Optimal defender allocation for massive security games: A branch and price approach

Manish Jain, Erim Kardeş, Christopher Kiekintveld, Fernando Ordóñez,
Milind Tambe
University of Southern California,
Los Angeles, CA 90089
{manish.jain,kardes,kiekintv,fordon,tambe}@usc.edu

ABSTRACT

Algorithms to solve security games, an important class of Stackelberg games, have seen successful real-world deployment by LAX police and the Federal Air Marshal Service. These algorithms provide randomized schedules to optimally allocate limited security resources for infrastructure protection. Unfortunately, these state-of-the-art algorithms fail to scale-up or to provide a correct solution for massive security games with arbitrary scheduling constraints. This paper provides ASPEN, a branch-and-price algorithm to overcome this limitation based on two key contributions: (i) A column-generation approach that exploits an innovative compact network flow representation, avoiding a combinatorial explosion of schedule allocations; (ii) A branch-and-bound approach with novel upper-bound generation via a fast algorithm for solving under-constrained security games. ASPEN is the first known method for efficiently solving real-world-sized security games with arbitrary schedules. This work contributes to a very new area of work that applies techniques used in large-scale optimization to game-theoretic problems—an exciting new avenue with the potential to greatly expand the reach of game theory.

Categories and Subject Descriptors

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

General Terms

Algorithms, Security, Arbitrary Scheduling Problem, Performance

Keywords

Game Theory, Stackelberg Games, Algorithms, Uncertainty, Security, Randomization, Column Generation, Branch and Price, ASPEN, SPARS

1. INTRODUCTION

Algorithms for attacker-defender Stackelberg games, resulting in randomized schedules for deploying limited security resources at airports, subways, ports, and other critical infrastructure have garnered significant research interest [13, 9]. Indeed, two important deployed security applications are using such algorithms: ARMOR and IRIS. ARMOR has been in use for over two years by Los Angeles International Airport police to generate canine-patrol and vehicle-checkpoint schedules [15, 14]. IRIS was recently deployed by the Federal Air Marshals Service (FAMS) to create flight schedules for air marshals [16][1]. These applications are fueled by efficient algorithms that enable scale-up [13, 5, 2, 7] in the input games, with the latest significant scale-up achieved in ERASER-C, the algorithm used in IRIS [9].

Unfortunately, current state of the art algorithms for Stackelberg games are inadequate for larger security scheduling applications [14]. For example, given that US carriers fly over 27,000 domestic and 2,000 international flights daily, generating randomized flight schedules for the limited air marshals resources is a massive scheduling challenge. IRIS addresses an important part of this space — the international sector — but only considers schedules of size two (one departure and one return flight). However, algorithms like ERASER-C in IRIS fail to scale-up and/or to provide a correct solution when air marshals are allowed to fly tours of more than two flights (common in the domestic sector) [9]. The culprit is the exponential explosion in the defender's strategy space in such games caused by the arbitrary size and structure of possible security schedules, and the concomitant combinatorial allocations of security resources to schedules. Indeed, as shown by Korzhyk et al. [10], the problem can be solved in polynomial time only if the schedules are of size 0 or 1, or if there is exactly one resource type for a schedule size of 2, and is NP-hard in general.

Motivated by FAMS and other applications with complex scheduling constraints, including transportation networks and border patrols, this paper presents algorithms for SPARS (*Security Problems with ARbitrary Schedules*) — where there are no special restrictions on the possible schedules. Our main contribution is ASPEN (Accelerated SPars ENgine), a new algorithm for SPARS solving massive Stackelberg security games with arbitrary scheduling constraints. ASPEN is based on the branch and price framework used to solve very large mixed-integer programs, and provides two novel contributions. First, ASPEN uses column generation to avoid representing the full (exponential) strategy space for the defender. To this end, we provide a novel decomposition of the security scheduling problem (SPARS) into a master problem and a network flow subproblem that can be used to generate feasible defender strategies as needed. Second, ASPEN uses a novel branch-and-bound method for searching the space of attacker strategies, achieving significant performance improvements by integrating branching criteria and bounds using the ORIGAMI algorithm [9]. Additionally, we apply column generation to improve the scope of correct application of previous methods such as ERASER-C. We evaluate ASPEN empirically on problems motivated by the FAMS domain, illustrating that this is the first known method for efficiently solving real-world-sized security games with arbitrary schedules.

2. SPARS

[1]FAMS deploys armed air marshals on US commercial aircraft to deter and defeat terrorist/criminal attempts to gain control of the aircraft.

We adopt the model of security games from Kiekintveld et. al. [9], though our work here will focus on the most general type of scheduling problem allowed in this framework. A security game is a two-player game between a defender and an attacker. The attacker's pure strategy space $\mathcal{A}$ is the set of targets T that could be attacked, $T = \{t_1, t_2, \ldots, t_n\}$. The corresponding mixed strategy $\mathbf{a} = \langle a_i \rangle$ is a vector where a_i represents the probability of the adversary attacking t_i. The defender allocates resources of different types $\lambda \in \Lambda$ to protect targets, with the number of available resources given by $R = \{r_1, r_2, \ldots, r_{|\Lambda|}\}$. Each resource can be assigned to a *schedule* covering multiple targets, $s \subseteq T$, so the set of all legal schedules $S \subseteq \mathcal{P}(T)$, where $\mathcal{P}(T)$ represents the power set of T. There is a set of legal schedules for each λ, $S_\lambda \subseteq S$.

The defender's pure strategies are the set of *joint schedules* that assign each resource to at most one schedule. Additionally, we assume that a target may be covered by at most 1 resource in a joint schedule (though this can be generalized). A joint schedule $\mathbf{j}$ can be represented by the vector $\mathbf{P_j} = \langle P_{jt} \rangle \in \{0, 1\}^n$ where P_{jt} represents whether or not target t is covered in joint schedule $\mathbf{j}$. We define a mapping M from $\mathbf{j}$ to $\mathbf{P_j}$ as: $M(\mathbf{j}) = \langle P_{jt} \rangle$, where $P_{jt} = 1$ if $t \in \bigcup_{s \in \mathbf{j}} s$; $P_{jt} = 0$ otherwise. The set of all feasible joint schedules is denoted by $\mathbf{J}$. The defender's mixed strategy $\mathbf{x}$ specifies the probabilities of playing each $\mathbf{j} \in \mathbf{J}$, where each individual probability is denoted by x_j. Let $\mathbf{c} = \langle c_t \rangle$ be the vector of coverage probabilities corresponding to $\mathbf{x}$, where $c_t = \sum_{\mathbf{j} \in \mathbf{J}} P_{jt} x_j$ is the marginal probability of covering t.

Payoffs depend on the target attacked and whether or not a defender resource is covering the target. $U_d^c(t)$ denotes the defender's utility if t is attacked when it is covered by a resource of any type. If t is not covered, the defender gets $U_d^u(t)$. Likewise, the attacker's utilities are denoted by $U_a^c(t)$ and $U_a^u(t)$. We assume adding coverage to target t is strictly better for the defender and worse for the attacker: $U_d^c(t) > U_d^u(t)$ and $U_a^c(t) < U_a^u(t)$, however, not necessarily zero-sum. For a strategy profile $\langle \mathbf{c}, \mathbf{a} \rangle$, the expected utilities for the defender and attacker are given by:

$$U_d(\mathbf{c}, \mathbf{a}) = \sum_{t \in T} a_t \left(c_t U_d^c(t) + (1 - c_t) U_d^u(t) \right) \quad (1)$$

$$U_a(\mathbf{c}, \mathbf{a}) = \sum_{t \in T} a_t \left(c_t U_a^c(t) + (1 - c_t) U_a^u(t) \right) \quad (2)$$

We adopt a Stackelberg model in which the defender acts first and the attacker chooses a strategy after observing the defender's mixed strategy. Stackelberg games are common in security domains where attackers can surveil the defender strategy [13]. The standard solution concept is Strong Stackelberg Equilibrium (SSE) [11, 4, 17, 9], in which the leader selects an optimal mixed strategy based on the assumption that the follower will choose an optimal response, breaking ties in favor of the leader.[2] There always exists an optimal pure-strategy response for the attacker, so we restrict our attention to this set in the rest of the paper. The formal definition of a Stackelberg equilibrium is given in Definition 1. The problem of finding Stackelberg equilibria in security games has been shown to be in polynomial time only if the schedule size is 0 or 1, or if the schedule size is 2 and there is exactly one resource type; it is NP-hard in all other settings [10].

DEFINITION 1. *A pair of strategies $\langle \mathbf{C}, g \rangle$ forms a* Strong Stackelberg Equilibrium *(SSE) if they satisfy the following:*

1. *The leader (defender) plays a best-response: $U_d(\mathbf{c}, g(\mathbf{c})) \geq U_d(\mathbf{c}', g(\mathbf{c}'))$, for all $\mathbf{c}'$.*
2. *The follower (attacker) plays a best-response: $U_a(\mathbf{c}, g(\mathbf{c})) \geq U_a(\mathbf{c}, g'(\mathbf{c}))$, for all $\mathbf{c}, g'$.*
3. *The follower breaks ties optimally for the leader: $U_d(\mathbf{c}, g(\mathbf{c})) \geq U_d(\mathbf{c}, \tau(\mathbf{c}))$, for all $\mathbf{c}$, where $\tau(\mathbf{c})$ is the set of follower best-responses to $\mathbf{c}$.*

Example: Consider a FAMS game with 5 targets (flights), $T = \{t_1, \ldots, t_5\}$, and three marshals of the same type, $r_1 = 3$. Let the set of feasible schedules be as follows:

$$S_1 = \{\{t_1, t_2\}, \{t_2, t_3\}, \{t_3, t_4\}, \{t_4, t_5\}, \{t_1, t_5\}\}$$

Thus, in this example, the set of targets can be thought of as being arranged in a pentagon, where each of the five links corresponds to a schedule. The set of feasible joint schedules is shown below, where column $\mathbf{J}_1$ represents the joint schedule $\{\{t_1, t_2\}, \{t_3, t_4\}\}$. Thus, since targets t_1, t_2, t_3 and t_4 are covered by $\mathbf{J}_1$, $\mathbf{P_{J_1}}$ has one's in the corresponding entries and 0 corresponding to t_5.

$$\mathbf{P} = \begin{array}{c} \\ t_1: \\ t_2: \\ t_3: \\ t_4: \\ t_5: \end{array} \begin{array}{c} \mathbf{J}_1 \quad \mathbf{J}_2 \quad \mathbf{J}_3 \quad \mathbf{J}_4 \quad \mathbf{J}_5 \\ \begin{bmatrix} 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \end{bmatrix} \end{array}$$

Each joint schedule in $\mathbf{J}$ assigns only 2 air marshals in this example, since no more than 1 FAM is allowed on any flight. Thus, the third air marshal will remain unused. Suppose all of the targets have identical payoffs $U_d^c(t) = 1$, $U_d^u(t) = -5$, $U_a^c(t) = -1$ and $U_a^u(t) = 5$. In this case, the optimal strategy for the defender randomizes uniformly across the joint schedules, $\mathbf{x} = \langle .2, .2, .2, .2, .2 \rangle$, resulting in coverage $\mathbf{c} = \langle .8, .8, .8, .8, .8 \rangle$. All pure strategies have equal payoffs for the attacker, given this coverage vector.

3. ASPEN SOLUTION APPROACH AND RELATED WORK

The ERASER-C mixed-integer linear program [9] is the most recent algorithm developed for larger and more complex Stackelberg security games [5]. Whereas previous work has focused on patrolling arbitrary topologies using Stackelberg games [2, 13], it has typically focused on a single defender. In contrast, ASPEN and ERASER-C focus on games with large numbers of defenders of different types, handling the combinatorial explosion in the defender's joint schedules. Unfortunately, as the authors note, ERASER-C may fail to generate a correct solution in cases where arbitrary schedules with more than two flights (i.e., multi-city tours) are allowed in the input, or when the set of flights cannot be partitioned into distinct sets for departure and arrival flights. For instance, ERASER-C incorrectly outputs the coverage vector $\mathbf{c} = \langle 1, 1, 1, 1, 1 \rangle$ for the example above (no legal joint schedule can cover more than 4 targets, so it is not possible to cover all targets with probability 1). ERASER-C avoids enumerating joint schedules to gain efficiency, but loses the ability to correctly model arbitrary schedules. Furthermore, ERASER-C only outputs a coverage vector $\mathbf{c}$ and not the distribution $\mathbf{x}$ over joint schedules $\mathbf{J}$ necessary to implement the coverage in practice (though for some restricted cases it is possible to construct $\mathbf{x}$ from $\mathbf{c}$ in polynomial time using the Birkhoff-von Neumann Theorem [10]). New algorithms are needed to solve general SPARS problems that ERASER-C cannot handle.

[2]This tie-breaking rule is counter-intuitive, but the defender can make this response strictly optimal for the attacker by playing a strategy an infinitesimal ϵ away from the SSE strategy.

SPARS problems can be formulated as mixed-integer programs in which adversary strategies are represented by integer variables $\mathbf{a}$ with $a_t = 1$ if target t is attacked and 0 otherwise. Two key computational challenges arise in this formulation. First, the space of possible strategies (joint schedules) for the defender suffers from combinatorial explosion: a FAMS problem with 100 flights, schedules with 3 flights, and 10 air marshals has up to 100,000 schedules and $\binom{100000}{10}$ joint schedules. Second, integer variables are a well-known challenge for optimization. Branch and Price [1] is a framework for solving very large optimization problems that combines branch and bound search with column generation to mitigate both of these problems. Column generation [6] can be viewed as a "double oracle" algorithm [12], and is used to avoid explicitly enumerating all the variables in a large problem (in our problem, these variables represent the joint schedules). This method operates on joint schedules (and not marginal probabilities, like ERASER-C), so it is able to handle arbitrary scheduling constraints directly.

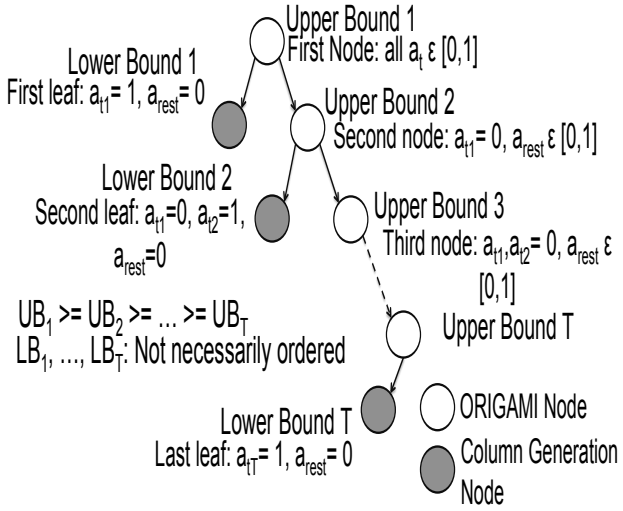


Figure 1: Working of Branch and Price

An example of branch and price for our problem is shown in Figure 1, with the root representing the original problem. Branches to the left (gray nodes) set exactly one variable t_i in $\mathbf{a}$ to 1 and the rest to zero, resulting in a linear program that gives a lower bound on the overall solution quality. Branches to the right fix variable t_i to zero, leaving the remaining variables unconstrained. An upper bound on solution quality computed for each white node can be used to terminate execution without exploring all of the possible integer assignments. Solving the linear programs in each gray node normally requires enumerating all joint schedules for the defender. Column generation (i.e., pricing) is a technique that avoids this by iteratively solving a restricted *master problem*, which includes only a small subset of the variables, and a *slave problem*, that identifies new variables to include in the master problem to improve the solution.

Unfortunately, branch and price and column generation are not "out of the box approaches" and have only recently begun to be applied in game-theoretic settings [8]. We introduce a novel master-slave decomposition to facilitate column generation for SPARS, including a network flow formulation of the slave problem. We also show experimentally that conventional linear programming relaxations used for branch and bound perform poorly in this domain, and we replace them with novel techniques based on fast algorithms for security games without scheduling constraints.

4. ASPEN COLUMN GENERATION

The linear programs at each leaf in Figure 1 are decomposed into into *master* and *slave* problems for column generation (see Algorithm 1). The master solves for the defender strategy $\mathbf{x}$, given a restricted set of columns (i.e., joint schedules) $\mathbf{P}$. The objective function for the slave is updated based on the solution of the master, and the slave is solved to identify the best new column to add to the master problem, using *reduced costs* (explained later). If no column can improve the solution the algorithm terminates.

Algorithm 1 Column generation employed at each leaf

1. Initialize P
2. Solve Master Problem
3. Calculate reduced cost coefficients from solution
4. Update objective of slave problem with coefficients
5. Solve Slave Problem

if Optimal solution obtained **then**

6. Return $(\mathbf{x}, \mathbf{P})$

else

7. Extract new column and add to $\mathbf{P}$
8. Repeat from Step 2

4.1 Master Problem

The master problem (Equations 3 to 8) solves for the probability vector $\mathbf{x}$ that maximizes the defender reward (Table 1 describes the notation). This master problem operates directly on columns of $\mathbf{P}$, and the coverage vector $\mathbf{c}$ is computed from these columns as $\mathbf{Px}$. Constraints 4–6 enforce the SSE conditions that the players choose mutual best-responses defined in Definition 1, mirroring similar constraints in ERASER-C. The defender expected payoff (Equation 1) for target t is given by the t^{th} component of the column vector $\mathbf{DPx} + \mathbf{U}_d^u$ and denoted $(\mathbf{DPx} + \mathbf{U}_d^u)_t$. Thus, Equation 4 coupled with Equation 3 corresponds to condition 1, the leader's best response, of Definition 1. Similarly, the attacker payoff for target t is given by $(\mathbf{APx} + \mathbf{U}_a^u)_t$. Constraints 4 and 5 are active only for the single target t^* attacked ($a_{t^*} = 1$). This target must be the adversary's best-response, due to Constraint 6, thus corresponding to condition 2, in Definition 1. The follower will break ties optimally for the leader in this formulation of the master problem, since the formulation will choose that target t^* which is the follower's best response and maximizes the defender payoff.

$$\max \quad d \tag{3}$$

$$\text{s.t.} \quad \mathbf{d} - \quad \mathbf{DPx} - \mathbf{U}_d^u \leq (\mathbf{1} - \mathbf{a})M \tag{4}$$

$$\mathbf{k} - \quad \mathbf{APx} - \mathbf{U}_a^u \leq (\mathbf{1} - \mathbf{a})M \tag{5}$$

$$\mathbf{APx} + \mathbf{U}_a^u \leq \mathbf{k} \tag{6}$$

$$\sum_{j \in J} x_j = 1 \tag{7}$$

$$\mathbf{x}, \mathbf{a} \geq 0 \tag{8}$$

4.2 Slave Problem

The purpose of the slave problem is to find the best column to add to the current columns in $\mathbf{P}$. This is done using *reduced cost*, which captures the total change in the defender payoff if a candidate column is added to $\mathbf{P}$. The candidate column with minimum reduced cost improves the objective value the most [3]. The reduced cost $\bar{c}_j$ of variable x_j, associated with column $\mathbf{P_j}$, is given

Table 1: Notation

Variable	Definition	Dimension
$\mathbf{J}$	Joint Schedules	$\|J\|$
T	Targets	$\|T\|$
$\mathbf{P}$	Mapping between T and J	$\|T\| \times \|J\|$
$\mathbf{x}$	Probability Distribution over J	$\|J\| \times 1$
$\mathbf{a}$	Attack vector	$\|T\| \times 1$
d	Defender Reward	-
k	Adversary Reward	-
$\mathbf{d}$	Column vector of d	$\|T\| \times 1$
$\mathbf{k}$	Column vector of k	$\|T\| \times 1$
$\mathbf{D}$	Diag. matrix of $U_d^c(t) - U_d^u(t)$	$\|T\| \times \|T\|$
$\mathbf{A}$	Diag. matrix of $U_a^c(t) - U_a^u(t)$	$\|T\| \times \|T\|$
$\mathbf{U}_d^u$	Vector of values $U_d^u(t)$	$\|T\| \times 1$
$\mathbf{U}_a^u$	Vector of values $U_a^u(t)$	$\|T\| \times 1$
M	Huge Positive constant	-

in Equation 9, where $\mathbf{w}, \mathbf{y}, \mathbf{z}$ and h are dual variables of master constraints 4, 5, 6 and 7 respectively. The dual variable measures the influence of the associated constraint on the objective, defender payoff, and can be calculated using standard techniques [3].

$$\bar{c}_j = \mathbf{w}^T(\mathbf{DP_j}) + \mathbf{y}^T(\mathbf{AP_j}) - \mathbf{z}^T(\mathbf{AP_j}) - h \quad (9)$$

An inefficient approach would be to iterate through all of the columns and calculate each reduced cost to identify the best column to add. Instead, we formulate a minimum cost network flow (MCNF) problem that efficiently finds the optimal column. Feasible flows in the network map to feasible joint schedules in the SPARS problem, so the scheduling constraints are captured by this formulation. For a SPARS instance we construct the MCNF graph G as follows.

A source node source_λ with supply r_λ is created for each defender type $\lambda \in \Lambda$. A single sink node has demand $\sum_{\lambda \in \Lambda} r_\lambda$. Targets in schedule s for resource λ are represented by pairs of nodes $(a_{s_\lambda,t}, b_{s_\lambda,t})$ with a connecting link (so each target corresponds to many nodes in the graph). For every schedule $s_\lambda \in S_\lambda$ we add the following path from the source to the sink:

$$\langle \text{source}_\lambda, a_{s,t_{i_1}}, b_{s,t_{i_1}}, a_{s,t_{i_2}}, \ldots, b_{s,t_{i_L}}, \text{sink} \rangle$$

The capacities on all links are set to 1, and the default costs to 0. A dummy flow with infinite capacity is added to represent the possibility that some resources are unassigned. The number of resources assigned to t in a column $\mathbf{P_j}$ is computed as follows:

$$\text{assigned}(t) = \sum_{s \in S} \text{flow}[\text{link}(a_{s,t}, b_{s,t})]$$

Constraints are added to G so that the number of scheduled resources on each target is not greater than 1. Therefore, the added constraints can be written down as follows:

$$\text{assigned}(t) \leq 1 \quad \forall\, t \in T.$$

A partial graph G for our earlier example is shown in Figure 2, showing paths for 3 of the 5 schedules. The paths correspond to schedules $\{t_1, t_2\}$, $\{t_2, t_3\}$ and $\{t_1, t_5\}$. The supply and demand are both 3, corresponding to the number of available FAMS. Double-bordered boxes mark the flows used to compute $\text{assigned}(t_1)$ and $\text{assigned}(t_2)$. Every joint schedule corresponds to a feasible flow in G. For example, the joint schedule $\{\{t_2, t_3\}, \{t_1, t_5\}\}$ has a flow of 1 unit each through the paths corresponding to schedules $\{t_2, t_3\}$ and $\{t_1, t_5\}$, and a flow of 1 through the dummy. Similarly, any feasible flow through the graph G corresponds to a feasible joint schedule, since all resource constraints are satisfied.

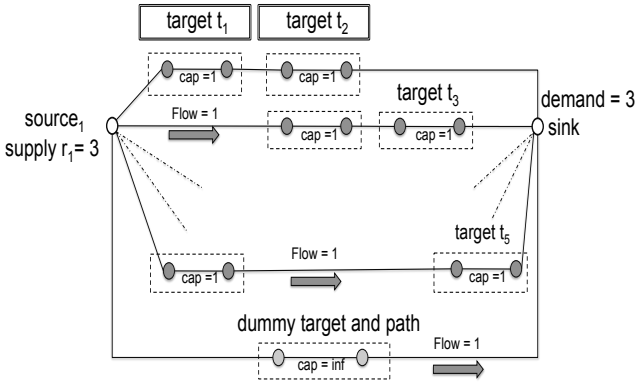


Figure 2: Example Network Graph

It remains to define link costs such that the cost of a flow is the reduced cost for the joint schedule. We decompose $\bar{c}_j$ into a sum of cost coefficients per target, $\hat{c}_t$, so that $\hat{c}_t$ can be placed on links $(a_{s,t}, b_{s,t})$ for all targets t. $\hat{c}_t$ is defined as follows:

$$\hat{c}_t = w_t.D_t + y_t.A_t - z_t.A_t \quad (10)$$
$$D_t = U_d^c(t) - U_d^u(t) \quad (11)$$
$$A_t = U_a^c(t) - U_a^u(t) \quad (12)$$

where w_t, y_t and z_t are t^{th} components of $\mathbf{w}, \mathbf{y}$ and $\mathbf{z}$. The overall objective given below for the MCNF problem sums the contributions of the reduced cost from each individual flow and subtracts the dual variable h. If this is non-negative, no column can improve the master solution, otherwise the optimal column (identified by the flow) is added to the master and the process iterates.

$$\min_{\text{flow}} \sum_{(a_{s,t}, b_{s,t})} \hat{c}_t.\text{flow}[(a_{s,t}, b_{s,t})] - h$$

To recap, the entire column generation algorithm employed at each node of the branch and bound tree shown in Figure 1 is given by Algorithm 1 and was described in this section. If the minimum reduced cost obtained at an iteration is non-negative, it indicates that the optimal solution for the current leaf in the tree has been achieved. Otherwise, a new column $\mathbf{P}_j$ is obtained by setting $P_{jt} = \text{assigned}(t)$ for all $t \in T$, and the master problem is re-solved.

5. IMPROVING BRANCHING AND BOUNDS

ASPEN uses branch and bound to search over the space of possible attacker strategies. A standard technique in branch and price is to use LP relaxation, i.e. allow the integer variables to take on arbitrary values, to give an optimistic bound on the objective value of the original MIP for each internal node. Unfortunately, our experimental results show that this generic method is ineffective in our domain. We introduce ORIGAMI-S, a novel branch and bound heuristic for SPARS based on ORIGAMI [9], which is an efficient solution method for security games without scheduling constraints and heterogeneous resources. We use ORIGAMI-S to solve a relaxed version of SPARS, and integrate this in ASPEN to give

bounds and select branches.

$$\min \quad k \tag{13}$$
$$\mathbf{U}_a(\mathbf{c}) = \quad \mathbf{Ac} + \mathbf{U}_a^u \tag{14}$$
$$\mathbf{0} \leq \quad \mathbf{k} - \mathbf{U}_a(\mathbf{c}) \tag{15}$$
$$\mathbf{k} - \mathbf{U}_a(\mathbf{c}) \leq \quad (\mathbf{1} - \mathbf{q}) \cdot M \tag{16}$$
$$c_t = \quad \textstyle\sum_{s \in S} \tilde{c}_{t,s} \qquad \forall t \in T \tag{17}$$
$$\sum_{s \in S_\lambda} \tilde{c}_{T_\lambda(s),s} \leq \quad r_\lambda \qquad \forall \lambda \in \Lambda \tag{18}$$
$$\sum_{t \in T} c_t \leq \quad L \cdot \textstyle\sum_{\lambda \in \Lambda} r_\lambda \tag{19}$$
$$\mathbf{c} \leq \quad \mathbf{q} \tag{20}$$
$$\mathbf{q} \in \quad \{0, 1\} \tag{21}$$
$$\mathbf{c}, c_{t,s} \in \quad [0, 1] \qquad \forall t \in T, s \in S \tag{22}$$

The ORIGAMI-S model is given in Equations 13–22. It minimizes the attacker's maximum payoff (Equations 13–16). The vector $\mathbf{q}$ represents the *attack set*, and is 1 for every target that gives the attacker maximal expected payoff (Equation 16). The remaining nontrivial constraints restrict the coverage probabilities. ORIGAMI-S defines a set of probabilities $\tilde{c}_{t,s}$ that represent the coverage of each target t in each schedule $s \in S_\lambda$. The total coverage c_t of target t is the sum of coverage on t across individual schedules (Equation 17). We define a set T_λ which contains one target from each schedule $s \in S_\lambda$. The total coverage assigned by resource type λ is upper-bounded by r_λ (Equation 18), analogous to the constraint that the total flow from a source in a network flow graph cannot be greater than the available supply. Total coverage is also bounded by multiplying the number of resources by the *maximum* size of any schedule (L) in Equation 19. The defender can never benefit by assigning coverage to nodes outside of the attack set, so these are constrained to 0 (Equation 20).

ORIGAMI-S is solved once at the beginning of ASPEN, and targets in the attack set are sorted by expected defender reward. The maximum value is an initial upper bound on the defender reward. The first leaf node that ASPEN evaluates corresponds to this maximum valued target (i.e, setting its attack value to 1), and a solution is found using column generation. This solution is a lower bound of the optimal solution, and the algorithm stops if this lower bound meets the ORIGAMI-S upper bound. Otherwise, a new upper bound from the ORIGAMI-S solution is obtained by choosing the second-highest defender payoff from targets in the attack set, and ASPEN evaluates the corresponding leaf node. This process continues until the upper bound is met, or the available nodes in the search tree are exhausted.

THEOREM 1. *The defender payoff, computed by ORIGAMI-S, is an upper bound on the defender's payoff for the corresponding SPARS problem. For any target not in the attack set of ORIGAMI-S, the restricted SPARS problem in which this target is attacked is infeasible.*

Proof Sketch: ORIGAMI and ORIGAMI-S both minimize the maximum attacker payoff over a set of feasible coverage vectors. If there are no scheduling constraints, minimizing the maximum attacker payoff also maximizes the defender's reward [9]. Briefly, the objective of both ORIGAMI and ORIGAMI-S is to maximize the size of the attack set, such that the coverage probability of each target in the attack set is also maximal. Both of these weakly improve the defender's payoff because adding coverage to a target is strictly better for the defender and worse for the adversary.

ORIGAMI-S makes optimistic assumptions about the amount of coverage probability the defender can allocate by taking the maximum that could be achieved by *any* legal joint schedule and allowing it to be distributed arbitrarily across the targets, ignoring the scheduling constraints. To see this, consider the marginal probabilities $\mathbf{c}^*$ of any legal defender strategy for SPARS. There is at least one feasible coverage strategy for ORIGAMI that gives the same payoff for the defender. Constraints 17 and 22 are satisfied by $\mathbf{c}^*$, because they are also constraints of SPARS. Each variable $\tilde{c}_{T_\lambda(s),s}$ in the set defined for Constraint 18 belongs to a single schedule associated with resource type λ, and at most r_λ of these can be selected in any feasible joint schedule, so this constraint must also hold for $\mathbf{c}^*$. Constraint 19 must be satisfied because it assumes that each available resource covers the largest possible schedule, so it generally allows excess coverage probability to be assigned. Finally, constraint 20 may be violated by $\mathbf{c}^*$ for some target t. However, the coverage vector with coverage identical to $\mathbf{c}^*$ for all targets in the ORIGAMI-S attack set and 0 coverage outside the attack set has identical payoffs (since these targets are never attacked).

6. COLUMN GENERATION ON MARGINALS FOR JOINT PATROLLING SCHEDULES

ERASER-C [9] is the algorithm that is used in IRIS and addresses the FAMS security game. However, it only generates the vector $\mathbf{c}$ (*marginals*), which specify the probability of each flight being covered by a federal air marshal. Joint schedules $\mathbf{P}$ and the distribution of over joint schedules $\mathbf{x}$ assigning every federal air marshal to a flight schedule is not provided. While the use of marginals in ERASER-C did provide an exponential scale-up over DOBSS [13], the best prior algorithm, the actual joint schedules still need to be generated from these marginals. In the section, we propose the use of the same column generation approach described in Section 4 to generate these joint schedules from the marginals output by ERASER-C. Thus, the objective of this section is to find joint schedules $\mathbf{P}$ and probability distribution $\mathbf{x}$ such that the obtained coverage vector is given by the output of ERASER-C, and can be formulated as:

$$\begin{aligned} \min_x \quad & ||\mathbf{Px} - \mathbf{c}||_1 \\ & \textstyle\sum_j x_j = 1 \\ & x_j \geq 0 \end{aligned} \tag{23}$$

Here, $||\mathbf{Px} - \mathbf{c}||_1$ refers to the L_1 norm of $||\mathbf{Px} - \mathbf{c}||_1$, or the sum of the absolute differences between each corresponding term of $\mathbf{Px}$ and $\mathbf{c}$. L_1 is chosen so as to keep the objective function linear.

The formulation of the associated master problem in this case is given from Equation 24 to Equation 28. The absolute distance between corresponding terms of $\mathbf{Px}$ and $\mathbf{c}$ is calculated in Equations 25 and 26, and is minimized in the objective as given in Equation 24.

$$\min_{x,\gamma} \sum_{t \in T} \gamma_t \tag{24}$$
$$s.t. \quad \mathbf{Px} - \gamma \leq \mathbf{c} \tag{25}$$
$$\mathbf{Px} + \gamma \geq \mathbf{c} \tag{26}$$
$$\sum_{t \in T} x_t = 1 \tag{27}$$
$$x \geq 0 \tag{28}$$

The slave problem in this case is the same as the one used before, where the reduced cost of a joint schedule is:

$$\bar{c}_j \quad = \quad -(\mathbf{w}_1 + \mathbf{w}_2)^T \mathbf{P_j} - \sigma \tag{29}$$
$$\tag{30}$$

where $\mathbf{w}_1, \mathbf{w}_2, \sigma$ are the optimal dual variables of the current master problem associated constraints 25, 26, and 27. Again, the reduced cost $\bar{c}_j$ can be decomposed into reduced cost coefficients per target $\hat{c}_t$, which can be computed using Equation 31.

$$\hat{c}_t = -(w_{1t} + w_{2t}) \quad (31)$$

7. EXPERIMENTAL RESULTS

We evaluate our algorithms on randomly generated instances of the scheduling problem, and provide two sets of results. We first compare the runtime results for ASPEN with regular Branch and Prices and column generation on the output of ERASER-C. We then provide runtime results for ASPEN when the problem size is scaled.

7.1 Comparison on FAMS domain

We compare the runtime performance of ASPEN, branch and price without the ORIGAMI-S heuristic (BnP) and ERASER-C. For this experiment we generate random instances of FAMS problems [9] with schedules of size two, with one departure flight and one arrival flight drawn from disjoint sets, so the set of feasible schedules form a bipartite graph with nodes as flights (for correct operation of ERASER-C). We vary the number of targets, defender resources, and schedules.

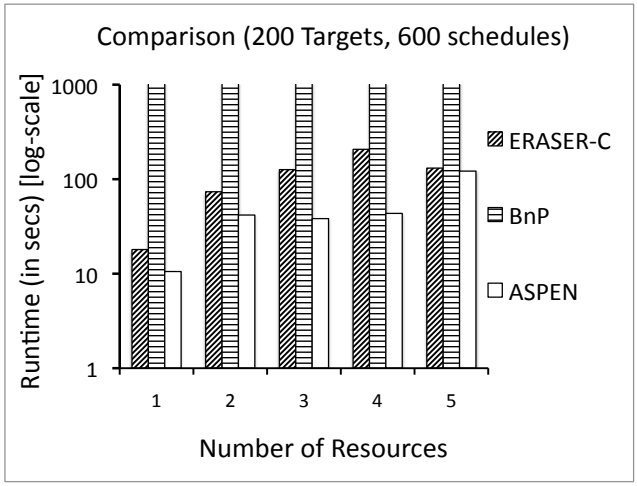


Figure 3: Runtime Comparison Changing Number of Resources

As mentioned earlier, ERASER-C outputs only the coverage vector $\mathbf{c}$. Obtaining the probability distribution $\mathbf{x}$ over joint schedules $\mathbf{J}$ from ERASER-C, specifically when the problem contains heterogeneous resources is a non-trivial challenge given the large numbers of joint schedules. This distribution over joint schedules was obtained using column generation on marginals as described in Section 6. In the following, when we refer to ERASER-C runtimes, they include the time for this column generation in order to allow a head-to-head comparison with ASPEN.

All experiments were based on 15 sample games, and problem instances that took longer than 30 minutes to run were terminated. Results varying the number of defender resources are shown in Figure 3. The y-axis shows the runtime in seconds on the *logarithmic scale*. The x-axis shows the number of resources. ASPEN is the fastest of the three algorithms. The effectiveness of the ORIGAMI-S bounds and branching are clear in the comparison with standard BnP method. The improvement over ERASER-C was an unexpected trend, and can be attributed to the number of columns generated by the two approaches, as shown in Table 2. In fact, ASPEN was 6 times faster than ERASER-C in some instances. Yet it is not the precise amount of speedup, but the fact that ASPEN is extremely competitive with ERASER-C in this specialized domain that is key; for ASPEN can correctly solve a far more general set of security games (SPARS) as we report next.

Table 2: Number of columns: 200 targets, 600 schedules

Resources	ASPEN	ERASER-C	BnP (max. 30 mins)
10	126	204	1532
20	214	308	1679
30	263	314	1976
40	227	508	1510
50	327	426	1393

We observe similar results in the second and third data sets presented in Figures 4 and 5. Figure 4 shows the results when the number of schedules is changed, where as Figure 5 shows the results when the number of targets is varied. The y-axis in both figures shows the runtime in seconds in a logarithmic scale. The x-axis shows the number of schedules and targets respectively. For example, the average runtime required by ERASER-C when there are 1000 schedules, 200 targets and 10 resources is 29.26 seconds for ERASER-C, 5.34 seconds for ASPEN and the simulation was terminated after 30 minutes for Branch and Price.

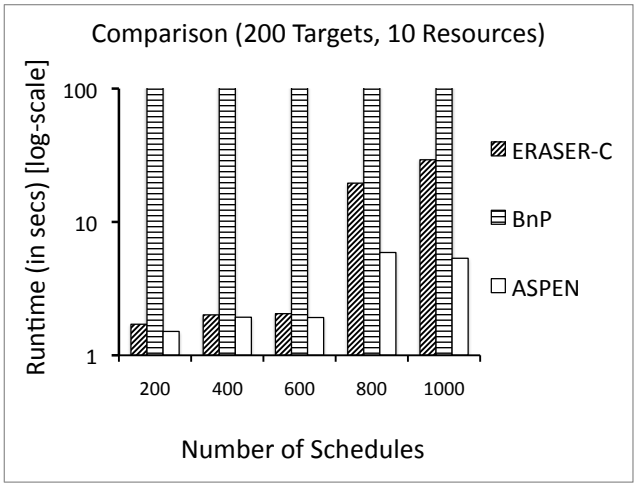
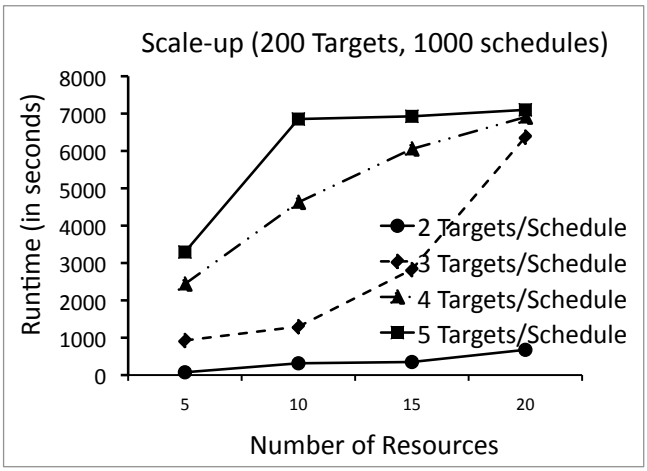


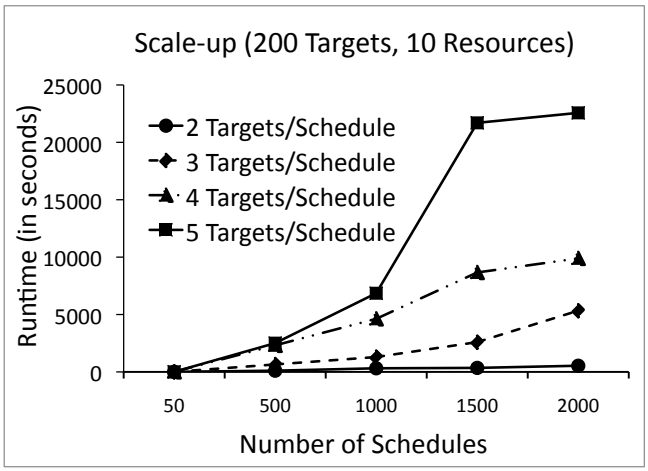
Figure 4: Runtime Comparison Changing Resources and Schedules

7.2 ASPEN on Large SPARS Instances

We also evaluate the performance of ASPEN on arbitrary scheduling problems as the size of the problem is varied to include very large instances. No comparisons could be made because ERASER-C does not handle arbitrary schedules and the only correct algorithms known, DOBSS [13] and BnP, do not scale to these problem sizes. Since ERASER-C does not handle arbitrary schedules, Branch and Price does not scale to these problem sizes (as shown in the previous section) and the only known algorithm (DOBSS [13]) that correctly solves this class of games requires an exponential representation size, ASPEN is the only algorithm capable of solving these instances. We vary the number of resources, schedules, and



(a) Resources



(b) Schedules

Figure 6: Runtime Scale-up Changing Number of Resources and Schedules is varied

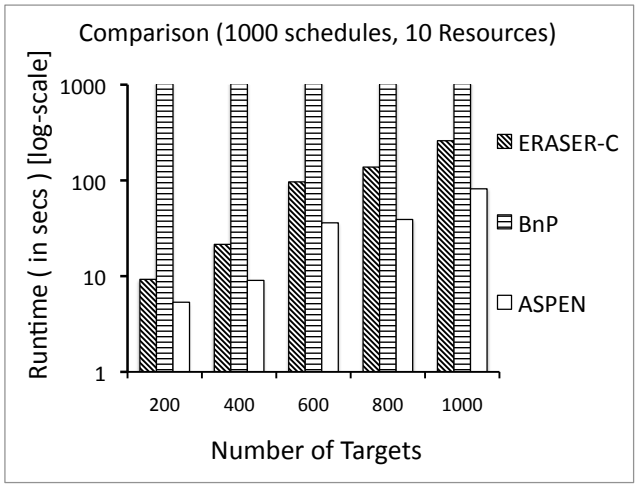


Figure 5: Runtime Comparison Changing Number of Targets

targets as before. In addition, we vary the number of targets per schedule for each of the three cases to test more complex scheduling problems.

Table 3: Number of columns: 200 **targets,** 1000 **schedules**

Resources	3 Tar. / sch.	4 Tar. / sch.	5 Tar. / sch.
5	456	518	658
10	510	733	941
15	649	920	1092
20	937	1114	1124

Figure 6(a) shows the runtime results with 1000 feasible schedules and 200 targets, averaged over 10 samples. The x-axis shows the number of resources, and the y-axis shows the runtime in seconds. Each line represents a different number of schedules per target. The number of joint schedules in these instances can be as large as 10^{23} ($\binom{1000}{10} \approx 2.6 \times 10^{23}$). Interestingly, the runtime does not increase much when the number of resources is increased from 10 to 20 when there are 5 targets per schedules. Column 4 of Table 3 illustrates that the key reason for constant runtime is that the average number of generated columns remains similar.

The graph also shows that increasing the complexity of schedules (i.e., the number of targets per schedule) increases the runtime. This happens because the complexity of the slave problem increases when the complexity of schedules is increased, in turn leading to the generation of more columns before the optimal solution is attained. This leads to the increase in runtime with the increase in complexity of schedules. This can also be seen in Table 3 when looking across a row. For example, the average number of columns required when the number of resources is 5 is $157, 456, 518$ and 658 when there are $2, 3, 4$ and 5 targets per schedule.

Similar trends are obtained in the other two sets of experiments as well. Figure 6(b) shows the runtime results when the number of schedules is increased, whereas Figure 7 shows the results when the number of targets is varied. The y-axes in both the cases shows the runtime in seconds, whereas the x-axis shows the number of schedules and targets respectively.

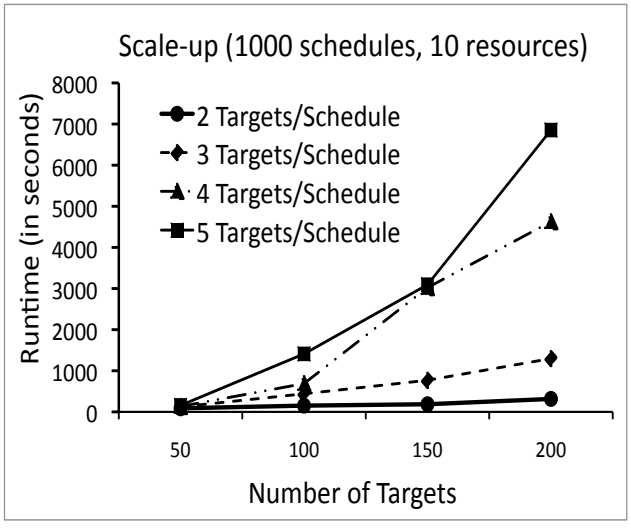


Figure 7: Runtime Scale-up Changing Number of Targets

8. CONCLUSIONS

We present a branch and price method, ASPEN, for solving large-scale Stackelberg security games with arbitrary constraints, including important real-world applications such as FAMS scheduling. ASPEN incorporates several novel contributions, including a decomposition of SPARS to enable column generation and the integration of ORIGAMI-S to substantially speed up the branch and bound search. Experimental results show that ASPEN is competitive with ERASER-C for the restricted class of games where ERASER-C is applicable. More importantly, ASPEN solves far more general instances of scheduling problems where ERASER-C and other existing techniques fail. ASPEN is also substantially faster than a standard implementation of branch and price for this domain. This work contributes to a very new area of work that applies techniques used in large-scale optimization to game-theoretic problems—an exciting new avenue with the potential to greatly expand the reach of game theory.

9. REFERENCES

[1] C. Barnhart, E. Johnson, G. Nemhauser, M. Savelsbergh, and P. Vance. Branch and price: Column generation for solving huge integer programs. In *Operations Research*, volume 46, pages 316–329, 1994.

[2] N. Basilico, N. Gatti, and F. Amigoni. Leader-follower strategies for robotic patrolling in environments with arbitrary topologies. In *AAMAS*, pages 500–503, 2009.

[3] D. Bertsimas and J. N. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, 1994.

[4] M. Breton, A. Alg, and A. Haurie. Sequential Stackelberg equilibria in two-person games. *Optimization Theory and Applications*, 59(1):71–97, 1988.

[5] V. Conitzer and T. Sandholm. Computing the optimal strategy to commit to. In *ACM EC-06*, pages 82–90, 2006.

[6] G. B. Dantzig and P. Wolfe. Decomposition principle for linear programs. In *Operations Research*, volume 8, pages 101–111, 1960.

[7] N. Gatti. Game theoretical insights in strategic patrolling: Model and algorithm in normal-form. In *ECAI-08*, pages 403–407, 2008.

[8] E. Halvorson, V. Conitzer, and R. Parr. Multi-step multi-sensor hider-seeker games. In *IJCAI*, pages 336–341, 2009.

[9] C. Kiekintveld, M. Jain, J. Tsai, J. Pita, M. Tambe, and F. Ordonez. Computing optimal randomized resource allocations for massive security games. In *AAMAS*, pages 689–696, 2009.

[10] D. Korzhyk, V. Conitzer, and R. Parr. Complexity of computing optimal stackelberg strategies in security resource allocation games. *Technical Report, 2010.* Department of Computer Science, Duke University. http://www.cs.duke.edu/∼dima/security_games_bvn.pdf.

[11] G. Leitmann. On generalized Stackelberg strategies. *Optimization Theory and Applications*, 26(4):637–643, 1978.

[12] H. McMahan, G. Gordon, and A. Blum. Planning in the presence of cost functions controlled by an adversary. In *International Conference on Machine Learning (ICML)*, pages 97–104, 2003.

[13] P. Paruchuri, J. P. Pearce, J. Marecki, M. Tambe, F. Ordonez, and S. Kraus. Playing games with security: An efficient exact algorithm for Bayesian Stackelberg games. In *AAMAS-08*, pages 895–902, 2008.

[14] J. Pita, H. Bellamane, M. Jain, C. Kiekintveld, J. Tsai, F. Ordóñez, and M. Tambe. Security applications: Lessons of real-world deployment. In *SIGECOM Issue 8.2*, December 2009.

[15] J. Pita, M. Jain, C. Western, C. Portway, M. Tambe, F. Ordonez, S. Kraus, and P. Parachuri. Deployed ARMOR protection: The application of a game-theoretic model for security at the Los Angeles International Airport. In *AAMAS-08 (Industry Track)*, pages 125–132, 2008.

[16] J. Tsai, S. Rathi, C. Kiekintveld, F. Ordóñez, and M. Tambe. IRIS a tool for strategic security allocation in transportation networks. In *AAMAS-09 (Industry Track)*, 2009.

[17] B. von Stengel and S. Zamir. Leadership with commitment to mixed strategies. Technical Report LSE-CDAM-2004-01, CDAM Research Report, 2004.