Computing Minimax Strategy for Discretized Spatio-Temporal Zero-Sum Security Games *

Haifeng Xu[1], Fei Fang[1], Albert Xin Jiang[1],
Vincent Conitzer[2], Shaddin Dughmi[1], Milind Tambe[1]

[1] {haifengx,feifang,jiangx,shaddin,tambe}@usc.edu
University of Southern California, Los Angeles, CA 90007, USA
[2] conitzer@cs.duke.edu
Duke University, Durham, NC 27708, USA

Abstract. Among the many deployment areas of Stackelberg Security games, a major area involves games played out in space and time, which includes applications in multiple mobile defender resources protecting multiple mobile targets. Previous algorithms for such *spatio-temporal security games* fail to scale-up and little is known of the computational complexity properties of these problems. This paper provides a novel oracle-based algorithmic framework for a systematic study of different problem variants of computing optimal (minimax) strategies in spatio-temporal security games. Our framework enables efficient computation of a minimax strategy when the problem admits a polynomial-time oracle. Furthermore, for the cases in which efficient oracles are difficult to find, we propose approximations or prove hardness results.

Keywords: Security Games, Zero-Sum Games, Minimax Equilibrium, Oracle, Equilibria Computation.

1 Introduction

Among the multiple deployment areas of Stackelberg Security games [15, 16, 2, 3], a recent major application area involves games played out in space and time, which we refer to as *spatio-temporal security games*. This class of security games is particularly valuable for security of major transportation systems, where multiple mobile resources protect multiple mobile targets. For example, spatio-temporal security games are in use to generate patrol patterns for US Coast Guard patrol boats for the Staten Island ferries (mobile targets)—ferrying 60000 passengers a day, this system is considered a major terrorist target [6]. However, this is just one of many possible ferry systems around the world that require security. Other potential applications include protecting refugee aid convoys with overhead UAVs and protecting vessels from pirate activity [1].

Unfortunately, current algorithms for such spatio-temporal security games suffer from lack of scalability. For example, [1] provide a formulation with non-linear constraints that faced scaling problems with a single defender resource. [6] provide a linear

* A preliminary version of this work is under review for AAAI 2014. The paper might be withdrawn if there is a conflict.

program with better scalability properties for such games, but their formulation suffers from exponential slowdown with increasing number of defender resources; indeed it is seen to fail to scale up beyond three defender resources for 13 time steps. Additionally, little is known of the computational complexity properties of such spatio-temporal security game problems.

At a high level, the main challenge for scaling up spatio-temporal security games is their exponential growth of defender pure strategies. This exponential blow-up is due to two factors: first, the defender has multiple resources, and needs to pick a patrol schedule for each resource; second, each defender resource's set of patrol schedules grows exponentially in the number of time steps. As a result, the number of pure strategies is exponential in the number of resources and the number of time steps. Existing works in security games helps alleviate such exponential numbers of pure strategies via incremental strategy generation in security games [4, 17, 13] and use of compact marginal representations [9, 10, 8]. Unfortunately, these approaches fail to provide a systematic understanding of complexity properties of spatio-temporal security games or provide efficient algorithms that exploit the special structure of different variants of the game.

To address these challenges, we provide the following contributions: (i) We present the first systematic study of computational complexity of computing optimal (minimax) strategies in spatio-temporal security games. We consider several variants in the game setting; for the general setting, we provide an approximation algorithm. For several important restricted settings, we provide polynomial-time algorithms, while for another variant we give strong theoretical evidence that the problem is hard. (ii) Our experimental results based on a ferry-protection domain show that our algorithms scale-up significantly beyond what is achievable by [6]. For many of our theoretical results, we use an oracle-based algorithmic framework that reduces the minimax problem to a combinatorial optimization problem. An overarching theme in our solution techniques for the various settings is the exploitation of *spatio-temporal structure*, which allows us to formulate and solve these problems using graph-based techniques, often making use of additional geometric properties of the domain.

2 Settings and Notation

We study algorithms for scheduling resources in a discretized temporal and 1-D spatial domain (Figure 1) to protect weighted moving targets. They are motivated by the domain of ferry protection [6], where multiple mobile patrollers protect ferries carrying passengers. In the grid in Figure 1, the x-axis denotes a discretized temporal domain of $N + 1$ time points and the y-axis denotes a discretized 1-D spatial domain of $M + 1$ positions.

There are T moving targets. We use a pair (t, n) to denote a target t at time n. Let h_{tn} be the position (i.e., height) of the pair (t, n) (shown as stars in Figure 1). The targets need not land on the discretized positions, i.e., h_{tn} are not necessarily integers. The defender has K *homogeneous* (i.e., indistinguishable) resources. Resources can only land on the *discretized* positions and have maximum speed Δ (a constant). That is,

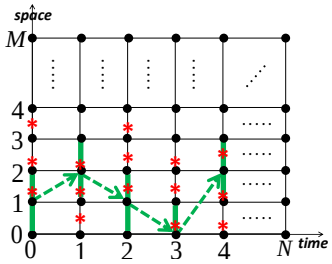


Fig. 1. Discretized Grid.

a move from m_n at n to m_{n+1} at $n+1$ is feasible if and only if [3] $|m_{n+1} - m_n| \leq \Delta$. Note that we do not require any assumption on the speed of the targets.

We use $[M]$, $[N]$, $[K]$, $[T]$ to denote the set of discretized spatial positions, set of discretized time points, set of resources and set of targets, respectively. A patrol schedule is simply a set consisting of positions that a resource would land on at each time. From now on, we call this a *patrol path*. We use a vector $v = (m_0, m_1, ..., m_N)$ to denote a patrol path in which the resource lands on position m_n at time n for any $n \in [N]$. We say path $v' = (m'_0, m'_1, ..., m'_N)$ is *weakly (strictly) under* path $v = (m_0, m_1, ..., m_N)$ if $m'_n \leq (<) m_n$ for all $n \in [N]$, and *weakly (strictly) above* is defined similarly. A patrol path is feasible if every individual move is feasible. It is easily observed that the number of feasible patrol paths is exponential in N, the number of time layers. A pure strategy for the defender consists of K feasible patrol paths, denoted as $\{v_k\}_{k\in[K]}$.

Resources have a protection radius, within which any target will be protected. We assume protection by multiple resources is equally efficient as protection by one resource. In Figure 1, the dashed arrow describes part of a patrol path and thickened segments along the spatial dimension denote the protected ranges. An attacker's pure strategy is a target-time pair (t, n), meaning that he attacks once, at time n, the target t. The Attacker's utility by attacking the pair (t, n) is its *weight* w_{tn} if it is not protected, and 0 otherwise. The weight of the same target can be different at different times, i.e., $w_{tn_1} \neq w_{tn_2}$ (for example, a ferry may not always carry the same number of people). We assume this is a zero-sum game and aim to compute the defender's minimax mixed strategy. *Our results are summarized in Table 1.* Some of our proofs are omitted in this paper due to the space limitations.

3 A New Algorithmic Framework

We provide a novel algorithmic framework to theoretically analyze the complexity of computing minimax strategy for the spatio-temporal security games introduced in the

[3] Here, we do not consider any acceleration limit. In a later section, we will further study the case with limited acceleration.

Table 1. Table of Solvability Status

CASES	SOLVABILITY
Constant Number of Resources (K)	$poly$ time (Fang et al. 2013)
Constant Number of Time Layers (N)	$poly$ time (Theorem 1)
Non-overlapping Protection Range	$poly$ time (Theorem 2)
Homogeneous Targets	$poly$ time (Theorem 3)
General	(1-1/e)-approx oracle (Observation 1)
General+Acceleration Limit	NP-hard oracle (Theorem 5)

previous section. This framework differs significantly from [6]; we provide an LP and reduce it to a combinatorial optimization problem, for which it is easier to analyze complexity results.

We first formalize the minimax strategy problem as an LP. Instead of $\{v_k\}_{k\in[K]}$, we use an alternative representation to denote pure patrol strategies. Specifically, let vector $e = (..., e_{tn}, ...) \in \{0,1\}^{TN}$ denote a pure strategy of the defender in the following way: given a pure strategy, $e_{tn} = 1$ if and only if this pure strategy protects the pair (t, n), $e_{tn} = 0$ otherwise. Let E denote the set of all pure strategies and $\mathcal{P}_w = ConvE$, the convex hull of set E, be the set of mixed strategies.

Notice that, $\mathcal{P}_w$ is also the set of marginal probabilities of protecting target-time pairs (t, n) that correspond to mixed strategies. The defender's optimal strategy in a zero-sum game can be formulated as the following LP (LP_g):

$$\begin{aligned} & \min\ u \\ s.t\ \ & x \in \mathcal{P}_w \\ & (1 - x_{tn})w_{tn} \le u, \forall\, t \in [T], n \in [N] \end{aligned}$$

Now we reduce LP_g to a combinatorial optimization problem by two steps of reductions. First, let the polyhedron $\mathcal{P}_g = \{(x, u) : x \in \mathcal{P}_w, (1 - x_{tn})w_{tn} \le u, \forall t, n\}$ denote the feasible set for LP_g, then LP_g can be solved in polynomial time with the ellipsoid method, as long as $\mathcal{P}_g$ admits an efficient separation oracle—that is, an algorithm that decides, for any (x, u), whether it is in $\mathcal{P}_g$ and returns a violated constraint if not. The following key lemma connects the polyhedron $\mathcal{P}_g$ and $\mathcal{P}_w$.

Lemma 1. *Separation oracles for $\mathcal{P}_w$ and $\mathcal{P}_g$ reduce to each other in* poly(T,N) *time.*

Proof. $\mathcal{P}_w \Rightarrow \mathcal{P}_g$: given a separation oracle $\mathcal{O}_w$ for $\mathcal{P}_w$, one can construct a separation oracle for $\mathcal{P}_g$ by simply checking the TN extra constraints $(1 - x_{tn})w_{tn} \le u$.

$\mathcal{P}_g \Rightarrow \mathcal{P}_w$: let $u_0 = \max_{t,n} w_{tn}$. For any $x_0 \in \mathbb{R}^{TN}$, $x_0 \in \mathcal{P}_w$ if and only if $(x_0, u_0) \in \mathcal{P}_g$. Furthermore, if $x_0 \notin P_w$, any hyperplane $a^T x + bu = c$ separating (x_0, u_0) from $\mathcal{P}_g$ gives a hyperplane $a^T x = c - bu_0$ separating x_0 from $\mathcal{P}_w$.

Unfortunately, $\mathcal{P}_w$ is defined by a set of constraints with exponential size. So our second step of reduction connects the oracle problem for $\mathcal{P}_w$ to another optimization problem.

Lemma 2. *The separation oracle problem for $\mathcal{P}_w$ reduces to the following LP (LP_w) in polynomial time:*

$$\max \sum_{t\in[T],n\in[N]} w_{tn}x_{tn}$$
$$s.t.\ \ x \in \mathcal{P}_w$$

for arbitrary weight profile $\{w_{tn}\}$.

As a result, LP_g reduces to LP_w in polynomial time.

Proof. We are following an argument from [7]; proof is provided for completeness.

Assume we are given an oracle $\mathcal{O}$ that computes LP_w. Consider another polyhedron $\mathcal{P}_w^\circ = \{y : y^T x \leq 1, \forall x \in \mathcal{P}_w\}$. For any y, we can decide whether $y \in \mathcal{P}_w^\circ$ by solving an instance of LP_w: $\max y^T x$, s.t. $x \in \mathcal{P}_w$. This can be computed by oracle $\mathcal{O}$ and output an optimal solution x^*. If $y^T x^* \leq 1$, then $y \in \mathcal{P}_w^\circ$, otherwise $y^T x^* = 1$ is a hyperplane separating y from $\mathcal{P}_w^\circ$.

Until now, we have constructed a separation oracle for $\mathcal{P}_w^\circ$ using $\mathcal{O}$. Therefore, we can maximize any linear function over $\mathcal{P}_w^\circ$ in polynomial time, again by ellipsoid method. Then, similar arguments as above implies that we can construct a separation oracle for $\mathcal{P}_w^{\circ\circ} = \{x : y^T x \leq 1, \forall y \in \mathcal{P}_w^\circ\} = \mathcal{P}_w$. This proves that the separation oracle problem for $\mathcal{P}_w$ reduces to LP_w.

Then we have, LP_g reduces by ellipsoid method to the separation oracle problem for $\mathcal{P}_g$, which again reduces to the separation oracle problem for $\mathcal{P}_w$ (by Lemma 1), which then reduces to LP_w.

LP_w picks a mixed patrol strategy to maximize the sum of weights it covers. So we call LP_w the *Weight collection problem*, while LP_g the *Game theoretic min-max problem*. A key observation here is that, as a linear program, LP_w has an optimal vertex solution on $\mathcal{P}_w$, which corresponds to a pure strategy that collects the maximum sum of weights. Therefore, LP_w can be thought of as the following *combinatorial problem*: given the weight w_{tn} and position h_{tn} for each pair (t, n), finding K feasible paths in the grid that covers the most weights. This is also the defender's best response problem for an arbitrary attacker mixed strategy, by regarding w_{tn} as the loss of the pair (t, n) under attacker's mixed strategy, i.e., the weight of pair (t, n) multiplied by the probability of attack at (t, n). As we will see later, this problem admits efficient combinatorial algorithms in some important special cases.

Before ending this section, we describe a technical lemma capturing the structure of the optimal vertex solutions of LP_w, which plays a key role in our latter arguments.

Lemma 3. *There exists an optimal vertex solution for LP_w corresponding to a defender pure strategy, say $\{v_k\}_{k\in[K]}$, in which v_k is* strictly *under v_{k-1} for all k.*

The proof of this lemma is not particularly insightful, so we omit it. The basic idea is to adjust a given optimal solution to another optimal solution satisfying the conditions in Lemma 3. The adjustments maintaining feasibility are intuitively explained in Figure 2.

Fig. 2. Two kinds of adjustment. Left: when two paths are crossing; Right: when two paths overlap.

4 When Any Parameter Is Constant

In this section, we show that when any of the parameters M, N, K, T is a constant, computing a minimax strategy admits polynomial-time algorithms. The interesting cases are $M > K$ and $T > K$, because the defender can use resources to cover all the discretized positions if $M \leq K$ and can dedicate a separate resource to follow each target if $T \leq K$ (assuming targets are not faster than resources). So, we only consider the cases where either K or N is a constant, because M or T being a constant would only be of interest when K is constant.

The LP formulation in [6] has size $O(NM^{2K})$, which is polynomial in M and N assuming K is constant. So we consider another case where the number of time layers N is constant. Specifically, we show the following.

Theorem 1. *There is a polynomial-time algorithm for LP_g when N is constant.*

Algorithm 1 Dynamic Programming for Weight Collection

Input: position h_{tn} and weight w_{tn}, $\forall\, t \in [T], n \in [N]$.
Output: optimal objective value and corresponding pure strategy.

1: State $OPT(v;k)$ denotes the maximum objective value using k resources when the highest patrol path is v; $S(v;k)$ denotes the corresponding optimal pure strategy.
2: For all feasible v, compute $OPT(v;1)$ equaling the total weight covered by v and let $S(v;1) = \{v\}$.
3: $\forall v$, let path set $\mathcal{P}(v) = \{u :\ u$ is strictly under v$\}$.
4: **for** k=2 to K **do**
5: **for** all feasible v **do**
6: Compute $OPT(v;k)$

$$= \max_{u \in \mathcal{P}(v)} \{OPT(u;k-1) + \mathcal{C}(v \setminus u)\}$$

 where $\mathcal{C}(v \setminus u)$ is the sum of weights covered by v but not by u.
7: $S(v;k) = S(u^*;k-1) \cup \{v\}$ where u^* is a path achieving "max" in Step 6.
8: **end for**
9: **end for**
10: Output $\max_v OPT(v;K)$ and corresponding $S(v^*;K)$.

Using our algorithmic framework, Lemma 2 and the following Lemma 4 together yield a proof of Theorem 1.

Lemma 4. *If N is a constant, Algorithm 1 runs in polynomial time and outputs an optimal vertex solution for LP_w, for any weight profile $\{w_{tn}\}$.*

Proof. Lemma 3 guarantees there is always an optimal pure strategy in which paths do not cross or touch. Algorithm 1 computes such an "ordered" optimal pure strategy using dynamic programming. This algorithm is polynomial-time because N is a constant, therefore the number of states $OPT(v; k)$ is *poly(M,K)*.

5 Solving Large-Scale Cases

For large-scale problems, we show that two important special cases admit polynomial-time algorithms, the general problem admits a (1-1/e)-approximation oracle and the oracle problem for a slightly extended version is NP-hard.

5.1 Non-overlapping Protection Range

In this section, we consider the special case where the protection ranges of distinct discretized points n_1 and n_2 never overlap. That is, the protection radius of a resource is at most $\frac{1}{2}d_\Delta$ where d_Δ is the distance between two spatial discretized points. We assume that targets are located close enough to the spatial discretized points such that each target can be covered by at least one grid point.

We show that for this case, there is a polynomial time algorithm by showing that we can obtain an optimal vertex solution for LP_w in polynomial time. Since the protection ranges do not overlap, for any point, we can say that the targets within its protection range "belong" to that point and the sum of the weights of targets within its protection range is the "reward" of covering that point. Now any pure strategy can be thought of as a flow over the grid points that collects rewards. However, a potentially problematic discrepancy in reducing this to a standard flow problem is that, if multiple patrol paths pass through the same grid point, LP_w counts the reward at that point only once, but a standard reward-collecting flow formulation would count the reward once for every unit of flow through the point. Fortunately, Lemma 3 guarantees that, to find an optimal pure strategy for LP_w, it suffices to find an optimal pure strategy under the additional constraint that paths do not overlap. We can do this by slightly modifying the grid and adding capacity constraints to the network flow formulation (Figure 3). Then, by the integrality of network flows, optimal solutions to the network flow problem constitute optimal pure strategies for LP_w.

Theorem 2. *If the protection ranges at different grid points do not overlap with each other, then an optimal vertex solution to LP_w can be found in polynomial time. Therefore, LP_g admits a polynomial-time algorithm.*

5.2 Homogeneous Targets

Oftentimes in practice targets are homogeneous, i.e., $\exists w$, such that $w_{tn} = w, \forall t, n$. Examples include defending cargo ships. For this case, we provide a polynomial-time algorithm to compute an optimal solution to LP_g directly.

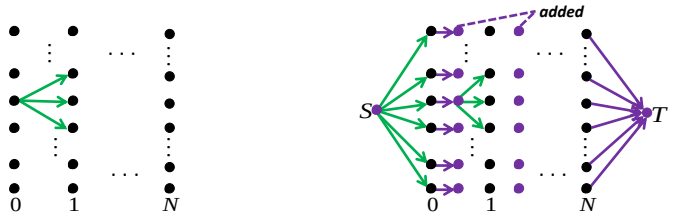


Fig. 3. Left: Original Grid; Right: Constructed Network in which all the edges have capacity 1. Only the added purple edges have non-zero weights, with the weight of such an edge equaling the sum of all the target weights covered by the corresponding grid point.

Note that if all the targets have the same weight, LP_g degenerates to the following form: $\max\ u$ satisfying $x \in \mathcal{P}_w$ and $x_{t,n} \geq u$. In other words, it seeks a probabilistic coverage of all the targets, such that the minimum probability over all targets is maximized. We relate this probabilistic coverage problem (PC) to the following deterministic coverage problem (DC): given the positions of all the targets at different times (i.e., h_{tn}), DC seeks to find the minimal number of resources such that they can cover all the targets surely at any time, i.e., with probability 1.

We show PC admits a polynomial-time algorithm by the following two steps: 1.) there is a "duality" relationship between PC and DC, and the optimal solution of PC can be recovered from that of DC efficiently (Theorem 3); 2.) DC admits a greedy polynomial-time algorithm (Algorithm 2).

Algorithm 2 Greedy Algorithm for Deterministic Coverage Problem (DC)

Input: the position of t at time n (h_{tn}), $\forall\, t \in [T], n \in [N]$;
Output: Optimal value K_0 and path set P.

1: Initialization: $K_0 = 0$, $P = \emptyset$.
2: **while** there are pairs (t, n) not covered **do**
3: $K_0 = K_0 + 1$; construct path v_{K_0} to be the time-wise lowest path that does not leave any pair (t, n) above its protection range uncovered;[4] add v_{K_0} to P.
4: **end while**

We use $OPT(PC_K)$ and $OPT(DC)$ to denote the optimal objective values of problem PC (with K resources) and problem DC, respectively. The following lemma plays a key role in our "duality" argument.

Lemma 5. *$OPT(PC_K) \geq \frac{K}{K_0}$ if $OPT(DC) = K_0$; and $OPT(DC) \leq \frac{K}{p}$ if $OPT(PC_K) = p$.*

[4] A straightforward construction is as follows: starting from a path $v = (m_1, ..., m_N)$ that is above any uncovered pair (t, n), we then set $m_n = m_n - 1$ whenever there is an n such that $v = (m_1, ..., m_n - 1, m_N)$ is feasible and there is no uncovered target above the protection range of $m_n - 1$.

Lemma 5 yields the following "duality" relation between PC and DC.

Theorem 3. *$OPT(PC_K) = K/OPT(DC)$. Furthermore, the optimal solution of PC_K can be generated from that of DC efficiently.*

Proof. The first part of Lemma 5 yields $OPT(PC_K)OPT(DC) \geq K$, while the second part yields $OPT(PC_K)OPT(DC) \leq K$. So $OPT(PC_K) = K/OPT(DC)$.

Next, we prove that a solution to PC can be obtained from a solution to DC as follows. Given the optimal path set $[K_0]$ for DC, we can sample a combination of K paths (i.e., a pure strategy for the defender) from $[K_0]$ uniformly at random. This can be easily done in $poly(K_0)$ time. Any target is covered by a resource with probability $C_{K_0-1}^{K-1} \times \frac{1}{C_{K_0}^K} = \frac{K}{K_0}$ where $C_{K_0}^K$ means K_0 choose K.

We now show that DC admits a polynomial-time algorithm (Algorithm 2). Clearly, Algorithm 2 runs in polynomial time and outputs K_0 feasible paths covering all targets. The following theorem guarantees the optimality of Algorithm 2.

Theorem 4. *The K_0 output by Algorithm 2 is optimal.*

5.3 General Case

In this section we consider the general problem. Our basic idea is still to follow the oracle-based algorithmic framework. The first observation is that finding an optimal vertex solution for LP_w in the general case is a submodular maximization problem with an exponential-sized universe set.

Specifically, let A denote the set of all the feasible patrol paths, so that A has exponential size. Define a non-negative function $w : 2^A \rightarrow R^+$ as follows: $\forall B \subseteq A$, $w(B)$ equals the sum of weights covered by all the paths in the subset B. It is easy to see that $w(B)$ is a non-negative monotone submodular function.

Our problem can be stated as maximizing $w(B)$ subject to the cardinality constraint $|B| = K$, which is NP-hard for many classes of submodular functions, e.g., for weighted coverage function [12]. Fortunately, a simple greedy algorithm for non-negative monotone submodular function maximization that achieves an $(1 - \frac{1}{e}) - approximation$ [11] applies to our problem with a bit of further analysis. The straightforward implementation of this greedy algorithm runs in $poly(|A|, K)$, in which $|A|$ is exponentially large in our case. We note that $|A|$ shows up in the complexity bound because, at each step, the straightforward implementation needs to enumerate all the elements in A to decide which element covers the most *additional* value if added at the current step. However, in our case, this element can be computed efficiently without enumerating all the elements in A. That is, we first set all the covered weights to 0 and then compute the path that covers the most weight in the current weight profile, which can be done easily, e.g., by a flow formulation.

Observation 1 *Algorithm 3 is an $(1 - \frac{1}{e})$-approximation for LP_w for arbitrary weight profile $\{w_{tn}\}$.*

A constant-factor approximation to LP_w theoretically does not imply the same constant-factor approximation to LP_g. However, as a heuristic method, column generation using Algorithm 3 as an approximate oracle performs very well in practice.

We describe the algorithm as follows (Algorithm 3) and summarize its performance in Observation 1.

Algorithm 3 Greedy Weight Coverage

Input: h_{tn} and w_{tn}, $\forall\, t \in [T], n \in [N]$;
Output: path set $\{u_k\}_{k\in[K]}$.

1: **for** $k = 1 : K$ **do**
2: compute the path, say u_k, that covers the most weight with respect to current weight profile.
3: Set $w_{tn} = 0$ if t is covered by u_k at n.
4: **end for**

5.4 General Case with Acceleration Limit

In this section, we show solving the oracle problem for LP_w is NP-hard in a slightly extended case, specifically, when resources have a limit on their acceleration. We first model acceleration as follows.

Definition 1. *(Acceleration) For any triple of positions (m_{n-1}, m_n, m_{n+1}) at 3 adjacent time layers $(n-1, n, n+1)$, define $A_n = |m_{n+1} + m_{n-1} - 2m_n|$ to be the* acceleration *at time n.*

Intuitively, one can regard $m_{n+1} - m_n$ as the speed within the time unit between n and $n+1$, so $(m_{n+1} - m_n) - (m_n - m_{n-1}) = m_{n+1} + m_{n-1} - 2m_n$ would be the speed change between two adjacent time units, and can be viewed as the acceleration at time n.

The speed limit naturally gives an upper bound on A_n, i.e., $A_n \leq 2\Delta$. We show that if resources have a slightly stricter limit on acceleration, namely any feasible move must satisfy $A_n \leq 2(\Delta - 1)$, then LP_w is NP-hard.

Lemma 6. *LP_w is NP-hard, if $A_n \leq 2(\Delta - 1)$ for any n.*

Proof. We prove this by reducing from vertex cover.

Construct the following special case: set $\Delta = 3$, set patrol radius $r = \frac{2}{3}d_\Delta$, i.e., the protection range is $(m - \frac{2}{3}d_\Delta, m + \frac{2}{3}d_\Delta)$ at position m. Here d_Δ is the distance between two neightboring spatial points. Therefore, the position $m + \frac{1}{2}d_\Delta$ can be covered by a resource either at m or at $m+1$, but the position m can only be covered by a resource at m.

First, the following vertex cover problem (VC) is known to be NP-hard: given integer K and any graph $G = (V, E)$, finding a subset $V_0 \subseteq V$ that maximizes the number of edges it covers subject to $|V_0| = K$. We reduce VC to LP_w.

Given K and any graph instance $G = (V, E)$, we construct an LP_w instance as follows. Create $|V|$ targets, each one corresponding to a node in V. The moving paths for these $|V|$ targets are constructed as path 1,2,3, etc. in the left of Figure 4. Generally, two neighboring paths differ by a horizontal translation of 4 time layers. These paths are constructed in such way that each pair of targets cross each other exactly once and all the crossings have the same local geometry as shown in the right of Figure 4. Any pair of targets does not occupy the same discretized position at any time. We set the weight of each target to be 1 at any time layer.

Now, for any edge in E, we add a "*tiny*" target with small enough weight. Specifically, $\forall e = (u, v) \in E$, let n_e denote the discretized time when path u and v are closest, and u_{n_e} and v_{n_e} denote the position of path u and v at time n_e.[5] We put a tiny target with weight ϵ at the time n_e on the point in the middle between u_{n_e} and v_{n_e} (see the position of ϵ in Figure 4). Note that a resource can capture this ϵ weight at either u_{n_e} or v_{n_e}. One may think of this as a tiny target e that has weight ϵ only at this specific time and has weight 0 otherwise. We set ϵ small enough to satisfy $\epsilon < \frac{1}{|E|}$, so that sum of all the tiny weights is still less than 1, i.e., $|E|\epsilon < 1$. This completes the construction of the LP_w instance.

Our first observation is that any optimal vertex solution to LP_w must first optimally capture the weights from the target set V, because $|E|\epsilon < 1$, meaning that the loss of failing to cover any target corresponding to a node in V cannot be recovered even by covering all the ϵ tiny targets.

Now we show the *only* way to collect weights optimally from those $|V|$ targets is to pick any K targets and follow their paths precisely. The reason is that K resources can capture at most K targets at any time layer because of limited protection radius and non-overlapping paths. So capturing a sum of weights of KN is the best one can do and following any K targets achieves this. The reason that these are the only optimal strategies is that the only time when a resource can switch to another path is when the current path is about to intersect that path, however the switch at that time is infeasible because it needs an acceleration of $2\Delta - 1$, greater than the limit $2(\Delta - 1)$.

As a result, maximizing the collected weights is equivalent to choosing a target set $[K] \subseteq V$ to follow, such that the number of covered tiny targets is maximized. Note that a tiny target $e = (u, v)$ can be covered if and only if one of its end points corresponds to a path in $[K]$. So, an optimal vertex solution (i.e., optimal pure strategy) to LP_w gives an optimal solution to the original VC problem instance. Since $G = (V, E)$ is arbitrarily chosen, LP_w is NP-hard.

Lemma 1 and Lemma 6 together yield the following theorem.

Theorem 5. *The separation oracle problem for LP_g is NP-hard, if $A_n \leq 2(\Delta - 1)$ for any n.*

Theorem 5 does not imply that solving LP_g is also NP-hard. Still, this rules out perhaps the most natural approach to showing that LP_g is easy to solve.

[5] In a slight abuse of notation, here we use u, v to denote both nodes in G and paths corresponding to these nodes.

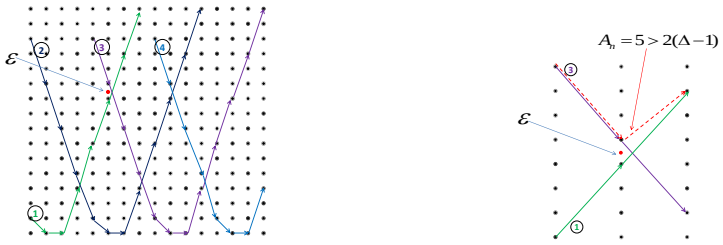


Fig. 4. Left: enumerated target paths; example position to put an ϵ tiny targets; Right: local intersection geometry, tiny target position and infeasible acceleration.

6 Experiments

We compare both *solution quality* and *time performance* of proposed algorithms in real data. All algorithms being tested are list as follows: (i) **LP**: linear programming formulation in [6]. (ii) **DP**: dynamic programming for LP_w (Algorithm 1). (iii) **NonOverlap**: network flow assuming non-overlapped protection range for LP_w. (iv) **Hom**: greedy algorithm assuming homogeneous targets (Algorithm 2). (v) **OrderGreedy**: greedy weight coverage algorithm for LP_w (Algorithm 3). The algorithms NonOverlap and Hom could be easily adopted as heuristic algorithms for the general case by pretending the protection ranges do not overlap or all the targets are homogeneous. Algorithms DP, NonOverlap, and OrderGreedy need to reduce from LP_g to LP_w by the ellipsoid method, which often suffers from numerical instability and poor performance in practice. We instead implemented these algorithms using column generation [5], which replaces the ellipsoid method (see online appendix for details). Although the number of iterations can be exponential in the worst case, this method is empirically efficient and thus is adopted here for testing the algorithms.

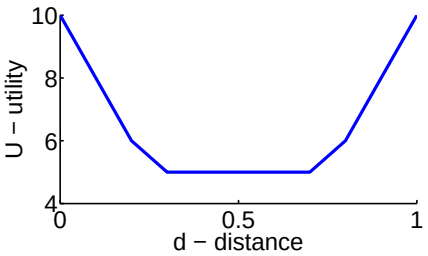
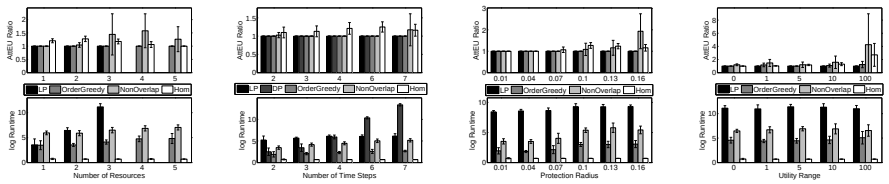


Fig. 5. Ferry utility

Experiments	K	T	N	M	r
Increase K	–	5	13	9	0.1
Increase N	3	5	–	5	0.14
Increase r	3	5	7	7	–
Increase Range	3	5	13	9	0.1
Large Scale	–	15	–	31	–

Fig. 6. Main Parameters

We test our algorithms in both practical settings in the ferry protection domain and randomly generated settings. Practical settings are generated based on domain descrip-



(a) Increase K (b) Increase N (c) Increase Re (d) Increase Range

Fig. 7. Experimental Results For Small Scale Practical Settings in Ferry Domain

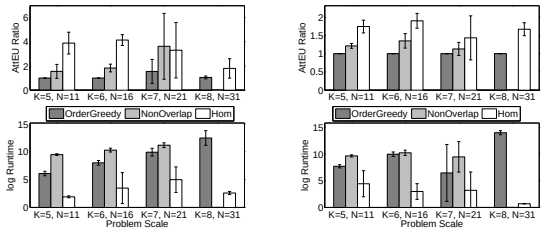
tion in [6]. Ferry's utility depends on its position between two terminals, and usually appears as a U-shape in practice (see Figure 5). For randomly generated settings, we randomly choose the moving path and utility of each target. Results are shown in Figure 7 and 8. In each figure, the y-axis of the upper plot shows the solution quality of different algorithms. The objective of LP_g is the attacker's maximum expected utility, denoted as AttEU. The defender aims to minimize AttEU, and thus a lower AttEU indicates a higher solution quality. For each instance, we calculate the ratio of AttEU of any algorithm to the best value among all tested algorithms. When the best value is 0, we add 0.001 to all values to get rid of the 0 denominator. The solution quality is the AttEU ratio averaged over 20 sampled instances. The y-axis of the lower plot shows the natural logarithm of runtime in milliseconds to make the comparison more clear. The minimum runtime is set to 1 millisecond.

Small scale experiments. We first focus on small scale data to evaluate the optimality of algorithms and their performance when the corresponding optimality assumptions are violated. All main parameters used are listed in Figure 6.

Figure 7(a) shows the performance of the baseline strategy (LP) as the number of resources (K) increases. LP is ensured to be optimal, however the runtime increases exponentially when K increases. When $K \geq 4$, LP runs out of memory and fails to return a solution. *So LP – the state of the art [6] – can only run if $K \leq 3$ and number of time steps is just 13*. Figure 7(b) shows that DP always achieves the optimal solution. When the number of time steps is small enough (e.g., $N \leq 3$), DP runs much faster than LP. As N increases, the advantage diminishes and can be even slower than the baseline algorithm when $N \geq 6$. So DP is especially useful for cases with small N. Figure 7(c) shows that NonOverlap achieves the optimal solution when the protection radius is small ($r < d_\Delta/2 = 1/(2(N-1)) = 0.083$) and it outperforms the baseline LP in runtime significantly. Even when the non-overlapping assumption is violated, this algorithm still provides a good approximation of the optimal solution, especially when the protection radius is close to $d_\Delta/2$. Figure 7(d) shows the performance of Hom as the utility range increases. Utility range is defined as the difference between the maximum and minimum utility of the targets. When utility range equals 0, all targets are homogeneous. From the figure, we know Hom runs orders of magnitude faster than the baseline LP. It obtains optimal solution when utility range is 0. As the utility range increases, the solution quality of Hom degrades but it still provides a reasonable approximation.

In all these experiments, we also tested the heuristic algorithm OrderGreedy. Surprisingly, it achieves optimal or near-optimal solution in most cases, while outperforming LP and DP significantly in runtime, which indicates it to be a good heuristic algorithm in many different settings. We also tested these algorithms on small scale randomly generated instances and the results are similar (see online appendix).

Large scale experiments. Figure 8 shows the performance of the heuristic algorithms for general case when the scale of the problem is large. The utility range is randomly chosen from $[0, 100]$ and the protection radius is randomly chosen from $[0, 0.05]$ ($d_{\Delta}/2 = 0.0167$). Figure 8(a) is based on practical settings in ferry domain and Figure 8(b) is based on randomly generated settings. It can be seen that different algorithms achieve best performance in different samples as none of the algorithms keep an AttEU ratio of 1. However, OrderGreedy achieves best solution quality in many cases, especially for practical settings. In terms of runtime, Hom is significantly faster than the other two algorithms and NonOverlap is the slowest. Notice that NonOverlap runs out of memory when the scale gets larger ($K = 8, N = 31$).



(a) Practical setting (b) Random settings

Fig. 8. Experimental Results For Large Scale Problems

7 Conclusions

This paper: (i) systematically studied computational complexity properties of spatio-temporal security games; (ii) proposed novel polynomial-time algorithms or proved approximations and hardness results for different variants of these game; (iii) examined all the proposed algorithms experimentally based on a real domain and showed significant improvements over previous best known algorithm for these games [6].

References

1. Bošanský, Branislav and Lisý, Viliam and Jakob, Michal and Pěchouček, Michal: Computing time-dependent policies for patrolling games with mobile targets. AAMAS 2011.
2. Eric Shieh and Bo An and Rong Yang and Milind Tambe and Craig Baldwin and Joseph DiRenzo and Ben Maule and Garrett Meyer: PROTECT: A Deployed Game Theoretic System to Protect the Ports of the United States. AAMAS 2012.

3. FZhengyu Yin and Albert Jiang and Matthew Johnson and Milind Tambe and Christopher Kiekintveld and Kevin Leyton-Brown and Tuomas Sandholm and John Sullivan: TRUSTS: Scheduling Randomized Patrols for Fare Inspection in Transit Systems. IAAI 2012.
4. Jain, Manish and Korzhyk, Dmytro and Vaněk, Ondřej and Conitzer, Vincent and Pěchouček, Michal and Tambe, Milind: A Double Oracle Algorithm for Zero-sum Security Games on Graphs. AAMAS 2011.
5. McMahan, H. Brendan and Gordon, Geoffrey J. and Blum, Avrim: Planning in the Presence of Cost Functions Controlled by an Adversary. ICML 2003.
6. Fei Fang and Albert Xin Jiang and Milind Tambe: Optimal Patrol Strategy for Protecting Moving Targets with Multiple Mobile Resources. AAMAS 2013.
7. Martin Grötschel and László Lovász and Alexander Schrijver: The ellipsoid method and its consequences in combinatorial optimization. Combinatorica, 291-295, 1984.
8. Letchford, Joshua and Conitzer, Vincent: Solving Security Games on Graphs via Marginal Probabilities. AAAI 2013.
9. Kiekintveld, Christopher and Jain, Manish and Tsai, Jason and Pita, James and Ordóñez, Fernando and Tambe, Milind: Computing optimal randomized resource allocations for massive security games. AAMAS 2009.
10. Dmytro Korzhyk and Vincent Conitzer and Ronald Parr: Complexity of computing optimal Stackelberg strategies in security resource allocation games. AAAI 2010.
11. Nemhauser, G. L. and Wolsey, L. A.: Best algorithms for approximating the maximum of a submodular set function. Math. Oper. Res., 177-188, 1978.
12. Feige, Uriel: A Threshold of Ln N for Approximating Set Cover. J. ACM, 634-652, 1998.
13. Rong Yang and Albert Xin Jiang and Milind Tambe and Fernando Ordonez: Scaling-up Security Games with Boundedly Rational Adversaries: A Cutting-plane Approach. IJCAI, 2013.
14. Desaulniers, Guy and Desrosiers, Jacques and Solomon, Marius M: Column generation. Springer, 2005.
15. Basilico, N. and Gatti, N. and Amigoni, F.: Leader-follower strategies for robotic patrolling in environments with arbitrary topologies. AAMAS 2009.
16. Letchford, J. and Vorobeychik, Y.: Computing randomized security strategies in networked domains. AARM Workshop In AAAI, 2011.
17. Bosansky, Branislav and Kiekintveld, Christopher and Lisy, Viliam and Cermak, Jiri and Pechoucek, Michal: Double-oracle Algorithm for Computing an Exact Nash Equilibrium in Zero-sum Extensive-form Games. AAMAS 2013.