The Thirty-Second AAAI Conference
on Artificial Intelligence (AAAI-18)

Catching Captain Jack: Efficient Time and Space Dependent Patrols to Combat Oil-Siphoning in International Waters

Xinrun Wang,[1] Bo An,[2] Martin Strobel,[3] Fookwai Kong[4]
[1,2,3]School of Computer Science and Engineering, Nanyang Technological University, Singapore
[4]DSO National Laboratories, Singapore
[1,2] {xwang033, boan}@ntu.edu.sg, [3, 4] {martin.r.strobel, dfookwaikong}@gmail.com

Abstract

Pirate syndicates capturing tankers to siphon oil, causing an estimated cost of $5 billion a year, has become a serious security issue for maritime traffic. In response to the threat, coast guards and navies deploy patrol boats to protect international oil trade. However, given the vast area of the sea and the highly time and space dependent behaviors of both players, it remains a significant challenge to find efficient ways to deploy patrol resources. In this paper, we address the research challenges and provide four key contributions. First, we construct a Stackelberg model of the oil-siphoning problem based on incident reports of actual attacks; Second, we propose a compact formulation and a constraint generation algorithm, which tackle the exponentially growth of the defender's and attacker's strategy spaces, respectively, to compute efficient strategies of security agencies; Third, to further improve the scalability, we propose an abstraction method, which exploits the intrinsic similarity of defender's strategy space, to solve extremely large-scale games; Finally, we evaluate our approaches through extensive simulations and a detailed case study with real ship traffic data. The results demonstrate that our approach achieves a dramatic improvement of scalability with modest influence on the solution quality and can scale up to realistic-sized problems.

Introduction

It was around midnight on the 4th of June 2015 when a group of pirates boarded the Malaysia-registered oil tanker Orkim Victory from a speed boat while under way to Kuantan port, Malaysia. After taking control of the ship, the pirates moved the Orkrim Victory to a different location where a second tanker pulled up alongside it. Within a couple of hours the pirates siphoned 770 metric tons of Automotive Diesel Oil and disappeared. This was the eighth of eleven piracy incidents occurring in the South China Sea (SCS) in 2015 (ReCAAP 2015a; 2015c). The estimated worldwide economic damage due to piracy reaches from $5 to $12 billion a year and a major part of the attacks occur in the SCS (Kemp 2015; Bowden et al. 2010). To combat the threat of piracy in the SCS, Singapore, Malaysia and Indonesia plan to establish patrols in the region (Xue 2015). Given the vast area of the SCS, the huge number of merchant ships passing through and the limited number of patrol boats, the question of how to efficiently deploy patrol resources is extremely challenging to the security agencies.

During the last decade, several scheduling and patrolling systems based on security games were developed and deployed in the real world, e.g., ARMOR used by the LA International Airport (Jain et al. 2010), IRIS supporting the U.S. Federal Air Marshal Service (Jain et al. 2010), and PROTECT used by the U.S. Coast Guard (An et al. 2012). The successful deployment has led to the creation of a new and active research area of security games and even subareas like green security games considering environmental issues (Fang et al. 2016; Yin and An 2016) and protecting large public events (Yin, An, and Jain 2014). However, these methods cannot be directly applied to our problem due to three main issues: i) our game is highly time and space dependent and both players take paths as their strategies while most previous works assume that at most one player takes paths (Basilico, Gatti, and Amigoni 2009; Fang, Stone, and Tambe 2015) or the values of targets are static over time (Zhang et al. 2017); ii) continuous externalities, e.g., alarm influence, are incorporated to make our model more realistic, which are ignored (Yin et al. 2012) or only considered for protecting targets (Gan, An, and Vorobeychik 2015; Gan et al. 2017); iii) our game, with a reasonable discretion of time and space, is extremely large-scale and cannot be solved by the existing methods (Yin et al. 2012; Yin and An 2016). We also note another line of research relate to our problem (Bošanskỳ et al. 2011; Jakob, Vaněk, and Pechoucek 2011; Vaněk et al. 2013) where the attacker attacks the ships whose schedules are fixed and known to both players. However, according to the incident reports, the attacker may damage the communication system and move the ship to another place, which deviates from the schdule.

In this paper we address the above challenges and provide four key contributions. First, we construct a Stackelberg Model of the Oil-Siphoning problem (SMOS), where both players take time-dependent paths on the grid as their strategies, based on the incident reports from actual attacks as well as special reports conducted by maritime authorities to make the model realistic and computable; Second, in order to compute the efficient defender's patrol strategy, we propose a compact formulation and a constraint generation (CG) algorithm, which avoids the exponentially increasing number of pure strategies of the defender and the attacker, respectively;

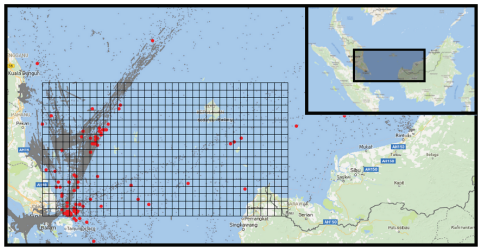


Figure 1: A map of the south region of SCS showing the ship locations during one week (grey) and pirate attacks between 2007 and 2016 (red dots). The rectangular area is considered in the case study in the case study in the evaluation section.

Third, to further improve the scalability, we propose an abstraction method to solve the extremely large-scale problem, which exploits intrinsic properties of the defender's strategy space to reduce the size of the game and makes a tradeoff between scalability and optimality; Finally, we evaluate our approaches through extensive simulations and a detailed case study with real ship traffic data. The results demonstrate that our approaches can scale up to realistic-size problems with modest influence on the solution quality, significantly outperforming existing methods, and provide robust solutions against the uncertainties in the real world.

Motivating Scenario

In this section, we analyze the situation in the south of the SCS shown in Figure 1, which is a current hotspot of siphoning attacks. According to the ReCAAP information sharing center (ISC), there were 99 piracy attacks in the SCS from 2007 to 2017. As the considered region is too large to be considered as a continuous model, we discretize the region into a grid of zones and assign the value of each zone according to the ship traffic data obtained from the automatic identification system (AIS). Besides, we discretize the time into equal steps, where the patrol boats use a number of steps from one node to its neighbors. Further, the vast majority of the attacks occurred during night time, which means that there is an ending time when the attackers have to finish their attack and that the patrol boats can relocate during the day, so there are no restrictions on the starting and ending zones of patrols and finite time steps are considered.

Note that siphoning can be done only in specific zones referred to as siphoning locations which can be determined via geographical features, usually close to the coast, or through ship density data. We assume that both the security agencies and the pirates know all possible siphoning locations. Based on the ReCAAP ISC reports (ReCAAP 2015b; 2015c), the model of the attack process includes: i) capturing the ship, including approaching, boarding the ship, controlling the crew and damaging the communication system, etc, which takes a certain amount of time; ii) steering the ship to the siphoning location through a path on the grid; and iii) siphoning the oil to another ship, which also takes a certain amount of time and during which the attacker cannot move. We assume that both capturing time and siphoning time are the same for all zones, respectively.

By patrolling through the zones, the patrol boats offer two kinds of protection. The first is that they might discover pirates during any step of the attack, which is mainly in the zones they patrol in, but also a weaker extent in neighboring zones, due to the noticed anomalies on AIS or radar systems installed on the boats. The second form of protection is due to the fact that patrolling ships can quickly react to alarms sent out by attacked ships before communication systems are damaged and so have a chance to disrupt attacks.

Model

We propose a **S**tackelberg **M**odel of the **O**il-**S**iphoning problem (SMOS) where the defender commits to a mixed patrolling strategy, then, after careful observation, the attacker performs an optimal response (Tambe 2011). We start by dividing the maritime territory into a set of zones $\mathcal{Z}$ which form a grid M. For a zone $i \in \mathcal{Z}$ we denote the neighbors of i on M by $\mathcal{N}(i)$[1]. Furthermore, we discretize the night into a sequence of time points $\mathbf{t} = \langle t_1, \ldots, t_\tau \rangle$, the consecutive time points are equidistant. Players can only act at these time points and every action, e.g., moving from one zone to a neighboring zone, is assumed to take one time step. The underlying grid together with the time line form a transition graph $G = (\mathcal{V}, \mathcal{E})$ on which the players can act. Every vertex $v = (i, t_k)$ consists of a zone $i \in \mathcal{Z}$ and a time step t_k. There is an edge e between two vertices $v = (i, t_k)$, $v' = (i', t_{k'})$ when $k' = k + 1$ and $i' \in \mathcal{N}(i)$.

Defender strategies. In our model the defender has m resources (i.e., patrol boats). Let $\mathcal{S} = \{v = (i, t_1)\}$ be the set of vertices corresponding to all possible zones at the first time step and $\mathcal{T} = \{v = (i, t_\tau)\}$ all possible zones at the last time step. Therefore, a patrol strategy P_r of a resource r is a path from a node in $\mathcal{S}$ to a node in $\mathcal{T}$. A pure strategy of the defender is then, consequently, given by a m-dimensional vector of patrol paths, i.e., $P = \langle P_1, \ldots, P_m \rangle$. A mixed strategy of the defender can be represented by a distribution of the possible pure strategies, i.e., $\mathbf{x} = \langle x_P \rangle$ where x_P is the probability of P being used. We have $\sum_P x_P = 1, \forall\, \mathbf{x}$.

Attacker strategies. Let $\mathcal{O} \subseteq \mathcal{Z}$ be the set of possible oil siphoning locations and let a and b denote the number of time steps it takes to capture a ship and to siphon the oil, respectively. We define an attacker's strategy as a path $F = ((i_1, t_h), \ldots, (i_l, t_{h+l}))$ on G where the first a time steps have to be spent on the same zone i, i.e., $i_1 = i_2 = \cdots = i_a$ and the last b time steps have to be spent on one of the oil siphoning zones, i.e., $i_{h+l} = i_{h+l-1} = \cdots = i_{h+l-b} \in \mathcal{O}$ and t_h is the time the attacker starts his attack. Since attacks are costly and occur with relative low frequencies, we assume there is an attacker and only pure strategies are considered, which is in line with previous works in security games (Kiekintveld et al. 2009).

Utilities and Stackelberg equilibrium. In our model each zone $i \in \mathcal{Z}$ has a certain value at each time step t_k denoted by $u(i, t_k)$ which is the value gained by the attacker when attacking a ship at zone i at time t_k. If the attacker successfully launches an attack F in zone i at t_k, he gets

[1]For convenience we define $i \in \mathcal{N}(i)$.

(a) (b)

Figure 2: (a) The red zones are neighboring zones of the black zone. (b) Suppose the attacker arrives at the black zone at $t_h + 2$, if the resource at any one of the red zones at t_h, the resource is *close enough* to catch the attacker.

$U(F) = u(i, t_k)$ and the defender gains $-U(F)$. If the defender detects the attacker, both players get 0. Therefore, our game is zero-sum, as most related works in security domain.

A resource can detect the attacker via three ways:

1. At a time step t_k, the attacker is detected with a probability $d \leq 1$ if they are inside the same zone;
2. At a time step t_k, the attacker is detected with a probability $e < d$ if they are in the neighboring zones;
3. At the starting time of the attack t_h, the attacker is detected if an alarm is raised by the ship being attacked and the resource is *close enough* to catch the attacker. The probability of raising an alarm is α.

A resource is considered close enough if it can interdict the attacker's path starting from its position at time t_h. Figure 2 presents a simple example of the three kinds of detection. Let P_r be the patrol strategy of resource r and F the attacker's strategy,

$$\mathcal{B}_r = \{v | v \in P_r \cap F\}$$

be the set of vertices they have in common,

$$\mathcal{N}_r = \{(i, t_k) | (i, t_k) \in P_r, \exists (j, t_k) \in F \text{ s.t. } j \in \mathcal{N}(i), j \neq i\}$$

the set of nodes where the patrol and the attacker are in neighboring zones and

$$\mathcal{R}_r = \{(i, t_h) | (i, t_h) \in P_r, \exists (j, t_k) \in F \colon d(i, j) \leq t_k - t_h\}$$

the set of zones from which the patrol could reach the attacker (at unit speed). Following the formulation adopted in (Jakob, Vaněk, and Pechoucek 2011; Yin et al. 2012; Yin and An 2016), the probability of detecting the attacker is calculated as $\min(1, d|\mathcal{B}_r| + e|\mathcal{N}_r| + \alpha \mathbb{1}_{\mathcal{R}_r})$ where $\mathbb{1}_{\mathcal{R}}$ is the indicator function of set $\mathcal{R}$, indicating if resource r could reach the attacker's path if an alarm is raised.

Further, we assume that the independent work of the patrols leads to an addition of the individual probabilities. Hence, given a strategy F of the attacker and a strategy $P = \langle P_r \rangle$ of the defender, the probability to detect the attacker is given by

$$dpp(P, F) = \min\left(1, \sum_{r=1}^{m} (d|\mathcal{B}_r| + e|\mathcal{N}_r| + \alpha \mathbb{1}_{\mathcal{R}_r})\right).$$

From this we can deduce the players' expected utilities. Given a pair of strategies (P, F) the attackers utility is $U^a(P, F) = (1 - dpp(P, F))U(F)$. When the defender is allowed to play a mixed strategy $\mathbf{x} = \langle x_P \rangle$, the attacker's expected utility is $U^a(\mathbf{x}, F) = \sum_P x_P U^a(P, F)$ by slight abuse of notation, the expected utility of the defender is defined as $U^d = -U^a$.

We want to compute the optimal strategy for the defender to commit to. Given the zero-sum setting, this is equivalent to maximizing the defender's utility under the assumption of best response of the attacker. This leads to the following optimization problem. Let $\mathcal{X}$ and $\mathcal{F}$ be the strategy space for the defender and attacker, respectively.

$$\max_{\mathbf{x} \in \mathbf{X}} U^d(\mathbf{x}, F)$$
$$\text{s.t. } U^a(\mathbf{x}, F) \geq U^a(\mathbf{x}, F'), \forall F' \in \mathcal{F}$$

Constraint Generation

In this section, we introduce a compact formulation to the original problem to avoid the exponentially large number of defender's pure strategies; and then adopt the constraint generation method to incrementally add the attacker's strategy into consideration to speed up the algorithm.

Compact LP formulation

The exponentially increasing number of pure strategies of both the defender and the attacker in the size of the game makes the computation of optimal solutions in the original formulation intractable. Many previous researches use compact representation as a remedy (Basilico and Gatti 2011; Yin et al. 2012; Yin and An 2016). In our model, a compact representation can be given via a coverage vector $\mathbf{c} = \langle c_v | v \in \mathcal{V} \rangle$ which assigns the marginal coverage on the node v in the graph. Let $\mathbf{x}$ be a given mixed strategy of the defender, we have $c_v = \sum_P x_P P(v)$ where $P(v)$ represents the number of resources going through v in the pure strategy P. For any pure strategy F of the attacker, we can write the attacker's utility $U^a(\mathbf{c}, F)$ as $(1 - dpp(\mathbf{c}, F))U(F)$ where

$$dpp(\mathbf{c}, F) = \min(1, d\sum_{v \in F} c_v + e\sum_{v \in \mathcal{N}(F)} c_v + \alpha \sum_{v \in \mathcal{R}(F)} c_v).$$

Now we can construct the linear program coreLP to compute the optimal coverage:

$$\max_{\mathbf{c}} U \tag{1a}$$
$$U \leq -U^a(\mathbf{c}, F), \forall F \in \mathcal{F} \tag{1b}$$
$$c_{(i,t_k)} = \sum_{j \in \mathcal{N}(i)} f_{((i,t_k),(j,t_{k+1}))}, \forall i \in \mathcal{Z}, k \in \{1, \ldots, \tau - 1\} \tag{1c}$$
$$c_{(i,t_k)} = \sum_{j \in \mathcal{N}(i)} f_{((j,t_{k-1}),(i,t_k))}, \forall i \in \mathcal{Z}, k \in \{2, \ldots, \tau\} \tag{1d}$$
$$m = \sum_{i \in \mathcal{Z}} c_{(i,t_k)}, k \in \{1, \tau\}, \tag{1e}$$

where $f_{((i,t_k),(j,t_{k+1}))} \in \mathbb{R}^+$ are flow variables, so that Eqs.(1c) and (1d) ensure that the coverage vector can be obtained by a mixed strategy (Yin and An 2016). Eq.(1e) forces the number of patrol resources at the beginning and end of the game to be m and Eq.(1b) represents that the attacker uses his best-response to the defender's strategy. Given a coverage vector $\mathbf{c}$, we can easily obtain a mixed strategy by splitting the flow into a set of weighted simple paths, which leads to only a minor loss for the defender (Yin et al. 2012). Notice that this formulation is still exponentially large in the size of the game since the number of pure strategies of the

attacker is exponentially large. Therefore, we use Constraint Generation (CG) to incrementally add the attacker's pure strategy into consideration to further improve the scalability.

CG for compact representation

In order to improve the scalability of coreLP, we apply Constraint Generation (CG) to the compact representation. Instead of starting with all of the exponentially many constraints, we start with only a small subset $\mathcal{F}'$. After we obtain a coverage vector for the subset $\mathcal{F}'$, we call the attacker oracle (AO). The AO computes, based on a given coverage vector $\mathbf{c}$, for every possible starting point (i, t_h) of an attack F the path which offers the attacker the best chances to escape. After considering every (i, t_h), we obtain the best response F^* of the attacker. If F^* is already in $\mathcal{F}'$ the algorithm terminates in an equilibrium, otherwise we add F^* into $\mathcal{F}'$ and restart the coreLP.

At the heart of the AO is Algorithm 1. For a given coverage $\mathbf{c}$ as well as the starting node (i_s, t_s), every step on the attacker's path brings a certain additional probability to be caught. Viewing these probabilities as weights on a graph, the goal of the attacker is to find a shortest path from (i_s, t_s) to a position where the attack is finished, i.e., reach a siphoning location and finish siphoning. This shortest path can be found with a modified Dijkstra's algorithm. Algorithm 1

Algorithm 1: findBestPath $(\mathbf{c}, i_s, t_s)$

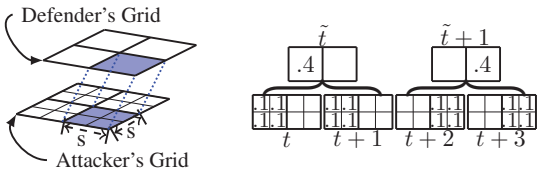
```
1  for (i, t_k) ∈ V do
2      c'_(i,t_k) ← d · c_(i,t_k) + e Σ_{j∈N(i)} c_(j,t_k);
3  Q ← initializeQueue();
4  while Q ≠ ∅ do
5      (p, i, t, b̂, A) ← Q.pop();
6      t++;
7      if b̂ = 0 then break ;
8      else if b̂ < b then
9          p_a, A' ← getAlarmPos(t_s, t, i, c, A);
10         Q.push((p + p_a + c'_(i,t), i, t, b̂ − 1, A'));
11     else for j ∈ N(i) do
12         p_a, A' ← getAlarmPos(t_s, t, j, c, A);
13         if j ∈ O then
             Q.push((p + p_a + c'_(i,t), i, t, b̂ − 1, A')) ;
14         Q.push((p + p_a + c'_(i,t), i, t, b̂, A'));
15 return (buildPath(i, t), p);
```

starts by computing the actual catching probability $\mathbf{c}'$ based on direct patrolling and the externality effect (lines 1 and 2). The elements $(p, i, t, \hat{b}, A)$ of priority queue Q consist of the probability to be caught p (this is the priority), the current position i, the current time t, the remaining time to finish the siphoning $\hat{b}$ and a list A of points which already have been considered as starting points of a patrol boat in case of an alarm. A is specific to the path and so cannot be precalculated. The initialization (line 3) also ensures that the first a time steps are spent in the same zone. Further, we keep track of nodes visited on the way to each node, so we can create the correct path at the end of the algorithm. In the main part of the algorithm (lines 4 to 14), we consider the node which currently has the lowest probability for detection, if the siphoning is finished we can stop (line 7), otherwise if the siphoning has started, we continue siphoning (lines 8 to 10) at the same position. The `getAlarmPos` finds new nodes from which a patrol starting at t_s could reach the current location in time to catch the attacker if an alarm would be raised. Moreover, `getAlarmPos` returns a list A' of all locations that have been considered so far. It also returns the additional catching probability p_a corresponding to the coverage of the locations added to A'. If the siphoning has not started (lines 11 to 14), the attacker can move to a neighboring zone j and if j is a siphoning location, the attacker also can start siphoning. Some special considerations are not displayed in Algorithm 1: If the attack does not finish in time the attack automatically fails; As soon as the catching probability $p \geq 1$ the attack will not be successful and does not need to be considered.; When $a = 0$ and $i_s \in \mathcal{O}$ the siphoning can start at t_s. Differently from Dijkstra's algorithm, we do not need to consider the update of probabilities of elements inside Q since the inbound weights into a node are all equal and so his weight cannot decrease after it is added to Q. Finally, the `buildPath` computes the attacker's path based on references to prior visited nodes (line 15).

The AO can find the attacker's best-response in time $O(|\mathcal{V}|^2(\log(|\mathcal{V}|) + |\mathcal{Z}|))$. The first term is the running time of the underlying Dijkstra algorithm and the second comes from the `getAlarmPos` subroutine, which finds new zones in linear time in $|\mathcal{Z}|$. So CG can solve small games efficiently. However, when the game becomes larger, e.g., with 100 zones and 12 time steps, it takes a long time to solve the coreLP, even with a small set $\mathcal{F}'$ due to the large number of variables. To further improve the scalability, we need to reduce the number of variables. Hence, we introduce an abstraction method in the next section.

Abstraction to Further Improve Scalability

Abstraction has been successfully applied to solve large-scale games by exploiting the intrinsic similarity of strategy space to shrink the game to a simpler one before solving it. As both players take paths as their strategies rather than a specific node and the zones on the attacker's escape path play a vital role for the defender to detect the attacker, the methods proposed in (Basilico, Gatti, and Amigoni 2009; Basak et al. 2016; Zhang et al. 2016), which typically remove the nodes unnecessary to reduce the size of the game, cannot be directly applied to our model. The basic idea of the abstraction method applied here is that the defender can only determine the coverage of a large area, which can be assigned to small areas following a pre-defined mapping method, which ensures that the generated coverage satisfies the flow conservation constraints. We note that the coverage of a specific node cannot influence the defender's utility dramatically, especially when the scale of the game becomes extreme large.



(a) (b)

Figure 3: (a) Abstraction with $s = 2$. (b) A counterexample which shows that the uniform mapping method cannot generate the valid coverage $\mathbf{c}$. The coverage of the defender's grid from $\tilde{t}$ to $\tilde{t}+1$ is valid for the defender's grid. While we use uniform mapping to map the coverage to the attacker's grid, the coverage at $t+2$ cannot be implemented by any flow from the coverage at $t+1$.

Abstraction of the grid

The first step to apply the abstraction method is to obtain the abstracted defender's grid by combining $s \times s$ neighboring nodes into a node where s is named as *sparsity*. Let $\tilde{i}$ denote the node on the defender's grid and we say the $i \in \tilde{i}, i \in \mathcal{Z}$ if i is included in $\tilde{i}$. We assume that the width and the length of the grid M, as well as τ, are divisible by s. The original grid M is referred as the attacker's grid. Figure 3a shows both defender's and attacker's grids. Now the defender's strategy is the valid coverage on the defender's grid, i.e., the coverage can be satisfied by some flow $\tilde{\mathbf{f}}$, denoted by $\tilde{\mathbf{c}}$ and the attacker's pure strategy is the same as previous. Note that because the node of the defender's grid is $s \times s$ times as large as the node of the attacker's grid, the length of a defender's time step is s times of the attacker's. To compute the attacker's best-response to a given $\tilde{\mathbf{c}}$, as well as the corresponding utilities, we need to map the coverage from the defender's grid to the attacker's grid. Hence, we introduce a mapping method Ψ in the next subsection.

Mapping method Ψ

In applying the abstraction method, we need to map the coverage vector $\tilde{\mathbf{c}}$ to the attacker's grid. The mapping method Ψ must ensure that the mapped coverage $\mathbf{c}$ is executable by some mixed strategy, i.e., there exists a flow $\mathbf{f}$ which satisfies Eqs.(1c) and (1d) on the attacker's grid. A straightforward method is uniform mapping, which uniformly assigns the flow over the nodes over each attacker's time step. However, Figure 3b provides a counterexample to show that uniform mapping cannot work. Therefore, we propose Algorithm 2 to generate the valid coverage on the attacker's grid.

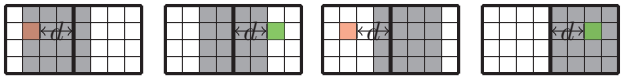
The basic idea of Algorithm 2 is keeping the trace of flow $\tilde{\mathbf{f}}$ on the defender's grid. Figure 4 is an example which shows the change of a flow at each attacker's time step from a node to one of its neighbors on the defender's grid. For each attacker's time step, if the node is covered, we will add the flow into its coverage. Note that the flow is uniformly assigned to each node it covers. After considering all flows on the defender's grid, we obtain the coverage $\mathbf{c}$.

In detail, we first map the defender's time step $\tilde{t}_k$ to the attacker's (line 3). For each flow $\tilde{f}_{(\tilde{i},\tilde{t}_k),(\tilde{j},\tilde{t}_{k+1})}$ such that

Algorithm 2: Mapping method $\Psi(\tilde{\mathbf{f}})$

```
1  c = 0;
2  for f̃_{(ĩ,t̃_k),(j̃,t̃_{k+1})} ∈ f̃ do
3      t ← s · t̃_k;
4      if ĩ ≠ j̃ then
5          for Δt = 1 : s do
6              for i ∈ ĩ && d_i + Δt < s do
7                  c(i, t + Δt) += (1/s^2) · f̃_{(ĩ,t̃_k),(j̃,t̃_{k+1})};
8              for j ∈ j̃ && d_j < Δt do
9                  c(j, t + Δt) += (1/s^2) · f̃_{(ĩ,t̃_k),(j̃,t̃_{k+1})};
10     else for Δt = 1 : s && i ∈ ĩ do
11         c(i, t + Δt) += (1/s^2) · f̃_{(ĩ,t̃_k),(j̃,t̃_{k+1})};
12     ;
13 return c;
```

$\tilde{i} \neq \tilde{j}$, which implies a flow flows from one node to another different node, we define the distance d_i for each node $i \in \tilde{i}$, as the distance from node i to the edge where the flow flows out. The distance $d_j, j \in \tilde{j}$ is defined analogously as the distance from node j to the edge where the flow flows in. For each attacker's time step Δt between $\tilde{t}_k$ and $\tilde{t}_{k+1}$ (line 5), for node $i \in \tilde{i}$, if $d_i + \Delta t < s$, which implies the flow does not leave this node, we will add the flow into i's coverage (lines 6 to 7). Analogously, if $d_j < \Delta t$, which implies the flow reaches this node, we will add the flow into j's coverage (lines 8 to 9). When $\Delta t = s$, the flow $\tilde{f}_{(\tilde{i},\tilde{t}_k),(\tilde{j},\tilde{t}_{k+1})}$ reaches $\tilde{j}$. For the flow $\tilde{f}_{(\tilde{i},\tilde{t}_k),(\tilde{j},\tilde{t}_{k+1})}$ with $\tilde{i} = \tilde{j}$, which implies a flow from a node to itself, we simply assign the flow to the nodes $i \in \tilde{i}$ uniformly (lines 10 to 11). Then we obtain the coverage $\mathbf{c}$.



(a) $\Delta t = 1$ (b) $\Delta t = 2$ (c) $\Delta t = 3$ (d) $\Delta t = 4$

Figure 4: The change of a flow from a node to one of its neighbors (right neighbor) along with the attacker's time step with $s = 4$. The gray nodes are covered by the flow at each attacker's time step. Besides, the red (green) node is one of the node of the defender's grid where the flow flows out (in) and the corresponding distance $d = 2$, defined in Algorithm 2, to the edge where the flow passes is displayed.

Proposition 1. *The generated* $\mathbf{c}$ *can be satisfied by a flow.*

Proof. We can find a flow to satisfy $\mathbf{c}$. For each $\tilde{f}_{(\tilde{i},\tilde{t}_k),(\tilde{j},\tilde{t}_{k+1})} \in \tilde{\mathbf{f}}$ such that $\tilde{i} \neq \tilde{j}$, for $i \in \tilde{i}$, we denote the neighbor as i' of i where the flow reaches i' before i, as well as j' for $j \in \tilde{j}$. For $\Delta t = 1 : s$, if $d_i + \Delta t < s$, $f_{(i',t-1+\Delta t)(i,t+\Delta t)} = (1/s^2) \cdot \tilde{f}_{(\tilde{i},\tilde{t}_k),(\tilde{j},\tilde{t}_{k+1})}$; and if $d_j < \Delta t$, $f_{(j',t-1+\Delta t)(j,t+\Delta t)} = (1/s^2) \cdot \tilde{f}_{(\tilde{i},\tilde{t}_k),(\tilde{j},\tilde{t}_{k+1})}$. For each

$\tilde{f}_{(\tilde{i},\tilde{t}_k),(\tilde{j},\tilde{t}_{k+1})} \in \tilde{\mathbf{f}}$ such that $\tilde{i} = \tilde{j}$, $f_{(i,t-1+\Delta t)(i,t+\Delta t)} = (1/s^2) \cdot \tilde{f}_{(\tilde{i},\tilde{t}_k),(\tilde{j},\tilde{t}_{k+1})}$. Now we get $\mathbf{f}$ which satisfies $\mathbf{c}$. □

Analysis We can use the ratio $r = \frac{U-U_s^a}{U-U^a}$ to evaluate the performance of the abstraction method where $U = \max\{u(i,t_k)\}$, U_s^a and U^a are utilities of the attacker with and without the abstraction, respectively. For the general games, it is difficult to obtain a good bound for the abstraction method theoretically due to the highly interdependence between time and space. However, for *uniform* games where all zones are with the same value, which means the shipping traffic are uniformly assigned to each zone and steady in all time steps, we have the following proposition.

Proposition 2. *For uniform games,* $r \geq 1/s^2$.

Proof. Because the zones are with the same value in the uniform game, the attacker will only attack the zones in $\mathcal{O}$, which can bring at least the same utility if he attacks other zones. Therefore, the defender would only assign the coverage to zones in $\mathcal{O}$ and keep the coverage steady during all time steps. Suppose the optimal coverage is $\mathbf{c}$ and the corresponding attacker's utility is U^a. Then, we can construct a valid coverage for the abstraction $\mathbf{c}_s$ with sparsity s: for each time step $t_k, k \in \{1,\cdots,\tau\}$, $c_s(i,t_k) = \frac{1}{s^2}\sum_{j\in\tilde{i}} c(j,t_k), \forall\ i \in \tilde{i}$, which implies that $c_s(i,t_k) \geq \frac{1}{s^2}c(i,t_k)$. As the values of all zones are the same, $r = \frac{dpp(\mathbf{c}_s,F_s)}{dpp(\mathbf{c},F)}$ where F_s and F are the attacker's best-response to $\mathbf{c}_s$ and $\mathbf{c}$, respectively. If $F_s = F$, $r \geq \frac{1}{s^2}$ due to the fact that $c_s(i,t_k) \geq \frac{1}{s^2}c(i,t_k)$. If $F_s \neq F$, as F is the best response to $\mathbf{c}$, , we can obtain that $dpp(\mathbf{c}_s,F_s) \geq \frac{1}{s^2}dpp(\mathbf{c},F_s) \geq \frac{1}{s^2}dpp(\mathbf{c},F)$, which leads to $r \geq \frac{1}{s^2}$. □

Proposition 2 provides the worst case guarantee and we found that the abstraction can achieve a close approximation to optimal solution quality with a dramatic improvement of scalability in the experimental evaluation.

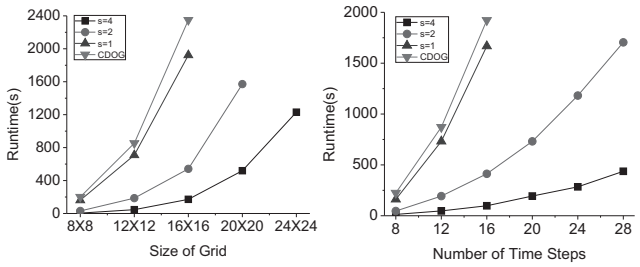
Evaluation

We evaluate the performance of our approach based on i) extensive experiments completed via simulations and ii) real-world ship density data of CSC. We use CPLEX (version 12.6) to solve the LPs and all computations are performed on a 64-bit PC with 16.0 GB RAM and a 3.50 GHz CPU. The detecting factor d, the external detecting factor e and the alarm probability α are fixed as 0.5, 0.1 and 0.25, respectively, unless otherwise specified.

Experimental evaluation on synthetic data

In this section, we systematically generate square grids with different sizes. The values of nodes of grids are randomly generated from a uniform distribution in $[0, 100]$. For all simulations, the attack time a and the siphoning time b are fixed as 2 and 3, respectively. All values are averaged over 30 instances unless otherwise specified.

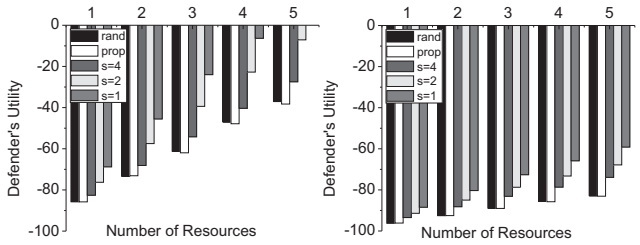
We compare the scalability of three versions of our algorithms: i) our CG algorithm, which is denoted as $s = 1$ in the figures; ii) our abstraction method with $s = 2$; and iii) our abstraction method with $s = 4$. The CDOG algorithm proposed in (Yin and An 2016) is also implemented as a benchmark with a slight modification that we relax the coverage vector $\mathbf{c}$ to be continuous. To evaluate the solution quality, we compare our algorithms' solutions with two heuristic patrol strategies: i) **rand** where $c(i,t_k) = m/|\mathcal{Z}|$, which implies that the defender randomly patrols all zones; and ii) **prop** where $c(i,t_k) = \sum_{t_k=1}^{\tau} u(i,t_k) \cdot m / \sum_{i=1}^{|\mathcal{Z}|}\sum_{t_k=1}^{\tau} u(i,t_k)$, which implies that the defender patrols the zones proportionally to the values of zones. Note that when $s = 1$, the abstraction method is the same as CG which returns the optimal solution as CDOG.



(a) $m = 3, \tau = 12$ (b) $|\mathcal{Z}| = 12 \times 12, m = 3$

Figure 5: Runtime analysis

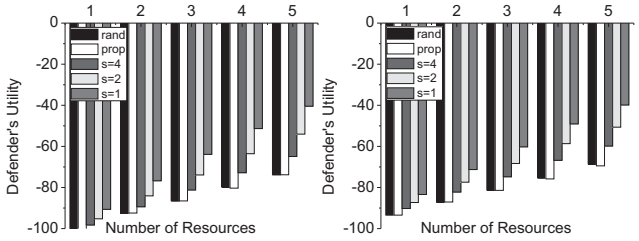
Scalability analysis. We first compare the scalability of our algorithms. The result is depicted in Figure 5a where the x-axis indicates the size of the grid. The result shows that both CG and CDOG cannot scale up to the grid larger than 16×16 with runtime cap of 2400 seconds. Besides, CG is always faster than CDOG because CDOG needs to call the defender oracle many times to generate the subgrid. While, our abstraction methods with $s = 4$ and $s = 2$ can solve the grid with 20×20 in less than 1600 seconds, which significantly outperform the CG and CDOG. We also compare the scalability of our algorithm on games with different time length. The result is showed in Figure 5b. The result shows that our abstraction method can scale up to the realistic length of time steps and for small sparsity, the runtime increases faster because when τ increases, more variables and constraints are added into the formulation. We also vary the number of resources, which has much less influence on the runtime and the result is not displayed.



(a) $|\mathcal{Z}| = 8 \times 8, \tau = 12$ (b) $|\mathcal{Z}| = 16 \times 16, \tau = 12$

Figure 6: Solution quality

Solution quality. As the main advantage of the abstraction method is that it can significantly improve the scalability with some loss of the optimality, we then evaluate the quality of the solutions obtained by the abstraction method. The results, displayed in Figures 6a and 6b, show that given the grids, when the number of resources m increases, our abstraction method shows a greater advantage against the random patrol and when the sparsity s becomes smaller, the abstraction method provides a better approximation to the optimal solution. It is worth to note that in all experiments, the abstraction obtains a much higher value of the ratio r ($0.6 \sim 0.8$) compared with the theoretical bounds. Given the number of resources m, when the gird becomes larger, the advantage of our solutions against the two baselines is reduced. Besides, the proportional patrol strategy gets even worse utility than the random patrol strategy in some cases, because patrolling the nodes with low values is also important to catch the attacker who takes paths as strategies.



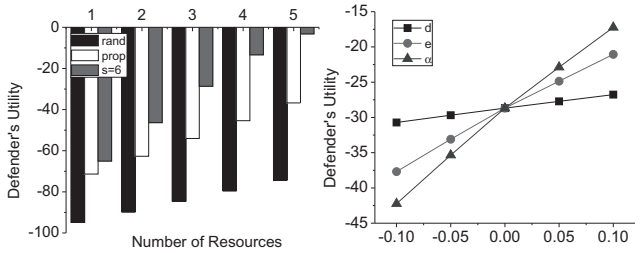
(a) $\hat{u}$ ($\delta = 0.1$) (b) $\hat{c}$ ($\rho = 0.1$)

Figure 7: Robustness

Robustness analysis. In reality, the defender may not know the exact value of each zone due to the fluctuation of ship density over time. Besides, during the patrol process, the real coverage of each zone may deviate from the defender's strategy. Thus, we evaluate the robustness of our solutions in two aspects: i) the real value of each zone $\hat{u}(i, t_k)$ ranges in $[1 - \delta, 1 + \delta] \cdot u(i, t_k)$, and ii) the real coverage of each zone $\hat{c}(i, t_k)$ ranges in $[1 - \rho, 1 + \rho] \cdot c(i, t_k)$. The results are displayed in Figure 7 with $\delta = \rho = 0.1$. The results show that our solutions are robust enough and outperform the random patrol strategy under a high level of uncertainty of targets' values and coverage. Note that when the number of resources m is extremely small, the advantage of our solution with large sparsity is reduced.

SCS real-world evaluation

We consider an area of 64800 NM^2 in the south of the SCS showed in Figure 1. We define the attacker's grid by dividing this area into small square zones with a length of 10 NM, which ensures that a patrol boat can provide efficient protection of a zone at each time step through its radar system. By a normal patrol speed of 15 knots, it takes 40 minutes from one zones to the next (DAMEN 2016), so we consider 18 time steps for twelve hours of darkness, i.e., $\tau = 18$. We assume $a = 4$ and $b = 6$ which imply that the attackers need roughly three hours to capture a ship and four to siphon the goods according to the incident reports. In order to determine the



(a) Solution quality (b) Range factors

Figure 8: Real-world application

value of each zone we obtained data from AIS used by vessel traffic services. The data cover the first week of February 2017 and after deletion of faulty entries, it contained about 20000 entries in the considered area. In 500 of the 648 zones no ship appeared during the week, we declare these zones as siphoning locations. The value of each zone at each time point was then mapped to lie in $[0, 100]$ where all locations where 30 or more ships came through in one time step were mapped to a value of 100[2].

The results are depicted in Figure 8. All instances are solved averagely in 2800 seconds, which is better than our simulated results due to the vast zones with value 0, which reduces the strategy space of both players. In Figure 8a, we range the number of resources and choose $s = 6$ to our abstraction method. The results show that our algorithm significantly outperforms the two baselines. In Figure 8b, we range different factors d, e and α by $[-0.1, 0.1]$ around the default values with $m = 3$ and $s = 6$. The results show that improving the alarm probability α has more influence on the utility, compared with d and e, because it increases the detecting probability of all closing zones near the attack, so we highly recommend that the ships install fast and responsive alarm system. Besides, improving e is more effective than improving d to improve the defender's utility due to the improvement of coverage of all neighbors of nodes.

Conclusion

This paper aims at addressing the oil-siphoning problem through efficiently assigning the limited patrol resources. A novel Stackelberg model, SMOS, is proposed ,based on the traffic data, where both players take paths as their strategies with different kinds of externalities. A compact formulation and a constraint generation algorithm with efficient attacker oracle to add constraints is proposed. To further improve the scalability, an abstraction method is proposed. Extensive experimental results and a detailed case study for SCS demonstrate that our approaches can result in dramatic improvement of scalability with modest influence on the solution quality and can scale up to realistic-sized problems.

[2]A flaw of AIS data is that there are some zones which have extremely larger traffic density than their neighbors at some time step. To make the data reasonable, zones with more than 30 are considered outliers, otherwise they will dominate the game.

Acknowledgments

This research is supported by NRF2015 NCR-NCR003-004 and NCR2016NCR-NCR001-002

References

An, B.; Shieh, E.; Tambe, M.; Yang, R.; Baldwin, C.; DiRenzo, J.; Maule, B.; and Meyer, G. 2012. PROTECT – A deployed game theoretic system for strategic security allocation for the United States coast guard. *AI Magazine* 33(4):96–110.

Basak, A.; Fang, F.; Nguyen, T. H.; and Kiekintveld, C. 2016. Abstraction methods for solving graph-based security games. In *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 13–33.

Basilico, N., and Gatti, N. 2011. Automated abstractions for patrolling security games. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence (AAAI)*, 1096–1101.

Basilico, N.; Gatti, N.; and Amigoni, F. 2009. Leader-follower strategies for robotic patrolling in environments with arbitrary topologies. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 57–64.

Bošanskỳ, B.; Lisỳ, V.; Jakob, M.; and Pěchouček, M. 2011. Computing time-dependent policies for patrolling games with mobile targets. In *The 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 989–996.

Bowden, A.; Hurlburt, K.; Aloyo, E.; Marts, C.; and Lee, A. 2010. *The economic costs of maritime piracy*. One Earth Future Foundation.

DAMEN. 2016. Stan Patrol 3007 - Executive Summary. http://products.damen.com/en/ranges/stan-patrol/stan-patrol-3007.

Fang, F.; Nguyen, T. H.; Pickles, R.; Lam, W. Y.; Clements, G. R.; An, B.; Singh, A.; Tambe, M.; and Lemieux, A. 2016. Deploying PAWS: Field optimization of the protection assistant for wildlife security. In *Proceedings of the Twenty-Eighth Innovative Applications of Artificial Intelligence Conference (IAAI)*, 3966–3973.

Fang, F.; Stone, P.; and Tambe, M. 2015. When security games go green: Designing defender strategies to prevent poaching and illegal fishing. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*, 2589–2595.

Gan, J.; An, B.; and Vorobeychik, Y. 2015. Security games with protection externalities. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI)*, 914–920.

Gan, J.; An, B.; Vorobeychik, Y.; and Gauch, B. 2017. Security games on a plane. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI)*, 530–536.

Jain, M.; Tsai, J.; Pita, J.; Kiekintveld, C.; Rathi, S.; Tambe, M.; and Ordóñez, F. 2010. Software assistants for randomized patrol planning for the LAX airport police and the Federal Air Marshal Service. *Interfaces* 40(4):267–290.

Jakob, M.; Vaněk, O.; and Pechoucek, M. 2011. Using agents to improve international maritime transport security. *IEEE Intelligent Systems* 26(1):90–96.

Kemp, T. 2015. Crime on the high seas: The world's most pirated waters. *CNBC*.

Kiekintveld, C.; Jain, M.; Tsai, J.; Pita, J.; Ordóñez, F.; and Tambe, M. 2009. Computing optimal randomized resource allocations for massive security games. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 689–696.

ReCAAP. 2015a. Annual report - piracy and armed robbery against ships in Asia. http://www.recaap.org/DesktopModules/Bring2mind/DMX/Download.aspx?Command=Core_Download&EntryId=421&PortalId=0&TabId=78.

ReCAAP. 2015b. Half yearly report - piracy and armed robbery against ships in Asia. http://www.neptunemaritimesecurity.com/imb-recaap-isc-release-2015-half-yearly-reports-on-piracy-armed-robbery-at-sea/.

ReCAAP. 2015c. Incident update siphoning of fuel/oil from Orkim Victory. http://www.recaap.org/Portals/0/docs/Latest%20IA/2015/Incident%20Update%20Orkim%20Victory%20(4%20Jun%2015).pdf.

Tambe, M. 2011. *Security and Game Theory: Algorithms, Deployed Systems, Lessons Learned*. Cambridge University Press.

Vaněk, O.; Jakob, M.; Hrstka, O.; and Pěchouček, M. 2013. Agent-based model of maritime traffic in piracy-affected waters. *Transportation Research Part C: Emerging Technologies* 36:157–176.

Xue, J. 2015. Singapore, Malaysia and Indonesia could extend joint patrols in South China Sea. *Channel News Asia*.

Yin, Y.; An, B.; and Jain, M. 2014. Game-theoretic resource allocation for protecting large public events. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI)*, 826–834.

Yin, Y., and An, B. 2016. Efficient resource allocation for protecting coral reef ecosystems. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI)*, 531–537.

Yin, Z.; Jiang, A. X.; Johnson, M. P.; Kiekintveld, C.; Leyton-Brown, K.; Sandholm, T.; Tambe, M.; and Sullivan, J. P. 2012. TRUSTS: Scheduling randomized patrols for fare inspection in transit systems. In *Proceedings of the Twenty-Eighth Innovative Applications of Artificial Intelligence Conference (IAAI)*, 2348–2355.

Zhang, C.; Bucarey, V.; Mukhopadhyay, A.; Sinha, A.; Qian, Y.; Vorobeychik, Y.; and Tambe, M. 2016. Using abstractions to solve opportunistic crime security games at scale. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems (AAMAS)*, 196–204.

Zhang, Y.; An, B.; Tran-Thanh, L.; Wang, Z.; Gan, J.; and Jennings, N. R. 2017. Optimal escape interdiction on transportation networks. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI)*, 3936–3944.