constant-factor approximation. Recall that the objective of the slave problem is to find a pure strategy $\mathbf{P}_j$ that minimizes the reduced cost $\bar{c}_j$ (see Equation 11). This is equivalent to maximizing (since z in Equation 11 is a constant):

$$F(\mathbf{P}_j) = -\sum_{t \in T} \omega_t(\mathbf{P}_j) \cdot y_t \tag{25}$$

The duals $\mathbf{y}$ from the master are always negative in this formulation making $F(\mathbf{P}_j)$ non-negative. $\omega_t(\mathbf{P}_j)$ is the effectiveness of pure strategy $\mathbf{P}_j$ at target t as defined in (3).

If $F(\mathbf{P}_j)$ is submodular, and if $\mathbf{P}_*$ is the optimal defender pure strategy, then, as shown by Nemhauser et al. [28] the solution $\mathbf{P}_j$ of the greedy algorithm satisfies

$$F(\mathbf{P}_j) \geq \frac{1}{2} F(\mathbf{P}_*) \tag{26}$$

For the special case where the time window, W, is greater than or equal to the maximum patrol time[5], Γ, we show that $F(\mathbf{P}_j)$ is submodular. $F(\mathbf{P}_j)$ is submodular if P_1 and P_2 are two sets of routes where $P_1 \subseteq P_2$ and $F(P_1 \cup \{X\}) - F(P_1) \geq F(P_2 \cup \{X\}) - F(P_2)$.

Theorem 1 *$F(\mathbf{P}_j)$ is submodular in $\mathbf{P}_j$ if $W \geq \Gamma$ and $\mathtt{eff}()$ is submodular.*

Proof Since $W \geq \Gamma$ and $\omega_t(\mathbf{P}_j) = \mathtt{eff}(S_{\mathbf{P}_j})$, where $S_{\mathbf{P}_j}$ is the set of activities of $\mathbf{P}_j$ on target t. To prove that $F(\mathbf{P}_j)$ is submodular, it suffices to show that $\omega_t(\mathbf{P}_j)$ is submodular because $F(\mathbf{P}_j)$ is defined as a non-negative linear combination of $\omega_t(\mathbf{P}_j)$. Considering Equation (2):

$$\mathtt{eff}(S_{P_1} \cup \alpha_X) - \mathtt{eff}(S_{P_1}) \geq \mathtt{eff}(S_{P_2} \cup \alpha_X) - \mathtt{eff}(S_{P_2})$$

we can write $\omega_t(P_1 \cup X) - \omega_t(P_1) \geq \omega_t(P_2 \cup X) - \omega_t(P_2), P_1 \subseteq P_2$. Thus, $\omega_t(\mathbf{P}_j)$ is submodular when the time window is greater than or equal to the maximum patrol time.

In real life situations, W may be less than Γ. We show that even in this situation, $F(\mathbf{P}_j)$ is submodular for 2 resources.

Theorem 2 *$F(\mathbf{P}_j)$ is submodular in $\mathbf{P}_j$ for two resources if $\mathtt{eff}()$ is submodular.*

[5] $W \geq \Gamma$ implies that two resources present at the same target at anytime during the patrol are considered to conduct a joint activity.

Proof We prove that $F(P_1 \cup \{X\}) - F(P_1) \geq F(P_2 \cup \{X\}) - F(P_2)$ where $P_1 = \{\emptyset\}$ and P_2 contains a single patrol $\{X_2\}$. To do this, we show that $\omega_t(\{X\}) \geq \omega_t(\{X_2, X\}) - \omega_t(\{X_2\})$, for each target t, based on the submodularity property of $\texttt{eff}()$ in (2). We proceed in two steps. First, we show that:

$$w_t(\{X\}) \geq w_t(\{X_2, X\}) - w_t(\{X_2\}) \tag{27}$$

We use case-reasoning. If X_2 and X are in the same window then the same argument as Theorem 1 applies. Hence, we only need to demonstrate the equation for the case where X_2 and X are not in the same window. We have $w_t(\{X_2, X\}) = \max(\texttt{eff}(X_2), \texttt{eff}(X))$, then:

$$w_t(\{X_2, X\}) - w_t(\{X_2\}) = \max(0, \texttt{eff}(X) - \texttt{eff}(X_2)) \tag{28}$$

$$\Leftrightarrow w_t(\{X_2, X\}) - w_t(\{X_2\}) \leq \texttt{eff}(X) = w_t(X) \tag{29}$$

Second, we show that Equation 27 is equivalent to $w_t(P_1 \cup \{X\}) - w_t(P_1) \geq w_t(P_2 \cup \{X\}) - w_t(P_2)$:

$$w_t(\{X\}) \geq w_t(\{X_2, X\}) - w_t(\{X_2\}) \tag{30}$$

$$\Leftrightarrow w_t(\{\emptyset\} \cup \{X\}) - w_t(\{\emptyset\}) \geq w_t(\{X_2, X\}) - w_t(\{X_2\}) \tag{31}$$

$$\Leftrightarrow w_t(P_1 \cup \{X\}) - w_t(P_1) \geq w_t(P_2 \cup \{X\}) - w_t(P_2) \tag{32}$$

This shows that $\omega_t(\mathbf{P}_j)$ is submodular. As a consequence, $F(\mathbf{P}_j)$ is also submodular because it is a non-negative linear combination of $\omega_t(\mathbf{P}_j)$.

Qualifying this result for $W < \Gamma$ for 2 resources *is important since this setup is used most frequently in the real world, e.g., the US Coast Guard.* For three or more resources, we can artificially construct counter-examples that break submodularity. However, given actual domain geometries, time windows, and operational rules, submodularity may hold even for larger number of resources – e.g., Theorem 1 shows that relaxing the time window may lead to such submodularity. Characterizing these spaces is a topic left for future work.

5.2 TSP Ordering with Transition Graph

To achieve the approximation bound in Equation (26), we need to optimally compute an individual patrol X_r for the defender resource r in line 5 of Algorithm 1. This can be solved by an MILP of similar form to the slave MILP (Equations (13)-(17)), but for a single patrol. The resulting MILP for a single patrol has less variables than the MILP for all patrols, however this still fails to scale up beyond 6 targets (Section 6).

Instead, we present a heuristic approach that achieves better scale-up by exploiting the spatial structure of the domain, and is provably optimal in some specific cases. Our approach is based on the following restricted version of the problem: we define an *ordering* of the targets and restrict the sequence of target

visits to be increasing in this order. We construct the *ordered transition graph* in the same way as described in Section However, now, an edge from node u to v is added *only if* target u.t appears before target v.t in the ordering. If there does not exist a direct edge from u to v, an edge is added between these nodes such that $\tau_{u.t,v.t}$ is equal to the shortest path from $u.t$ to $v.t$. Traversing along this edge does not impact the effectiveness of the intermediate targets. Instead of computing the maximum effectiveness of the multiple edges per target, each target is only visited once per patrol in the ordered transition graph. Since each target in the patrol is counted only once, the max expressions in (13), (15), and (16) can be replaced with linear expressions. The resulting problem is equivalent to a min-cost flow, which has integer extreme points that allow us to drop the integrality constraint (17), since a feasible solution of the resulting LP is guaranteed to be an integer flow. Hence, these LPs are easier to solve than the above MILPs, both in theory as well as in our experiments.

Fixing an ordering will exclude certain patrols. Therefore, we would like an ordering such that the resulting patrol, which corresponds to a subsequence of the ordering, will still be a sensible way to visit targets compared to patrols with alternative orderings. To that end, $\textsc{Smart}_{\text{H}}$ uses an ordering based on the solution of the traveling salesman problem (TSP). Given an input graph of all targets, $G = (T, E)$, the orderings are generated using a nearest neighbour algorithm (as discussed by Gutin et al. [16]) which determines the order to which the targets are visited in the patrol. Despite using an approximate algorithm, we are still able show that under certain conditions, the TSP ordering can yield an optimal solution of the single-patrol problem. When such conditions do not hold, it is likely that different orderings, based on more sophisticated TSP algorithms could result in a better solution. This could be a very interesting empirical challenge. However, we decided to focus our analysis on different aspects of the problem because in Section 6 we compare the performance of both $\textsc{Smart}_{\text{O}}$ and $\textsc{Smart}_{\text{H}}$ and show that the TSP ordering, despite being generated by an approximate algorithm, generates solutions that are very close to the optimal ones.

We look at a tree structure because various domains in the real world can be represented as a graph similar to a tree. For example, train lines can be represented as a line where edges connect each station and each station is connected to some other nodes representing the different levels of the station (e.g., platform, mezzanine or parking level). Similarly, ports can be imagined as a minimum spanning tree connecting all the locations within a port.

Theorem 3 *Suppose the input graph G is a tree, and the time window for joint effectiveness is greater than or equal to the maximum patrol time. Then* $\textsc{Smart}_{\text{H}}$ *computes a patrol for a single unit that optimizes the objective for the single unit problem in Algorithm 1.*

Proof We first observe that the optimal TSP tour of G visits each edge of the tree exactly twice. The TSP tour corresponds to a complete preorder traversal of the tree.

SMART_H outputs a patrol P on a subset of targets T_P, corresponding to a subsequence of the TSP ordering. We show that this patrol is a TSP tour of the corresponding subgraph on T_p, denoted G_p. There are two cases:

1. If G_P is a connected subtree of G then P is a preorder traversal of that subtree, and therefore is a TSP tour of the subtree.
2. If G_P is not connected, for each two targets in different connected components of G_P there is a unique path in the original tree graph G that connects the two. By adding nodes on these paths to G_P, we recover a connected subtree G'_P, an optimal TSP tour on which is also optimal for G_P. Then since P is a preorder traversal of the subtree G'_P, it is a TSP tour of G_P.

Consider a patrol P' on T_P that does not follow the TSP ordering. Let P be the patrol we get by reordering targets of P' so that they are increasing in the TSP ordering. Since P is a TSP tour of G_P, if P' finishes within the time limit then P also does. Furthermore, since the time window for joint effectiveness is large, joint activities in P' will also be joint activities in P, and thus P achieves the same slave objective as P'. Therefore, we never lose optimality by considering only patrols that follow the TSP order.

When the graph is a tree but the time window is smaller than the patrol time limit, the algorithm is not guaranteed to be optimal. However, as we show in our experiments, SMART_H generates optimal or near-optimal solutions for SMART problem instances.

6 Experimental Results

The section presents our simulations and our real-world experimental results. In Section 6.1, we extensively evaluate the performance of SMART_H and SMART_O in solving instances of SMART. We show the impact of the different components of our approach in terms of runtime and with respect to the difference in solution quality of the optimal versus heuristic algorithm. In Section 6.2, we describe our field experiment. This experiment constitutes the first real world head-to-head comparison between game-theoretic and human generated schedules. This comparison covers the effort to generate the schedules, the evaluation of security within a train line by security experts and the coordination between different deployed resources. This evaluation constitutes a contribution to evaluation not only of joint coordinated activities in the real world but also to the general evaluation of SSG-applications in the real world.

6.1 Simulations

In our simulations, we are not able to compare with previous algorithms [12, 17, 30, 41] due to the inability of these algorithms to scale up to the combinatorics

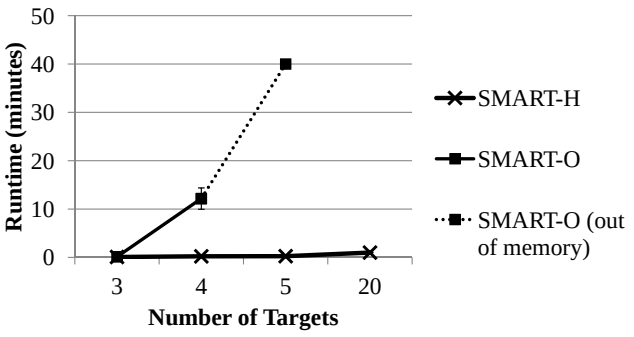


Fig. 4 Runtime of $\textsc{Smart}_{\mathrm{H}}$ vs $\textsc{Smart}_{\mathrm{O}}$

unleashed by joint activities. Hence, we compare different versions of $\textsc{Smart}_{\mathrm{H}}$ and $\textsc{Smart}_{\mathrm{O}}$.

In the results below, in each experiment, each data point shown is averaged over 100 game instances, generated with random payoffs in the range [-10,10] and two defender resources unless otherwise noted. All the figures contains error bounds indicating the standard error of the mean. Given, the large number of instances, in some cases the bars are not shown, since they are too small to be seen. All the results are tested for statistical significance using a Student t-test (p = 0.05). All experiments were run on graphs constructed beginning with a tree that spans the targets and then adding 10 random edges between nodes to create some random cycles in the graph. The root of the tree corresponded to the home base, t_b. The idea is to simulate the typical topology of a transportation hub or network, e.g., the areas of a port and the stations of a train line as in the port of Los Angeles or the Metro train line in Los Angeles. The time window was 30 minutes with 3 possible activities for the defender, taking 0, 5, or 15 minutes. In most of the experiments described below, the transition graph required to build the slave problem (see Section 4.1) was built using a time discretization of 5 minutes, i.e., any two adjacent nodes of the graph $u_1 = (t_1, \gamma_1)$ and $u_2 = (t_2, \gamma_2)$ were defined such that $\gamma_2 = \gamma_1 + 5$. In contrast, in the experiments where we compared $\textsc{Smart}_{\mathrm{O}}$ and $\textsc{Smart}_{\mathrm{H}}$, we used a larger time discretization of 15 minutes. This discretization was used for comparison purposes, i.e., $\textsc{Smart}_{\mathrm{O}}$ would not run with the large transition graph produced as a result of a larger discretization. In such a setting, one activity had a duration of 0 minutes while the remaining two had a duration of 15 minutes. All experiments were run on a machine with a Dual core 2.0 GHz processor and 4 GB of RAM. We present our results in the remainder of this section.

6.1.1 $\textsc{Smart}_{\mathrm{H}}$ *vs.* $\textsc{Smart}_{\mathrm{O}}$*: Runtime*

In this experiment, we compare $\textsc{Smart}_{\mathrm{H}}$ and $\textsc{Smart}_{\mathrm{O}}$ in terms of runtime to solve different Smart problem instances. Each instance is generated consid-

	SMART$_H$	SMART$_O$
3 targets	1.298	1.298
4 targets	-0.7135	-0.6930

Table 4 Solution Quality of SMART$_H$ vs. SMART$_O$

ering an increasing number of targets. The results are shown in Figure 4. In the figure, the number of target is shown on the x-axis while the runtime is shown on the y-axis. We can see that SMART$_O$ is not able to scale to problem instances with more than 4 targets. In contrast, SMART$_H$ takes seconds to scale up to instances of 20 targets. As shown in the figure, when the number of targets increases to 5, SMART$_O$ runs out of memory and takes at least 40 minutes to run before giving an out-of-memory error. In contrast, SMART$_H$ takes less than a minute to solve problems consisting of up to 20 targets. This shows that SMART$_H$ dramatically improves both runtime computation and memory consumption compared to SMART$_O$.

6.1.2 SMART$_H$ *vs.* SMART$_O$*: Solution Quality*

In this experiment, we compare the solution quality, or defender expected utility, of SMART$_H$ versus SMART$_O$ while varying the number of targets. The idea is to understand the quality of the solutions recovered by the more "practical" algorithm, SMART$_H$, in comparison with the optimal solutions generated by SMART$_O$. We generate SMART instances by increasing the number of targets (i.e., up to the maximum number of targets that can be solved by SMART$_O$).

The results are shown in Table 4. The results for 3 targets indicate that SMART$_H$ provides, in average, the same solution quality as SMART$_O$ for all game instances. By increasing the number of targets to 4, we can see that the average solution quality for SMART$_H$ becomes 0.0205 lower than SMART$_O$ with there being only *one* game instance where SMART$_H$ computed a lower defender expected utility than SMART$_O$. In this one instance, the final result of SMART$_H$ was -0.61685 while the final result for SMART$_O$ was 0.0625, or a difference of 0.679. Hence, given our problem settings, SMART$_H$ computes a solution which is, on average, very close to SMART$_O$ on all those instances that can actually be solved by SMART$_O$.

6.1.3 SMART$_H$*: Scalability using TSP ordering*

In this experiment, we measure the ability of SMART$_H$ to scale up to large problem instances while using the TSP ordering procedure described in Section 5.2 and while not using any ordering procedure. The idea is to measure the benefits that ordering the nodes will do to the SMART$_H$ algorithm in terms of its scalability. In the experiment, we consider one single defender resource.

Figure 5 shows the results of our experiment. In the figure, the x-axis is the number of targets and the y-axis is the runtime. As shown in the figure,

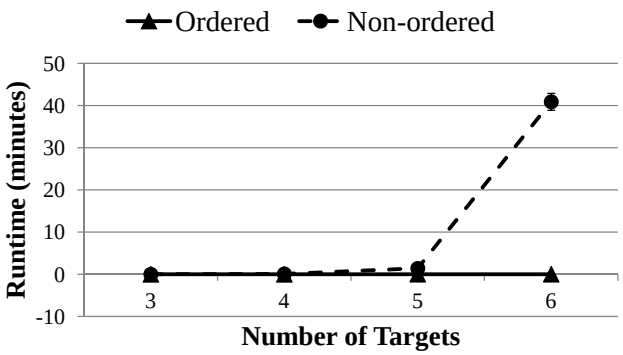
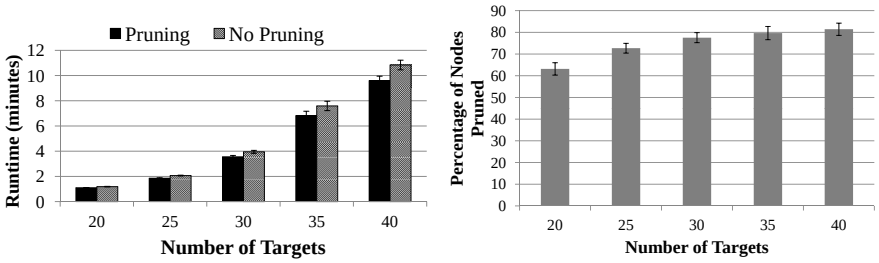


Fig. 5 Using ordered nodes



(a) pruning vs. non-pruning (b) pruning vs. number of targets

Fig. 6 SMART$_{\mathrm{H}}$: benefits of pruning nodes

by using the MILP and not ordering the nodes, or allowing the defender to visit any node at any order in the path, the runtime for 6 targets is over 40 minutes. For 7 targets, SMART$_{\mathrm{H}}$ not using node ordering runs out of memory. The impact of the TSP ordering heuristic is then significant: SMART$_{\mathrm{H}}$ takes seconds for solving problems with up to 7 targets. In fact, as we will see in the next experiments, this heuristic allows the algorithm to scale up easily to problems with up to 40 targets.

6.1.4 SMART$_{\mathrm{H}}$*: Pruning*

In this experiment, we analyze the benefits of generating tight upper bounds to prune the branch-and-price tree. The idea is to show in detail, why SMART$_{\mathrm{H}}$ can scale up to large problem instances, while preserving memory.

Figure 6(a) shows the runtime (in minutes) required by the algorithm to solve SMART problem instances either pruning nodes (using ORIGAMIP) or not. The x-axis shows the number of targets and the y-axis shows the runtime. As shown in the figure, the amount of time saved by pruning nodes increases with the number of targets. We ran a student t-test (p=0.05) which confirmed the statistical significance of these results. In larger problem instances, the branch-and-price tree is bigger. Thus, a larger number of nodes will be pruned.

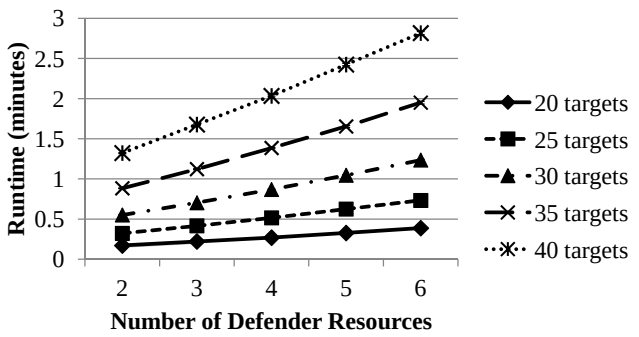


Fig. 7 Runtime with multiple defender coordinating

This will have an impact on runtime: by using OrigamiP, the algorithm will require less time to solve bigger problem instances compared to simply solving all the leaf-nodes of the tree (i.e., no pruning).

Figure 6(b) shows the percentage of nodes pruned by Smart_H considering Smart problem instances of different size. In the figure, the x-axis shows the number of targets while the y-axis shows the percentage of the nodes in the branch-and-bound tree that were pruned by Smart_H. The figure confirms the intuition behind the results in Figure 6(a), Smart_H will prune more nodes on larger problem instances, e.g., up to 70% of the nodes for instances of 40 targets.

6.1.5 Smart_H*: n-ary Joint Activities*

In this experiment, we measure the ability of Smart_H to solve large problem instances of up to 40 targets and large joint activity spaces of up to 10 activities at the same time. In Section 3, we defined the coverage effectiveness of a pure strategy in Equation 3. We used a max operator which, however, accounts only for the best single activity and joint couple of activities to calculate the coverage effectiveness. In this experiment, we modify Smart_H to handle n-ary combinations of resources efficiently by re-defining the underlying utility function as an additive joint activity function (capped at one). Smart_H's iterative modification of reward then efficiently computes μ based on this new utility function based on the n-ary interaction (line 6 of Algorithm 1).

Figures 7 depicts the results of the experiment. In the figure, the x-axis is the number of defender resources. Each of such numbers is also associated with a specific arity of joint actions (e.g., 3 resources with ternary interactions) and the y-axis is the runtime. As we can see, Smart_H can efficiently handle n-ary interactions since, the runtime never exceeds 3 minutes, even for instances consisting of 40 targets and 10 defender resources.

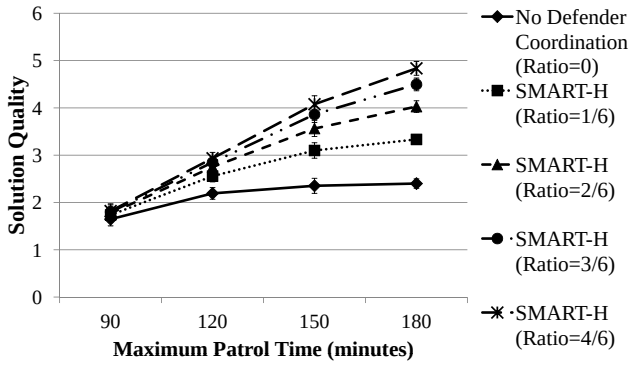


Fig. 8 Solution quality of SMART$_{\mathrm{H}}$ versus algorithm with no joint activities

6.1.6 SMART$_{\mathrm{H}}$*: Effectiveness of Joint Activities*

In this experiment, we compare the quality of the solutions obtained by running SMART$_{\mathrm{H}}$ on SMART problem instances where the resources have activities with varying levels of joint effectiveness. The idea is to measure how SMART$_{\mathrm{H}}$ will allocate resources when the effectiveness of their joint activities increases. We calculate the level of effectiveness as a ratio between the highest joint effectiveness value and the highest single effectiveness value, considering the best activity α_{max}:

$$\frac{\texttt{eff}(\alpha_{max}, \alpha_{max}) - \texttt{eff}(\alpha_{max})}{\texttt{eff}(\alpha_{max})} \tag{33}$$

For all test scenarios, $\texttt{eff}(\alpha_{max})$ is held constant to a value of 0.6, while varying the values of $\texttt{eff}(\alpha_{max}, \alpha_{max})$. For example, when $\texttt{eff}(\alpha_{max}, \alpha_{max}) = 0.8$, the subsequent ratio would be: $(0.8 - 0.6)/0.6 = 0.2/0.6 = 2/6$. The individual and joint effectiveness, here, are calculated as discussed in Section 3. They receive the maximum effectiveness and any additional resource visiting a target within the pre-defined time window will have no additional benefit (see Equation 3).

The results are shown in Figure 8. In the figure, the y-axis shows the solution quality and the x-axis denotes the maximum patrol time. Considering the experiment settings discussed at the beginning of this section, we can see that when the patrol time is increased, a simple strategy with no defender coordination (no benefit to joint activities) provides very little benefit to the solution quality while the improvement due to the coordination of multiple defender resources can almost double the solution quality. In more detail, taking into account joint activities between multiple defenders provides a solution quality that is double (when the ratio is 4/6) than that of an approach that does not handle defender coordination, i.e., when the ratio is 0.

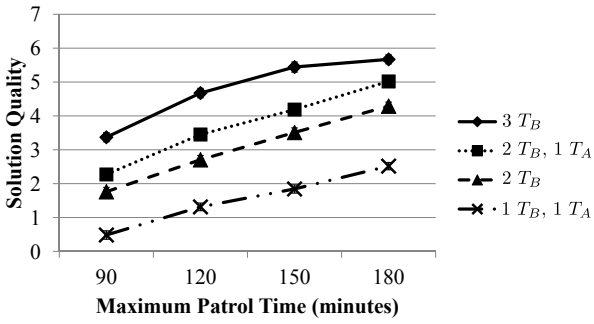


Fig. 9 Heterogeneous defender resources: type A (T_A) and type B (T_B) for 30 targets.

6.1.7 Solution Quality against Hetergeneous Resources

In this experiment, we compare the quality of the solutions obtained by different versions of SMART$_{\text{H}}$ considering an increasing number of heterogeneous resources with different abilities. The idea is to measure the impact that different hetergeneous resources will have on the quality of the solutions recovered by SMART$_{\text{H}}$. We consider two type of resources: type A (T_A) and type B (T_B). Resources of type A are different than resources of type B in the following ways: (1) shorter transit times; (2) shorter patrol time; (3) lower effectiveness values.

Figure 9 shows the results considering both type A and type B. The figure shows the solution quality (i.e. the expected utility for the defender) obtained varying the maximum patrol time, the number of resources, their type and considering 30 targets. As we can see, increasing the number of resources lead to a higher solution quality. More specifically, we can see that by considering three resources instead of two, the expected utility increases from 1.77 to 3.46 (i.e., 50%) considering 90 minute patrols and from 4.31 to 5.69 (i.e., 25%) considering 180 minute patrols. Hence, as the number of resources increases, SMART$_{\text{H}}$ is able to allocate them effectively by exploiting their abilities to improve the overall solution quality.

6.2 Real-world Experiment

We present in what follows a real-world experiment whereby we compared the game-theoretic allocation of resources computed using SMART$_{\text{H}}$ against a manual allocation, the standard methodology adopted by several security agencies. Security agencies refer to this type of experiment as a mass transit full scale exercise (FSE). It was important to perform this exercise in the real world rather than purely on paper with imaginary units, in order to ensure that the schedules generated by SMART$_{\text{H}}$ be compared to manual schedules when getting executed in the real-world under real-world constraints with real heterogeneous resource types.

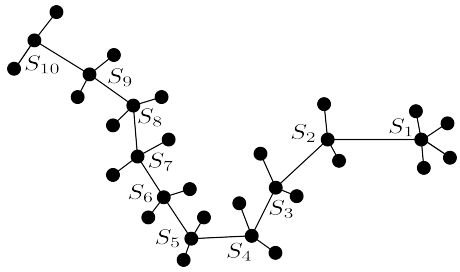


Fig. 10 The 10 stations of the FSE

A FSE is a training exercise where multiple security agencies analyze the way their resources cooperate to secure a specific area while simulating a critical scenario. This scenario typically describes a "high level" threat, e.g., intelligence reports confirming that a terrorist attack might take place in the Los Angeles Metro System. The FSE consists of simulating the response to this threat, i.e., increasing the number of resources patrolling a train line on a daily basis to improve the quality of the security. Nonetheless, in most real-world settings, the number of resources deployed by the agencies is not sufficient to cover all the different locations within a train line. For this reason, an intelligent and unpredictable allocation of security resources, which leverages their ability to work individually and cooperate together, is crucial to achieve a more effective security.

All the above reasons make an FSE a very promising test-bed to run our comparison between $\textsc{Smart}_{\mathrm{H}}$ ($\textsc{Smart}_{\mathrm{O}}$ would not scale to the dimensions of the FSE, as discussed in Section 6.1) and manual schedules. In addition, it would allow us to collect real-world data which we could use to analyze and improve the current algorithm.

6.2.1 Organization of the FSE

The FSE consisted of patrolling 10 stations of one train line of the LA Metro system for 12 hours. Each station on the train line is composed of three levels (street level, platform level and mezzanine) except station 1 which is composed of 5 levels (2 more platform levels). Figure 10 shows a graph illustrating the 10 stations.

The exercise involved multiple security agencies, each participating with a number of resources. Overall, 80 security personnel were involved. These resources were divided into 14 teams, each with different abilities. The resources deployed in the FSE are described in Table 5.

The exercise was divided into 3 different "sorties", each consisting of three hours of patrolling and one hour of debriefing. Human-generated schedules were used during the first sortie while $\textsc{Smart}_{\mathrm{H}}$ schedules were used during the second and the third sorties. To visualize the schedules generated using $\textsc{Smart}_{\mathrm{H}}$, each team was given a android smartphone app to visualize the game-theoretic schedule (see Figure 11).

Team Description		
Acronym	Name	Deployed Teams
T Teams	High Visibility Uniformed Patrol Teams	T_{16} T_{27} T_{38} T_{49} T_{510} T_{11}
HVWT Teams	High Visibility Weapon Teams	$HVWT_{12}$ $HVWT_{34}$
VIPR Team	Visible Intermodal Interdiction Team	VIPR
CRM Teams	Crisis Response Motors	CRM_1 CRM_2 CRM_3
EK9 Teams	Explosive K9 (canine)	$EK9_1$ $EK9_2$

Table 5 Teams deployed during the FSE

Fig. 11 The smartphone application used to visualize the schedule of the CRM team

The first two sorties were used to run the head-to-head comparison. Hence, the sorties were run under the same settings: the same number of officers had to cover the 10 stations for a cumulative time of 450 minutes. The two sorties were run during off-peak times (9h00 to 12h00 and 13h00 to 16h00, respectively), hence the type and the number of riders of the train lines could be considered to be, approximately, the same.

The purpose of Sortie 3 was to test whether the officers were capable of following SMART_H schedules for a longer period (900 minutes instead of 450)

and during peak time, i.e., when traffic and the number of people riding the trains increases. We found out that the officers were actually able to follow the schedules. Thus, since the purpose of this Sortie was unrelated to our comparison, we will focus on Sorties 1 and 2 in the remainder of this section.

6.2.2 The Schedule Generation Process

Each type of schedule was generated as follows:
Smart$_{\mathbf{H}}$ schedules: The schedules were generated by (i) instantiating a SMART problem instance using the specifics of the FSE discussed earlier; (ii) solving this problem instance using SMART$_{\mathrm{H}}$ and (iii) sampling a pure strategy to generate the patrol schedule for each of the different resources involved.

To define a problem instance we had to define three different sets of features of the problem. The first sets of features are the graphs for each resource type. By using the graph presented in Figure 10 as a baseline, we defined the graph for each resource as follows:

- T teams, HVWT teams and VIPR teams move using the trains and patrol all the different levels of a station. Hence, each resource type was assigned the graph in Figure 10.
- CRM teams move using their bikes and patrol only one level (the street / parking level) of the stations. Their graph is then a line connecting ten nodes, each representing the street level of a station.
- EK9 teams move using a car and patrol all levels. Their graph is defined as in Figure 10 but edges connect the nodes representing the street level instead of the platform level.

The second sets of features are the payoffs of the game.[6] We defined the payoffs for each target (32 in total) in discussions with security experts from the Los Angeles County Sheriff's Department (LASD). Each set of payoffs for each station was based on the number of people using the station every day and by the economic impact that losing this station would have on the city. The different levels of a single station had slightly different payoffs which were based on the number of persons present at each specific level of the station every weekday.

The payoffs were defined so that the game was zero-sum, i.e., given a target t, we defined $U_d^c(t) = U_a^c(t) = 0$ and $U_d^u(t) = -U_a^u(t) = v$ where $v \in [0, 10]$. This choice of payoffs is reasonable for this setting because the key here is the importance of stations, and the defender will lose exactly as much as the attacker will gain if his attack is successful.

The third features are the activities for each resource type and the corresponding single and joint effectiveness. Each team could perform a series of two types of activities:

[6] We are not able to reveal the value of these payoffs due to an agreement with the Los Angeles County Sheriff Department (LASD).

Observe: This action consisted of observing a specific target for 15 minutes. The officers had to explore the location (e.g., street level or platform) with great accuracy to ensure that it was secure.

GoTo: This action consisted of moving from one target to another (e.g., by using a car, by riding a train or using the stairs to move between targets at the same station). The duration of this action was estimated based on the train schedules, the traffic at the time of the day and the average time to move from one level of a station to another.

Given these activities, the single and joint effectiveness parameters are summarized in Table 6. In the table, the joint activities are represented as a vector of two values, one for each joint activity between the action in the row of the table and an observe action and a goto action, respectively. As we can see, *all* GoTo actions are given a 0 effectiveness, since moving from one station to another (i.e., riding the trains or taking the car) will not have any effect on the security of the stations. Most teams are assigned the same positive individual effectiveness of 0.7, except the VIPR team which has a greater individual effectiveness because it is composed of officers from multiple agencies carrying heavy weapons. VIPR teams, T-teams and HVWT teams typically work alone. Hence, to define their effectiveness values, their individual effectiveness is positive while their joint effectiveness is null (any joint effectiveness value below 0.7 would induce the same type of behavior, but we chose 0 since it is a clear indicator of the type of behavior that we want to obtain). The CRM teams are assigned a joint effectiveness greater then their individual effectiveness because they can perform all type of activites, but, typically, they prefer joint over individual activities. In contrast, EK9 teams typically work only in cooperation with other teams, therefore they are assigned a null individual effectiveness and a positive joint effectiveness of 0.75.[7]

Next, we defined the time window for a joint action to be effective as a 10 minutes interval. This value was chosen considering the time required by teams to move from a station to another, from a level to another and by discussion with the security agencies involved in the exercise. Finally, we decided the discretization of the transition graph to correspond to the shortest duration between the durations of all the actions available to the different resources (i.e., the goto action to move between two levels).

After defining the SMART problem instance, we solved it using SMART_{H}. We run the algorithm considering 14 resources (the teams defined in Table 5), and 32 targets (5 levels for station S_1 and 3 levels for the 9 other stations). To reach the cumulative time of 450 minutes, as required by the specifics of Sortie 2, we defined the patrol time of each resource such that a total close to 450 minutes could be obtained. The mixed strategy provided by the algorithm

[7] Whereas these estimates of individual and joint effectiveness could potentially be slightly altered, the purpose of our exercise was comparison with human schedulers. Since the comparison was ultimately conducted by security experts who evaluated the game-theoretic schedule to be superior to human-generated one, we may infer that the effectiveness values we obtained for the individual and joint activities in our SMART model for FSE were reasonable.

Team	Action	Individual eff	Joint eff
T-teams	Observe	0.7	[0.0, 0.0]
	GoTo	0	[0.0,0.0]
HVWT teams	Observe	0.7	[0.0, 0.0]
	GoTo	0	[0.0,0.0]
VIPR team	Observe	0.8	[0.0,0.0]
	GoTo	0	[0.0,0.0]
CRM teams	Observe	0.7	[0.75,0.0]
	GoTo	0	[0.0,0.0]
EK9 teams	Observe	0	[0.75,0.0]
	GoTo	0	[0.0,0.0]

Table 6 Individual and joint activities

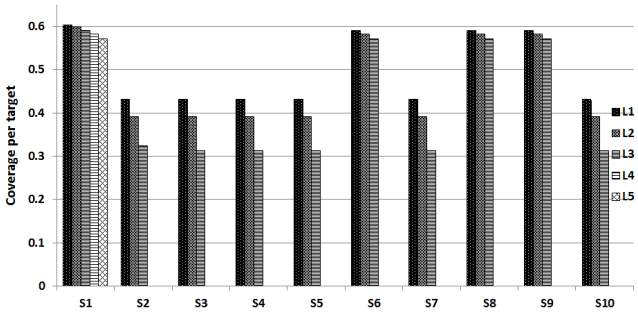


Fig. 12 The mixed strategy for the FSE

is shown in Figure 12. The figure shows the coverage c_t for each target t as defined by Equation 8 in the master in Section 4. In the figure, the levels of the stations with a higher payoff are assigned a higher coverage.

As a final step, the mixed strategy is sampled to generate a pure strategy. This pure strategy contains a schedule for each resource. It is shown in Table 11 in Appendix A.

Manual Schedules: The schedules were by human expert schedulers of the LASD. They were generated using a two-step process. First, each station was assigned a coverage duration of 45 minutes (i.e., $\frac{1}{10}^{th}$ of the time). The idea was to have the officers perform *three observe* actions at each station. Second, the human expert schedulers assigned teams to each station so that each station was covered for exactly 45 minutes. Joint team activities were used 6 times in six different stations. The resulting allocation is shown in Table 10 in the Appendix A. This simple two-step process was adopted to avoid the cognitive burden involved with leveraging the effectiveness of each team to cover the different stations individually or while coordinating with other teams. Despite its simplicity, this process was difficult for the human expert schedulers. It involved several discussions and required one entire day of work.

Having defined the two allocations, we can now analyze the results that we obtained.

	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}
Manual	3	3	3	2	3	2	2	2	2	2
SMART_H	2	2	3	3	2	2	2	3	3	2

Table 7 Count of Individual Activities

	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}
Manual	0	0	0	1	0	1	1	1	1	1
SMART_H	1	0	0	0	0	2	0	1	1	1

Table 8 Count of Joint Activities

6.3 Results

We first analyze the type of schedules generated as a result of using either SMART_H or manual scheduling. Then, we evaluate the results obtained by deploying the schedules during Sorties 1 and 2 and measuring their performance in the real world. The key is that if SMART_H were to perform equivalently to human schedulers, then it would indicate that we could save precious time so security experts could focus on maintaining security rather than on generating schedules.

6.3.1 Schedules Comparison

The allocation of resources generated by Manual scheduling and SMART_H are shown in Tables 10 and 11 in Appendix A. The numbers of individual and joint activities for both schedules are shown in Tables 7 and 8. In both tables we can see that the number of individual (IA) and joint (JA) activities for both approaches are the same (IA: both 24; JA: both 6). All the joint activities in the SMART_H schedules are performed by CRM and EK9 teams, i.e., the teams with a positive joint effectiveness. This is similar to the behavior of the manual generated schedules, where joint activities are mostly performed by EK 9 and CRM teams (once by the VIPR team). The remaining individual activities are performed by the T team, the HVWT team and the VIPR team.

There are two important differences between the two types of schedules. The first is that the game-theoretic scheduler sent the most effective VIPR team to the most important stations – because its individual effectiveness is greater than the effectiveness of other teams. This was not seen in the human schedule. The second difference between the two types of schedules is that the schedules generated using SMART_H assign the different teams to cover all the different levels of the different stations, whereas manual schedules do not specify such levels. The reason for this is that human schedulers were not able to reach this level of detail and thus they preferred to leave the decision of which level to patrol to the teams once they were deployed. In addition to accuracy, in SMART_H, the human effort was confined to providing the individual and joint effectiveness of available teams, and then the schedules were

Q_1	"The security scheduling system (SSS) makes the station safer."
Q_2	"The SSS is efficient."
Q_3	"The SSS is an effective deterrent for adversaries."
Q_4	"The SSS results in some areas being patrolled more than needed."
Q_5	"The SSS decreases the number of attempts to infiltrate train stations."
Q_6	"The SSS appears to make securing train stations easier."
Q_7	"The SSS results in enough security at all three levels of the train station."
Q_8	"The SSS results in security officials having a strong presence throughout the station."
Q_9	"The SSS results in ALL areas being patrolled as much as needed."
Q_{10}	"The SSS did NOT prevent patrollers from taking or completing an action."
Q_{11}	"The SSS provides security officials with enough time to secure all areas of the station."

Table 9 The 11 assertions used in the questionnaire during the FSE

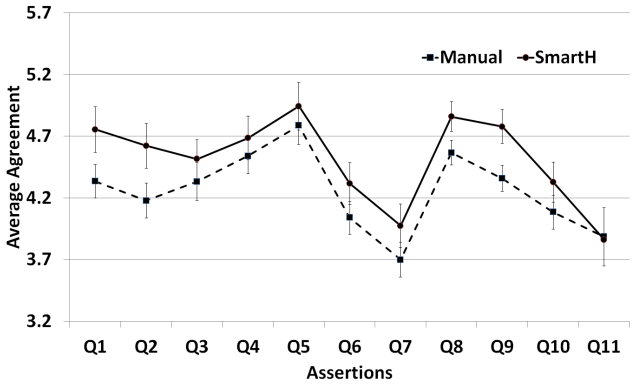
automatically generated in just a couple of hours. Hence, the effort required to generate the schedules using $\textsc{Smart}_{\text{H}}$ was much lower than the effort required to generate manual schedules, which, as discussed above, required one day of work due to its significant cognitive burden. Since typically such patrols would be conducted day-in and day-out for several days in high-threat periods, the savings of human effort achieved by game-theoretic schedulers are thus very significant.

6.3.2 Evaluation by Securty Experts

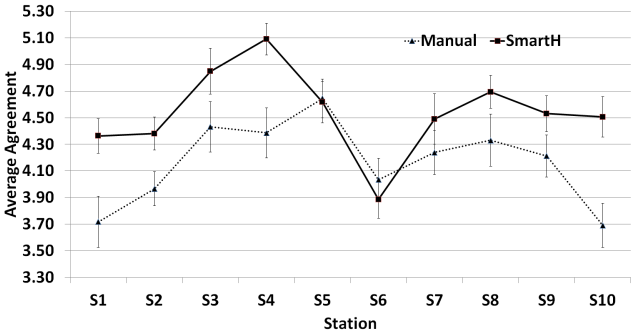
Each type of security allocation (either manual or game-theoretic based on $\textsc{Smart}_{\text{H}}$) was evaluated by security experts. In this setting, individual and joint activities between different resources played an important role.

For the purposes of this evaluation, a team of security experts (SEs) was placed at each station for the entire length of the exercise. Their task was to observe and evaluate the officers' patrolling activity during each sortie, and determine how their behavior was affecting the quality of the security within each station. In what follows, we report the conclusions of their analysis. The SEs did not know the type of schedules (so as to not bias their evaluation). To translate the observers' observations into a comparable value, each observer was asked to fill out a questionnaire every 30 minutes. The objective was to define a number of key sentences that could help to qualify the way in which the security officers had been patrolling the station in the last 30 minutes. Each questionnaire contained 11 *assertions* about the level of security within the station. Table 9 summarizes the assertions used in the questionnaire. Each assertion was a sentence defining a key aspect related to the security of a station. The assertions were defined in collaboration with a team of SEs from the LASD and with social scientists. Each SE had to determine his level of agreement with each assertion. The level of agreement was defined in the integer interval $\{0,6\}$, where 0 meant a strong disagreement, whereas 6 meant a strong agreement.

Figures 13(a) and 13(b) show the results that we obtained. Figure 13(a) shows the weighted average agreement obtained for each assertion calculated



(a) Assertions



(b) Stations

Fig. 13 Evaluation of the FSE: average agreement over the different questions and stations.

over all the stations (the average was calculated considering each station's corresponding weight). Figure 13(b) shows the average agreement obtained for each station calculated over all the assertions. The error bars in both figures show the standard error of the mean calculated for each specific assertion (in Figure 13(a)) and station (in Figure 13(b)). As we can see the difference between some data points of the two approaches do not seem to be statistically significant. A student t-test confirmed this trend. This is expected, since we were only able to collect data for few hours of a single day. Nonetheless, we can still acquire some interesting information about the performance of game-theoretic schedules in the field, by analyzing the results that are statistically significant. In the next section then, we will discuss how running additional experiments is a key challenge to confirm the trends presented here.

In Figure 13(a), we can see that $\textsc{Smart}_{\mathrm{H}}$ schedules seem to yield a higher level of agreement than manual schedules over all questions. As shown in the figure, the difference is significant only for assertions Q_1, Q_2, Q_8 and Q_9. As shown in Table 9, these four assertions correspond to very general statements about the security at each station which address the efficiency of the schedules,

their ability to provide a strong feeling of safety and to allow the officers to patrol each area as much as needed.

Similarly, in Figure 13(b), we can see that the average agreement is higher for SMART_H schedules over Manual schedules for stations S_1, S_2, S_3, S_4, S_8, S_9 and S_{10}. Some of these stations (S_1, S_8 and S_9) are the ones assigned a higher set of payoffs, as discussed above (see Figure 12). Hence, they correspond to the ones given a higher coverage by SMART_H (see Figure 12).

These results indicate that game-theoretic schedules were evaluated as more effective than manual schedules. Analysis reveals that whereas the game-theoretic schedules were able to incorporate joint activities in a fashion comparable to the human schedules– and this was important to ensure security effectiveness –the game-theoretic schedules were able to be more effective in two key ways that showed the limitations of human schedulers. First, manual schedules were generated by leaving the decision of which level of a station to patrol to each deployed team. The officers then, were not able to properly coordinate over the different levels to patrol and therefore they ended up patrolling the same levels. In do doing, the were not able to fully cover the different stations. In contrast, as shown in Appendix A, SMART_H produced a schedule which tackled the comprehensive security of all the ten different stations. The officers knew before-hand which levels they had to patrol and therefore, it was unnecessary for them to coordinate their decisions during the exercise.

Second, SMART_H produced a schedule which more effectively scheduled the VIPR team, i.e., the team with the highest effectiveness (0.8) for covering each target. As we can see in Table 11, the schedule generated by SMART_H had the VIPR team patrol *all* the most important stations at key levels. In contrast, manual schedules assigned the VIPR team, without accounting for its effectiveness. This made an impact on the security evaluators. By observing the VIPR at key locations, they considered the game-theoretic allocation more effective than the manual allocation, because it was using leveraging the abilties of the resources in a way that human experts could not achieve.

Overall, these results show the potential of game-theoretic security allocation to solve real world problems. In the next section, we discuss our future work, whereby we describe some new experiments that could be ran to further strenghten the results presented here.

7 Conclusions

This paper addressed the challenge of solving security games where multiple defenders resources receive benefits from performing joint coordinated activities. This challenge has not been addressed in previous work in security games. However, incorporating such joint activities into the existing SSG framework is critical for real-world applications.

To address this challenge, this paper presented four contributions. First, we presented SMART, a new type of SSG which accounts for multiple defenders

performing joint activities. Second, we presented two branch-and-price based approaches, SMART_O and SMART_H to solve SMART problem instances. SMART_O computes optimal solutions of SMART problem instances and uses a novel slave formulation that captures coordination in both space and time. SMART_H is an heuristic approach based on reward shaping and TSP ordering to speed-up the computation. Third, we provide proofs of theoretical properties of our algorithms while also showing the improved performance of the key components in simulation.

Fourth, we present the first head-to-head comparison between SSG based schedules and manual schedules in the field. To the best of our knowledge, this evaluation constitutes one of the largest evaluation of *algorithmic* game theory in the field to date. In more detail, we present the results that we obtained by organizing a large scale real-world experiment whereby 80 security officers (divided into 23 teams) patrolled 10 stations of a metro line for one day. In this experiment, we ran a head-to-head comparison between SSG-based schedules, generated using SMART_H, and human-generated schedules. Our results were based on an analysis provided by a team of security experts analyzing the performance of each type of schedule at each of the ten stations. The results showed that in comparison with human schedulers, game-theoretic schedules were able to reduce the effort to generate schedules, while improving coordination and perception of security presence.

In terms of future work, we aim to address one key challenge, that of execution uncertainty. In some security domains, the deployed resources may sometimes be delayed. For instance, USCG officers might be delayed because they need to board a boat. To address this challenge, our idea is to generalise the current model into a decentralised Markov decision process (dec-MDP [6]). Recently, this challenge was addressed by Shieh et al. [35]. In their work, Shieh et al. [35] propose a new security game model where a Dec-MDP is used to model execution uncertainty between different resources, i.e., situations where one resource is delayed and cannot complete his schedules albeit all the resources are still required to coordinate. The increase in complexity of the new model, which blends a security game and a DecMDP, is addressed by proposing a novel set of approximate algorithms which use column generation in a manner similar to the one presented in our work.

Additionally, as shown by the results presented in this paper, we believe that our work opens the door of applied research in security games to the realm of field evaluation. Given the strong connection that research in SSGs shares with real world security allocation problems, we argue that field evaluation should become a key area for future research in security games.

8 Acknowledgements

The authors would like to acknowledge their appreciation for the collaboration of the Los Angeles Sheriff's Department (LASD), the Booz-Allen Hamilton Company and the Transportation Security Administration's (TSA) Inter-

modal Security Training and Exercise Program (I-STEP). The LASD provided us with exceptional support and preparation which allowed us to organize our experiments with great detail and accuracy. Booz-Allen managed TSA's I-STEP Los Angeles Mass Transit Full-Scale Exercise, conducted May 16, 2014, thus allowing us to run experiments and collect data in very realistic and practical conditions. This research was supported by the United States Department of Homeland Security (DHS) through the National Center for Risk and Economic Analysis of Terrorism Events (CREATE) at the University of Southern California (USC) under Basic Ordering Agreement HSHQDC-10-A-BOA19, Task Order No. HST02-12-J-MLS151. 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, or the University of Southern California, or CREATE.

Appendix A

This appendix presents two tables:

- Table 10 depicts the security allocation resulting from the manual allocation process
- Table 11 depicts the security allocation resulting from the game-theoretic allocation process based on the $\textsc{Smart}_{\mathrm{H}}$ algorithm.

	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}
15 *min*		$VIPR$				T_{510}	T_{49}	T_{38}		T_{16}
30 *min*			$VIPR$					T_{38}		T_{16}
45 *min*			$VIPR$	$\frac{VIPR}{CRM_3}$	T_{510}			$\frac{EK9_2}{CRM_1}$		$\frac{EK9_1}{CRM_2}$
1 *Hour*										
15 *min*				$VIPR$		$\frac{EK9_1}{CRM_2}$				
30 *min*		CRM_3					$\frac{EK9_1}{CRM_1}$		$\frac{EK9_2}{CRM_2}$	
45 *min*	CRM_3		T_{38}	T_{49}	$HVWT_{12}$					
2 *Hours*					$HVWT_{12}$					
15 *min*										
30 *min*	T_{11}					$VIPR$	$VIPR$		T_{27}	
45 *min*	T_{11}	$VIPR$							T_{27}	
3 *Hours*										

Table 10 The human-generated security allocation

	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}
15 min	$T_{510}(s)$									
30 min	$T_{11}(m)$		$HVWT_{12}$(p)	$T_{38}(m)$			$T_{16}(m)$		$T_{27}(p)$	$T_{510}(p)$
45 min	$VIPR(p_1)$									
1 $Hour$	$T_{11}(p_2)$			$HVWT_{12}(p)$	$T_{38}(p)$					
15 min			$CRM_2(s)$			$VIPR(p)$		$T_{16}(p)$	$T_{49}(p)$	
30 min		$HVWT_{34}(p)$				$\frac{CRM_1}{EK9_1}(s)$	$VIPR(p)$	$T_{27}(m)$		$T_{49}(m)$
45 min				$CRM_3(s)$						
2 $Hours$								$\frac{CRM_1}{EK9_1}(s)$		
15 min						$HVWT_{34}(m)$		$VIPR(p)$		$\frac{CRM_2}{EK9_2}(s)$
30 min		$CRM_3(s)$	$T_{49}(m)$						$\frac{CRM_1}{EK9_1}(s)$	
45 min					$CRM_3(s)$	$\frac{CRM_2}{EK9_2}(s)$			$VIPR(m)$	
3 $Hours$										

Table 11 Security allocation generated by SMART_H: s represents the street level of a station, m the mezzanine level and p the platform level.

References

1. N. Agmon, S. Kraus, and G. A. Kaminka. Multi-robot perimeter patrol in adversarial settings. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, pages 2339–2345, 2008.
2. N. Agmon, G. A. Kaminka, and S. Kraus. Multi-robot adversarial patrolling: facing a full-knowledge opponent. *Journal of Artificial Intelligence Research (JAIR)*, 42(1):887–916, 2011.
3. Noa Agmon, Vladimir Sadov, Gal A. Kaminka, and Sarit Kraus. The Impact of Adversarial Knowledge on Adversarial Planning in Perimeter Patrol. In *AAMAS*, volume 1, pages 55–62, 2008.
4. C. Barnhart, E.L. Johnson, G.L. Nemhauser, M.W.P. Savelsbergh, and P.H. Vance. Branch and price: Column generation for solving huge integer programs. In *Operations Research*, volume 46, pages 316–329, 1994.
5. N. Basilico, N. Gatti, and F. Amigoni. Leader-follower strategies for robotic patrolling in environments with arbitrary topologies. In *Proceedings of the Eight International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 57–64, 2009.
6. R. Becker, S. Zilberstein, V. Lesser, and C. V. Goldman. Solving Transition Independent Decentralized Markov Decision Processes. *Journal of Artificial Intelligence Research (JAIR)*, 22:423–455, 2004.
7. D. Bertsimas and J. N. Tsitsiklis. *Introduction to Linear Optimization.* Athena Scientific, 1994.
8. A. Brown, C. F. Camerer, and D. Lovallo. To review or not to review? limited strategic thinking at the movie box office. *American Economic Journal: Microeconomics*, 4(2):1–26, 2012.
9. R. V. Clarke. *Preventing Mass Transit Crime - Crime Prevention Studies*, volume 6. Criminal Justice Press, 1996.
10. R. V. Clarke and Graeme Newman. Police and the prevention of crime. *Policing: A Journal of Policy and Practice*, 1:9–20, 2007.
11. V. Conitzer. Computing game-theoretic solutions and applications to security. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 2106–2112, 2012.
12. V. Conitzer and T. Sandholm. Computing the optimal strategy to commit to. In *Proceedings of the Seventh ACM Conference on Electronic Commerce (EC)*, pages 82–90, 2006.
13. J. P. Dickerson, G. I. Simari, V. S. Subrahmanian, and Sarit Kraus. A graph-theoretic approach to protect static and moving targets from adversaries. In *Proceedings of the Ninth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 299–306, 2010.
14. F. Fang, A. X. Jiang, and M. Tambe. Protecting moving targets with multiple mobile resources. *Journal of Artificial Intelligence Research (JAIR)*, 48:583–634, 2013.
15. N. Gatti. Game theoretical insights in strategic patrolling: Model and algorithm in normal form. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, pages 403–407, 2008.

16. G. Gutin, A. Yeo, and A. Zverovich. Traveling salesman should not be greedy: domination analysis of greedy-type heuristics for the tsp. *Discrete Applied Mathematics*, 117:81–86, 2002.
17. M. Jain, E. Kardes, C. Kiekintveld, M. Tambe, and F. Ordonez. Security games with arbitrary schedules: A branch and price approach. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 792–797, 2010.
18. M. Jain, J. Pita, J. Tsai, C. Kiekintveld, S. Rathi, F. Ordonez, and M. Tambe. Software assistants for patrol planning at lax and federal air marshals service. *Interfaces*, 40(4):267–290, 2010.
19. A. X. Jiang, A. D. Procaccia, Y. Qian, N. Shah, and Milind Tambe. Defender (mis)coordination in security games. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 199–206, 2013.
20. A. X. Jiang, Z. Yin, C. Zhang, M. Tambe, and S. Kraus. Game-theoretic randomization for security patrolling with dynamic execution uncertainty. In *Proceedings of the Twelfth International Conference on Autonomous Agents and Multiagent Systems*, pages 207–214, 2013.
21. C. Kiekintveld, M. Jain, J. Tsai, J. Pita, F. Ordóñez, and M. Tambe. Computing optimal randomized resource allocations for massive security games. In *Proceedings of The Eighth International Conference on Autonomous Agents and Multiagent Systems*, pages 233–239, 2009.
22. D. Korzhyk, V. Conitzer, and R. Parr. Security games with multiple attacker resources. In *Proceedings of the Twenty-second International Joint Conference on Artificial Intelligence (IJCAI)*, pages 273–279, 2011.
23. D. Korzhyk, V. Conitzer, and R. Parr. Solving stackelberg games with uncertain observability. In *Proceedings of the Tenth International Conference on Agents and Multi-agent Systems (AAMAS)*, pages 1013–1020, 2011.
24. J. Letchford and V. Conitzer. Solving security games on graphs via marginal probabilities. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 591–597, 2013.
25. J. Letchford and Vorobeychik. Optimal interdiction of attack plans. In *Proceedings of the Twelfth International Conference of Autonomous Agents and Multi-agent Systems (AAMAS).*, pages 199–206, 2013.
26. J. Letchford, L. MacDermed, V. Conitzer, R. Parr, and C. L. Isbell. Computing optimal strategies to commit to in stochastic games. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 1380–1386, 2012.
27. A. Machado, G. Ramalho, J. D. Zucker, and A. Drogoul. Multi-agent patrolling: An empirical analysis of alternative architectures. In *Proceedings of the International Conference of Multi-Agent-Based Simulation*, pages 155–170, 2003.
28. G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions–I. *Mathematical Programming*, 14(1):265–294, Dec 1978.

29. R. Ostling, J. Wang, J. Tao-yi, E. Y. Chou, and C. F. Camerer. Testing game theory in the field: Swedish lupi lottery games. *American Economic Journal: Microeconomics*, 3(3):1–33, 2011.
30. P. Paruchuri, J. P. Pearce, J. Marecki, M. Tambe, F. Ordonez, and S. Kraus. Playing games for security: An efficient exact algorithm for solving bayesian stackelberg games. In *Proceedings of the Seventh International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 539–547, 2008.
31. J . Pita, M. Jain, C . Western, C. Portway, M. Tambe, F. Ordonez, S. Kraus, and P. Paruchuri. Deployed armor protection: The application of a game theoretic model for security at the los angeles international airport. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2008.
32. J. Pita, M. Tambe, C. Kiekintveld, S. Cullen, and E. Steigerwald. GUARDS - Innovative Application of Game Theory for National Airport Security. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2710–2715, 2011.
33. E. Shieh, B. An, R. Yang, M. Tambe, C. Baldwin, J. DiRenzo, B. Maule, and G. Meyer. PROTECT: A Deployed Game Theoretic System to Protect the Ports of the United States. In *Proceedings of the Eleventh International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 13–20, 2012.
34. E. Shieh, M. Jain, A. X. Jiang, and M. Tambe. Efficiently solving joint activity based security games. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 346–352, 2013.
35. E. Shieh, A. X. Jiang, A. Yadav, P. Varakantham, and M. Tambe. Unleashing dec-mdps in security games: Enabling effective defender teamwork. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, 2014. In press.
36. E. Sless, N. Agmon, and S. Kraus. Multi-robot adversarial patrolling: Facing coordinated attacks. In *Proceedings of the Thirteenth International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 1093–1100, 2014.
37. R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, 1998.
38. M. Tambe. *Security and Game Theory: Algorithms, Deployed Systems, Lessons Learned.* Cambridge University Press, 2011.
39. J. Tsai, S. Rathi, C. Kiekintveld, F. Ordóñez, and M. Tambe. IRIS - a tool for strategic security allocation in transportation networks. In *Proceedings of the Eight International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 831–839, 2009.
40. O. Vanek, B. Bosansky, M. Jakob, and M. Pechoucek. Transiting areas patrolled by a mobile adversary. In *IEEE Symposium on Computational Intelligence and Games (CIG)*, pages 9–16. IEEE, 2010.
41. O. Vanek, M. Jakob, O. Hrstka, and M. Pechoucek. Using multi-agent simulation to improve the security of maritime transit. In *MABS*, 2011.

42. P. Varakantham, H. Chuin Lau, and Z. Yuan. Scalable randomized patrolling for securing rapid transit networks. In *Proceedings of the Conference for Innovative Applications for Artificial Intelligence (IAAI)*, pages 1563–1568, 2013.
43. Y. Vorobeychik and S. Singh. Computing stackelberg equilibria in discounted stochastic games. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 1478–1484, 2012.
44. Z. Yin, A. Jiang, M. Johnson, M. Tambe, C. Kiekintveld, K. Leyton-Brown, T. Sandholm, and J. Sullivan. Trusts: Scheduling randomized patrols for fare inspection in transit systems. In *Proceedings of the Conference on Innovative Applications for Artificial Intelligence (IAAI)*, pages 59–72, 2012.