Efficiently Solving Time-Dependent Joint Activities in Security Games

Eric Shieh, Manish Jain, Albert Xin Jiang, Milind Tambe

University of Southern California,
{eshieh, manishja, jiangx, tambe}@usc.edu

Abstract. Despite recent successful real-world deployments of Stackelberg Security Games (SSGs), scale-up remains a fundamental challenge in this field. The latest techniques do not scale-up to domains where multiple defenders must coordinate time-dependent joint activities. To address this challenge, this paper presents two branch-and-price algorithms for solving SSGs, SMART$_O$ and SMART$_H$, with three novel features: (i) a column-generation approach that uses an ordered network of nodes (determined by solving the traveling salesman problem) to generate individual defender strategies; (ii) exploitation of iterative reward shaping of multiple coordinating defender units to generate coordinated strategies; (iii) generation of tighter upper-bounds for pruning by solving security games that only abide by key scheduling constraints. We provide extensive experimental results and formal analyses.

1 Introduction

Stackelberg Security Games (SSGs) have been widely applied to real-world security domains with these applications depending on significant advances in fast algorithms for SSGs [1, 2]. Yet, scale-up remains a significant issue in advancing the scope of SSGs.

A major drawback of the current algorithms [3, 1, 4, 5] is their failure to scale-up in solving games that include joint coordinated activities, which require coordination in time and/or space, and provide the defender additional benefits. Yet such coordination is an important aspect of real-world security systems. For example, the PROTECT application for the US Coast Guard computes patrol routes for protecting ports in the US [6]; but this application only focuses on one-boat patrols. Yet there are benefits that may accrue from coordination across multiple boats, e.g., if a target is visited by a single patrol boat, it may only be 50% effective in detecting (and hence stopping) a potential attack. The arrival of a second boat may increase the effectiveness to 80% (as the attacker may be forced to react to a second boat). Yet no SSG algorithm today can scale-up to handle US Coast Guard coordinated patrols for small or large-scale ports. Even though such coordination is a critical aspect of other security domains as well, e.g., airports and border patrols, it has been addressed due to its complexity.

To remedy the challenge of scale-up, this paper makes the following contributions. First, it presents 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. Second, the paper puts forward an optimal algorithm, SMART$_O$, that computes optimal defender strategies for SMART problems. Third, the paper discusses a heuristic iterative procedure, SMART$_H$, that achieves further speed-up over SMART$_O$. Both algorithms use a branch-and-price algorithm [7] to deal with the large strategy space of the domain. Furthermore, these algorithms are able to exploit the structure of the joint activity coordination problem to gain speed up based on the following key ideas: (i) use of insights from the Traveling Salesman Problem to order the search space, especially in SMART$_H$, while maintaining coordination, (ii) efficient greedy computation of patrols per resource via iterative reward shaping to generate a joint patrol, and (iii) generation of tight upper-bounds exploiting scheduling constraints to allow pruning of the search space based on the submodular property of joint activities. The resulting speed-ups enable us to solve real-world problems of coordinated joint activities, e.g., the ports of Boston and NY.

2 SMART

A SMART problem instance is defined on a graph $G_r = (T, E_r)$, where the vertices T are the targets and the edges E_r represent connectivity between the targets for resource r. This allows for heterogeneous resources, e.g., boats or helicopters, which have the same targets but the connectivity between nodes can be different. For each $e \in E_r$, $\tau(e)$ represents the time it takes one defender resource to traverse the edge e. As usual with SSGs [8], for each target $t \in T$, there is an associated reward $U_d^c(t)$ and penalty $U_d^u(t)$ to the defender if t was protected with an *effectiveness* of 100% and 0% respectively. Similarly, payoffs $U_a^c(t)$ and $U_a^u(t)$ are defined for the attacker, with $U_d^u(t) < U_d^c(t)$ and $U_a^c(t) < U_a^u(t)$. The defender has a set of R resources, and each resource can choose an activity from the set $\mathcal{A} = \{\alpha_1, \alpha_2, \dots \alpha_K\}$. The notation used in SMART is described in Table 1.

The attacker's pure strategy space is the set of all targets, T. Each pure strategy of the defender is a route for each resource, such that the route starts and ends at a predefined home base $t_b \in T$, and the total route length of each individual patrol is upper bounded by Γ_r. The pure strategy X_i of *one* defender resource is a patrol route, which is represented as an ordered list of 3-tuples $X_i = [X_i^1, \dots, X_i^j, \dots]$. Here, each 3-tuple $X_i^j = (t, \alpha, \gamma)$ represents that this defender resource conducts and completes activity α at target t at time γ, where different activities require different amounts of time and effectiveness. Each pure strategy is restricted to begin and end at the home base, i.e. $X_i^1.t = t_b$ and $X_i^{|X_i|}.t = t_b$. Each route length is upper bounded by Γ_r, as follows:

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

$\mathcal{X}^r$ 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., $\prod_{r=1}^{R} \{\mathcal{X}^r\}$.

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
$U_d^c(t)/U_a^c(t)$	Defender/Attacker payoff for 100% protection of t
$U_d^u(t)/U_a^u(t)$	Defender/Attacker payoff for 0% protection of t
$\mathcal{A} : \{\alpha_1, \alpha_2, \dots \alpha_K\}$	Set of security activities
$\tau(\alpha)$	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
$\omega_t(\mathbf{P}_i)$	Effective coverage of t in $\mathbf{P}_i$
Γ_r	Maximum time of a patrol
W	Time window for a joint activity

Table 1. Notation Table

SMART is unique in that 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 its pure strategy if there exists at least two tuples $(t_i, \alpha_i, \gamma_i)$ and $(t_j, \alpha_j, \gamma_j)$ in the defender's pure strategy such that $t_i = t_j$ and $|\gamma_i - \gamma_j| \leq W$, where W is the time window for two activities on the same target.

For each activity α_i, $\texttt{eff}(\alpha_i)$ represents the effectiveness of the activity α_i. This effectiveness ranges from 0% to 100%, and measures the probability of the defender successfully preventing an attack on target t if the attack on t happened when the defender was conducting activity α_i at t. This is similar to what was done in PROTECT [6]. We define the effectiveness of the joint activity $\langle \alpha_i, \alpha_j \rangle$ as $\texttt{eff}(\alpha_i, \alpha_j)$. We assume that 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. $\texttt{eff}(S)$ represents the maximum effectiveness of an individual or a joint activity over a set S of activities performed at a target. $\texttt{eff}()$ is *submodular* if for all $S_1 \subseteq S_2$:

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

This means that each additional activity performed has diminishing gains in effectiveness. As we will see later in the paper, when this property holds we are able to prove nice theoretical properties of our algorithms.

The expected utility $U_d(\mathbf{P}_i, t)$ of the defender when the defender is conducting pure strategy $\mathbf{P}_i$, which is a single pure strategy for multiple defender resources, and the attacker attacks 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. For the purposes of this paper we assume that the time it takes to attack is longer than the time required to patrol, thus the attacker only cares about the maximum effective activity or joint activity. 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).

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.

3 SMART$_O$: Optimal Branch-and-Price Solver

SMART$_O$ computes an optimal solution of the SMART problem by building upon work that has leveraged the *branch-and-price* framework [1]. The two major novelties of SMART$_O$ over previous work are the formulation of the slave component to handle joint activities (in Section 3.1) and a better bounding component (in Section 3.2).

3.1 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 to handle joint activities, 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.

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

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

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

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

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

Master: The master LP given in Equations (6) to (10) solves for the optimal defender mixed strategy $\mathbf{x}$ over a given set of pure strategies J, given that the pure strategy of the attacker is set to t' (determined by the leaf node). This is similar in formulation to the ERASER algorithm [9]. $U_d(t, \mathbf{c})$ 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. 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 to be

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

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 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. The best column is identified using a mixed-integer linear program (MILP) formulation over the transition graph as defined below, *which captures all the spatio-temporal constraints of the problem in 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). The transition graph can have up to $|T| \times \Gamma_r$ nodes. 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, $\gamma_{\mathtt{v}} = \gamma_{\mathtt{u}} + \tau(t_{\mathtt{u}}, t_{\mathtt{v}}) + \tau(\alpha)$ 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 a virtual `source` and `sink` node that contain edges to/from the base target t_b to ensure that patrols start and end at t_b.

Example: Figure 1 shows a sample transition graph. Here, $t_b = t_1$ and the source has three edges, one for each activity α_1 – α_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 1.

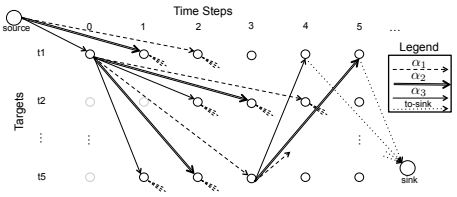


Fig. 1. An Example for the Transition Graph

Slave Problem MILP: This novel MILP component of SMART$_O$ solves for joint activities and generates the optimal defender pure strategy as given in Equation (12) to (16).

$$\min \quad \sum_{t \in T} y_t \cdot \max\{g_t, h_t\} \tag{12}$$

$$\sum_{e \in \mathtt{out}(\mathtt{u})} f(e_r) = \sum_{e \in \mathtt{in}(\mathtt{u})} f(e_r) \qquad \forall \mathtt{u}, r \tag{13}$$

$$g_t = \max_{e_r \in \mathtt{IA}(t), \forall r} \{f(e_r) \cdot \mathtt{eff}(e.\alpha)\} \tag{14}$$

$$h_t = \max_{e_i, e_j \in \mathtt{JA}(r_i, r_j, t), \forall i,j \in R} \{(f(e_i) + f(e_j) - 1) \cdot \mathtt{eff}(e_i.\alpha, e_j.\alpha)\} \tag{15}$$

$$f(e_r) \in \{0, 1\} \quad \forall e_r, \qquad g_t, h_t \in [0, 1] \quad \forall t \in T \tag{16}$$

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 (12), (14), and (15)) in accordance with our assumption of the attacker's decision making.

Here, the set $\mathtt{IA}(d)$ 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:

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

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

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

The result from the slave MILP is a set of 0-1 integer flows for each defender resource r. From the flows, the defender pure strategy $\mathbf{P}_j$ is computed, and the effective coverage $\boldsymbol{\omega}(\mathbf{P}_j)$ is then calculated and returned to the master.

3.2 Branch-and-bound component

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. To compute tight upper bounds, we present ORIGAMIP, a novel modification of ORIGAMI [9] specifically designed to generate tighter upper bounds for SMART problem instances by exploiting the structure of the domain.

$$\min_{\mathbf{c},\mathbf{f(e)}} k \tag{17}$$

$$0 \leq k - U_a(t, \mathbf{c}) \leq (1 - q_t) \cdot M \quad \forall t \in T \tag{18}$$

$$\sum_{e \in \mathtt{out(source)}} f(e) = R, \quad \sum_{e \in \mathtt{in(sink)}} f(e) = R \tag{19}$$

$$\sum_{e \in \mathtt{out(u)}} f(e) = \sum_{e \in \mathtt{in(u)}} f(e) \quad \forall \mathtt{u} \tag{20}$$

$$c_t \leq \sum_{e=\mathtt{(u,v)} \mid \mathtt{v}.t=t} f(e) \cdot \mathtt{eff}(\alpha_k) \quad \forall t \in T \tag{21}$$

$$c_t \in [0, 1] \;\; \forall t \in T, \quad f(e) \in [0, R] \;\; \forall e \in E \tag{22}$$

ORIGAMIP uses the transition graph defined in the slave formulation (Section 3.1). Equations (17)–(18) minimize the attacker's maximum expected utility, with $U_a(t, \mathbf{c})$ representing the attacker's utility given the defender's effective marginal coverage is $\mathbf{c}$ and the attacker attacks t. Equations (19)–(20) define the flows of the edges and enforce the flow conservation property. Equation (21) limits the coverage of the defender based on the amount of flow through the edges and the respective activity.

ORIGAMIP estimates the effectiveness of a defender patrol on a target as being the addition of the effectiveness of all individual activities on a target. This is an overestimate of the effectiveness (thereby providing an upper bound on defender utility) if the effectiveness function $\mathtt{eff}$ is sub-additive, i.e., $\mathtt{eff}(\alpha_i) + \mathtt{eff}(\alpha_j) \geq \mathtt{eff}(\alpha_i, \alpha_j)$, which follows from the submodularity property in (2).

Proposition 1. ORIGAMIP *computes valid upper bounds if* $\mathtt{eff}$*() is submodular.*

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.

4 SMART$_H$: Further scaling up SMART

SMART$_O$ fails to scale beyond 4 targets in our computational experiments; thus we present SMART$_H$, a heuristic approach for SMART that achieves further scale-up. It uses the previously discussed branch-and-price framework, but the slave is now solved using a novel heuristic formulation, which is built on two intuitions related to coordination: (i) Joint patrols can be computed by considering individual patrols iteratively, by *shaping the reward function* between iterations to reflect the additive benefit of joint activities. (ii) 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$_H$ slave only needs to solve a set of *linear programs* (as opposed to solving a MILP in SMART$_O$'s slave).

4.1 Reward Shaping

The slave in SMART$_H$ 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. Reward shaping has been used in other reinforcement learning contexts [10]; here we leverage this idea for coordination among multiple resources.

Algorithm 1 SMART$_H$ Greedy Algorithm

```
1: Input: y, 𝒢
2: Initialize P_j, μ
3: for all r_i ∈ R do
4:     X_i ← SolveSinglePatrol(y, μ, 𝒢_r)
5:     P_j ← P_j ∪ X_i
6:     μ ← ComputeCostCoef(P_j, 𝒢_r)
7: ω(P_j) ← ConvertToColumn(P_j)
8: return P_j, ω(P_j)
```

SMART$_H$ 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 3–5. Line 4 computes the best individual patrol X_r for the defender resource r (described in Section 4.2). 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.

How close to optimal is the solution of the greedy algorithm? [11] 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 $\bar{c}_j$. This is equivalent to maximizing:

$$F(\mathbf{P}_j) = -\sum_{t \in T} \omega_t(\mathbf{P}_j) \cdot y_t \tag{23}$$

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 the solution $\mathbf{P}_j$ of the greedy algorithm satisfies

$$F(\mathbf{P}_j) \geq (1 - 1/e) F(\mathbf{P}_*) \tag{24}$$

For the relaxed constraint where the time window, W, is greater than or equal to the maximum patrol time, Γ,[1] we show that $F(\mathbf{P}_j)$ is submodular. $F(\mathbf{P}_j)$ is submodular if P_1 and P_2 are two pure strategies where $P_1 \subseteq P_2$ and $F(P_1 \cup \{X\}) - F(P_1) \geq F(P_2 \cup \{X\}) - F(P_2)$. We show $F(\mathbf{P}_j)$ is submodular by showing that $\omega_t(\mathbf{P}_j)$ is submodular since $F(\mathbf{P}_j)$ is defined by a non-negative linear combination of $\omega_t(\mathbf{P}_j)$.

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

Proof (Proof Sketch).

Because $W \geq \Gamma$, $\omega_t(\mathbf{P}_j) = \texttt{eff}(S_{\mathbf{P}_j})$, where $S_{\mathbf{P}_j}$ is the set of activities of $\mathbf{P}_j$ on target t. Together with (2), this directly implies that $\omega_t(P_1 \cup X) - \omega_t(P_1) \geq \omega_t(P_2 \cup X) - \omega_t(P_2), P_1 \subseteq P_2$.

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 $\texttt{eff}()$ is submodular.*

Proof (Proof Sketch). 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 $\{X_2\}$. For each target t, we show that $\omega_t(\{X\}) \geq \omega_t(\{X_2, X\}) - \omega_t(\{X_2\})$ based on the submodularity property of $\texttt{eff}()$ in (2).

Qualifying this result for $W < \Gamma$ for 2 resources *is important since this setup is used most frequently in the real-world, e.g., 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.

4.2 TSP Ordering with Transition Graph

To achieve the approximation bound (24), we need to optimally compute an individual patrol X_r for the defender resource r in Line 4 of Algorithm 1. This can be solved by an MILP of similar form to the slave MILP (Equations (12)-(16)), 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 5).

Instead, we present a heuristic approach that achieves better scale-up by exploiting the spatial structure of the domain, and is provably optimal in certain cases. Our approach is based on the following restricted version of the problem: we define an *ordering* of the targets and restrict the patrols' sequence of target visits to be increasing in this order. We construct the *ordered transition graph* in the same way as described in Section 3.1; however, now, an edge from node $\texttt{u}$ to $\texttt{v}$ is added *only if* target $\texttt{u}.t$ appears before target $\texttt{v}.t$ in the ordering. If there does not exist a direct edge from $\texttt{u}$ to $\texttt{v}$, an edge

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

is added between these nodes where the distance is that of the shortest path. 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. The resulting problem is equivalent to a min-cost flow, which has integer extreme points that allow us to drop the integrality constraint (16), since a feasible solution of the resulting LP is guaranteed to be an integer flow. 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, SMART$_H$ uses an ordering based on the solution of the traveling salesman problem (TSP) for the input graph with all targets $G = (T, E)$. We show that under certain conditions, using the TSP ordering results in an optimal solution of the single-patrol problem. We look at a tree structure because various domains in the real-world, e.g., ports, contain an graph similar to a tree.

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* SMART$_H$ *computes a patrol for a single unit that optimizes the slave objective.*

Proof (Proof Sketch). 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 G_P, which is the subgraph of G restricted to T_P. 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' 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. Since the time window 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.

5 Experimental Results

We cannot compare with previous algorithms [3, 1, 4, 5] due to the inability of these algorithms to scale-up to the combinatorics unleashed by joint activities. This problem

is so complex that even human schedulers are unable to generate schedules. Therefore, we are limited to comparing different versions of SMART$_H$ and SMART$_O$.

The experiments were run on 100 game instances generated with random payoffs in the range [-10,10] and two defender resources unless otherwise noted. All experiments were run on graphs resembling ports: the graphs were constructed beginning with a tree that spans the targets and then adding 10 random edges between nodes. The time window was 30 minutes with 3 possible activities for the defender, taking 0, 5, or 15 minutes. The time discretization was 5 minutes, with the exception of the tests comparing SMART$_O$ to SMART$_H$, where the time discretization was 15 minutes. All experiments were run on a machine with a Dual core 2.0 GHz processor and 4 GB of RAM.

Figure 2 shows the runtime of SMART$_H$ versus SMART$_O$. The x-axis is the number of targets and the y-axis is the runtime. SMART$_O$ was only able to run for 4 or less targets. For 20 targets, SMART$_H$ takes less than a minute to run. This shows the difficulty of solving SMART instances and the importance of finding efficient solution techniques.

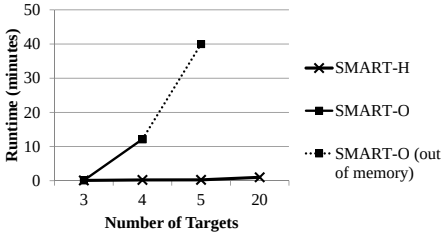
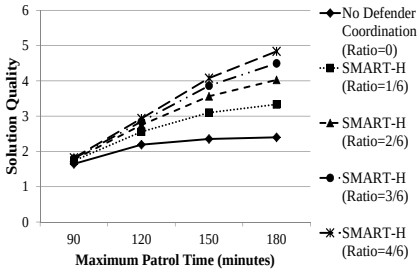
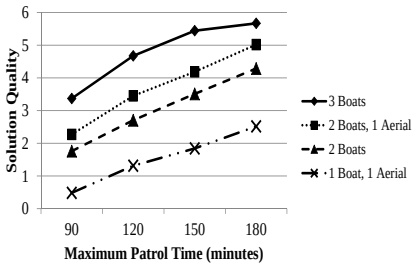


Fig. 2. Evaluating Runtime of SMART$_H$ vs SMART$_O$

Table 2 shows the solution quality, or defender expected utility, of SMART$_H$ versus SMART$_O$. For 3 targets, SMART$_H$ provided the same exact solution quality as SMART$_O$ for all game instances. For 4 targets, the average solution quality for SMART$_H$ was 0.0205 lower than SMART$_O$ with there being only one game instance where SMART$_H$ computed a lower defender expected utility than SMART$_O$ (difference of 0.679). This shows that on average SMART$_H$ computes a final result that is very close to SMART$_O$. However, SMART$_O$ is unable to run on bigger game instances.



(a) Solution quality of SMART$_H$ versus algorithm with no joint activities



(b) Heterogeneous defender resources (SMART$_H$)

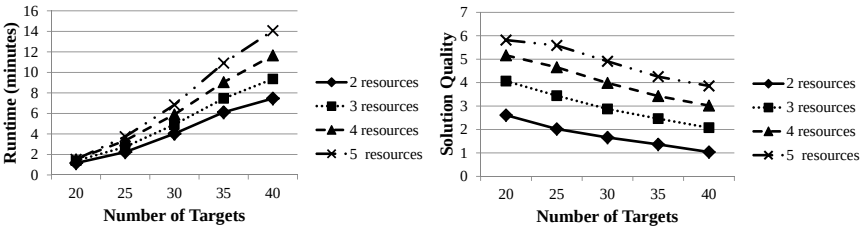
Fig. 3. Evaluating solution quality compared to previous algorithms and heterogeneous resources

Figure 3(a) shows that as we increase the importance of the coordinated joint activity, as measured by the ratio of effectiveness of joint activities (discussed below), the defender achieves a higher expected reward. The y-axis shows the solution quality and the x-axis denotes the maximum patrol time. This ratio is computed as shown with α_{max} equivalent to the highest quality activity: $\frac{\texttt{eff}(\alpha_{max},\alpha_{max})-\texttt{eff}(\alpha_{max})}{\texttt{eff}(\alpha_{max})}$. As 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. Without coordination, once the higher valued targets are covered with the maximum effective activity, additional time does not provide more effectiveness for those targets.

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

Table 2. Solution Quality of SMART$_H$ vs. SMART$_O$

Figure 3(b) shows that SMART$_H$ can handle heterogeneous defender resources (a boat and an aerial type) for 10 targets. The aerial resource is different than the boat in the following ways: (1) shorter transit times; (2) shorter patrol time; (3) lower effectiveness values. The x-axis is the maximum patrol time for a boat resource and the y-axis is the solution quality. In this experiment, the aerial resource has a patrol time and effectiveness value that is 25% of the boat resource's time/value. SMART$_H$ was able to run these instances.

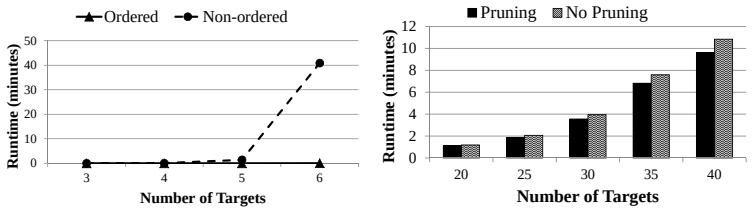


(a) Runtime with multiple resources (b) Solution quality with multiple resources

Fig. 4. Evaluating runtime and solution quality of SMART$_H$ for homogeneous resources

The ability of the SMART$_H$ algorithm to handle multiple homogeneous defender resources is shown in Figure 4(a) and 4(b), with the number of defender resources (boats) varying from 2 to 5. We assume that the defender will only get a joint effectiveness when 2 resources are coordinated; if 3 or more resources cover a target there is no additional joint effectiveness. In Figure 4(a), the x-axis is the number of targets and the y-axis is the runtime. Each line represents the runtime for a different number of defender resources. As the number of defender resources increases, the runtime increases. However, the increase in runtime due to the increase of defender resources is relatively minor, e.g., when there are 40 targets, the difference in increasing the number of resources from 2 to 3 only results in a two minute increase in runtime while there is still a

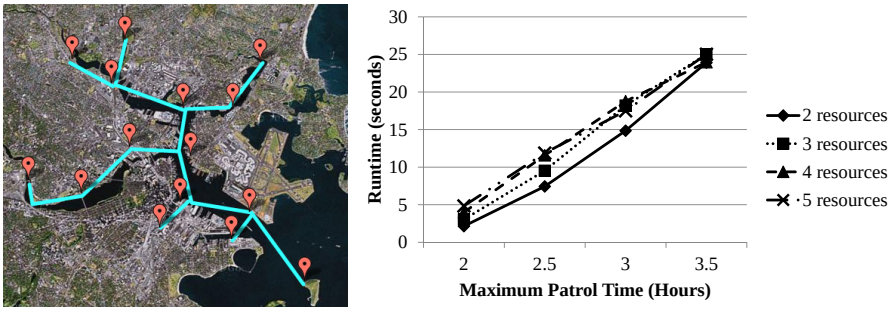
benefit to additional resources. This shows the scaleability of SMART$_H$ to handle more than two defender resources.



(a) Using ordered nodes (b) Using pruning

Fig. 5. Runtime improvements for SMART$_H$

Next, the different components of SMART$_H$ are examined to show the impact of node ordering and pruning. In Figure 5(a), the x-axis is the number of targets and the y-axis is the runtime. This figure shows the improvement of SMART$_H$ in using TSP ordering of the nodes (Section 4.2). When using a MILP (no TSP ordering), the runtime for 6 targets is over 40 minutes while the runtime for using TSP ordering is less than a second. For 7 targets, the algorithm using a MILP runs out of memory. In Figure 5(b), the x-axis is the number of targets and the y-axis is the runtime. This figure shows the benefit of ORIGAMIP in generating tight upper bounds to prune the branch-and-price tree (Section 3.2). As the number of targets increases, the amount of time saved by pruning increases.



(a) Sample Boston map (b) Runtime graph

Fig. 6. Example of Boston scenario

Real World Motivated Example: In addition to running simulations of SMART, we took the port of Boston as an example scenario that included 15 total targets as shown in Figure 6(a). We conducted simulations using SMART$_H$ on this graph varying the number of defender resources and maximum patrol time with the runtimes depicted in Figure 6(b). In this figure, the x-axis is the maximum patrol time and the y-axis is the

runtime in seconds. This shows that we were able to solve the Boston scenario in under 30 seconds.

6 Conclusion and Related Work

Addressing joint coordinated activities is the next frontier of the SSG research area. Such coordination is critical for real-world applications, and yet beyond the capabilities of both today's best algorithms to solve SSGs and human schedulers.

To address this challenge, this paper presented two novel algorithms, SMART$_O$ and SMART$_H$, that leveraged the properties and structure of joint activities to provide significant speed-ups to solve SMART problem instances. We provide proofs of theoretical properties of our algorithms while also showing the improved performance of the key components.

In terms of other related work, joint planning among agents has been studied in the planning community with models such as DEC-MDPs [12, 13] but do not address adversarial agents. In terms of related work within SSGs, we have discussed limitations of related algorithms throughout this paper. Studies on multiagent patrols beyond SSGs have focused on frequency-based patrols and adversarial environments [14, 15], including patrols in marine environments that take into account uncertain environmental conditions [16]. These studies aim to minimize the time lag between visits and do not consider targets of varying importance nor the impact of *joint activities*.

References

1. Jain, M., Kardes, E., Kiekintveld, C., Ordóñez, F., Tambe, M.: Security Games with Arbitrary Schedules: A Branch and Price Approach. In: AAAI. (2010) 792–797
2. Tambe, M.: Security and Game Theory: Algorithms, Deployed Systems, Lessons Learned. Cambridge University Press (2011)
3. Conitzer, V., Sandholm, T.: Computing the optimal strategy to commit to. In: ACM EC-06. (2006) 82–90
4. Paruchuri, P., Pearce, J.P., Marecki, J., Tambe, M., Ordóñez, F., Kraus, S.: Playing games with security: An efficient exact algorithm for Bayesian Stackelberg games. In: AAMAS-08. (2008) 895–902
5. Vanek, O., Jakob, M., Hrstka, O., Pechoucek, M.: Using multi-agent simulation to improve the security of maritime transit. In: MABS. (2011)
6. Shieh, E., An, B., Yang, R., Tambe, M., Baldwin, C., DiRenzo, J., Maule, B., Meyer, G.: PROTECT: A deployed game theoretic system to protect the ports of the united states. In: AAMAS. (2012)
7. Barnhart, C., Johnson, E., Nemhauser, G., Savelsbergh, M., Vance, P.: Branch and price: Column generation for solving huge integer programs. In: Operations Research. Volume 46. (1994) 316–329
8. Yin, Z., Korzhyk, D., Kiekintveld, C., Conitzer, V., Tambe, M.: Stackelberg vs. Nash in Security Games: Interch- angeability, Equivalence, and Uniqueness. In: AAMAS. (2010) 1139–1146
9. Kiekintveld, C., Jain, M., Tsai, J., Pita, J., Tambe, M., Ordóñez, F.: Computing optimal randomized resource allocations for massive security games. In: AAMAS. (2009) 689–696

10. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press (1998)
11. Nemhauser, G.L., Wolsey, L.A., Fisher, M.L.: An analysis of approximations for maximizing submodular set functions–I. Mathematical Programming **14**(1) (Dec 1978) 265–294
12. Becker, R., Zilberstein, S., Lesser, V.: Decentralized markov decision processes with event-driven interactions. In: AAMAS. (2004)
13. Bernstein, D., Zilberstein, S., Immerman, N.: The complexity of decentralized control of markov decision processes. In: Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence, Morgan Kaufmann Publishers Inc. (2000) 32–37
14. Agmon, N., Sadov, V., Kaminka, G.A., Kraus, S.: The Impact of Adversarial Knowledge on Adversarial Planning in Perimeter Patrol. In: AAMAS. Volume 1. (2008) 55–62
15. Machado, A., Ramalho, G., Zucker, J.D., Drogoul, A.: Multi-agent patrolling: An empirical analysis of alternative architectures. In: MABS. (2003) 155–170
16. Agmon, N., Urieli, D., Stone, P.: Multiagent patrol generalized to complex environmental conditions. In: AAAI. (2011) 1090–1095