Efficient Resource Allocation for Protecting Coral Reef Ecosystems

Yue Yin\textsuperscript{1,2}, Bo An\textsuperscript{3}
\textsuperscript{1}The Key Lab of Intelligent Information Processing, ICT, CAS
\textsuperscript{2}University of Chinese Academy of Sciences, Beijing, China
\textsuperscript{3}School of Computer Science and Engineering, Nanyang Technological University, Singapore
\textsuperscript{1}melody1235813@gmail.com, \textsuperscript{3}boan@ntu.edu.sg

Abstract
Corals reefs are valuable and fragile ecosystems which are under threat from human activities like coral mining. Many countries have built marine protected areas (MPAs) and protect their ecosystems through boat patrol. However, it remains a significant challenge to efficiently patrol the MPAs given the limited patrol resources of the protection agency and potential destructors’ strategic actions. In this paper, we view the problem of efficiently patrolling for protecting coral reef ecosystems as a game-theoretic perspective and propose 1) a new Stackelberg game model to formulate the problem of protecting MPAs, 2) two algorithms to compute the efficient protection agency’s strategies: CLP in which the protection agency’s strategies are compactly represented as fractional flows in a network, and CDOG which combines the techniques of compactly representing defender strategies and incrementally generating strategies. Experimental results show that our approach leads to significantly better solution quality than that of previous works.

1 Introduction
Coral reefs are precious natural resources, which form some of the world’s most productive ecosystems, providing complex marine habitats that support a wide range of other organisms. However, some human activities, like coral mining, can severely damage the coral reef ecosystems. Once coral reefs are destroyed, it may take tens of years for them to restore. Therefore, many countries have built marine protected areas (MPAs) to restrict potentially damaging activities by patrolling in the MPAs [Bellwood \textit{et al.}, 2004]. It is a great challenge to efficiently protect the MPAs through patrolling since protection agencies usually have to protect a large open water area using very limited resources (e.g., the protection agency in the Yalongwan MPA in China protects an 85 square kilometers area with 3 patrol boats). In addition, potential destructors can learn the protection agency’s strategies through surveillance, then choose the most undetectable time and the most covert path in the open water to arrive at a specific area to perform illegal activities. We aim at developing efficient patrol strategies for protecting the coral reef ecosystems.

Though the crisis faced by coral reefs has been investigated by many researchers [Bellwood \textit{et al.}, 2004; Pandolfi \textit{et al.}, 2003], most previous works focus on conservation planning instead of detecting and deferring potential damage [Carrardine \textit{et al.}, 2009]. Meanwhile, there has been significant progress on applying game theoretic approaches to security domains like protection of infrastructures [Tambe, 2011; Letchford and Conitzer, 2013; An \textit{et al.}, 2013; Yin \textit{et al.}, 2014; 2015; Wang \textit{et al.}, 2016; Shieh \textit{et al.}, 2012]. In our scenario, the interaction between the protection agency (defender) and the potential destructor (attacker) can also be modeled as a game, but previous work cannot be directly used here due to two new challenges. First, the playfield is a large open water area, both players’ strategies are time-dependent paths, i.e., the defender patrols while the attacker chooses some time to sail to his target area. Second, unlike activities such as igniting a bomb which can be done quickly, damaging activities at an MPA (e.g., coral mining) only succeed if they last for a relatively long time. Most previous works assume that at most one player takes paths [Fang \textit{et al.}, 2015; Basilico \textit{et al.}, 2009], or that time is irrelevant [Jain \textit{et al.}, 2013] and attack can be done immediately [Gan \textit{et al.}, 2015]. For previous works that considered attack duration, they either consider time duration of attacks as external parameters but not part of the attacker’s strategy [Alpern \textit{et al.}, 2011], or have different goals from us [Yin \textit{et al.}, 2012; Bosansk \textit{et al.}, 2015]. The two new challenges make the strategy spaces of the players larger and more complicated, which leads to great challenge in computation.

This paper makes four key contributions. First, we propose a defender-attacker Stackelberg game model to formulate the problem of protecting MPAs, in which both game players take time-dependent paths, and payoffs of players are affected by the time duration of the attack. Second, we propose a compact linear program to solve the game, in which we compactly represent defender strategies as fractional flows on graphs to reduce the number of variables in the game. To further scale up the algorithm, our third contribution is a compact-strategy double-oracle algorithm on graphs (CDOG) which combines the techniques of compactly representing defender strategies and incrementally generating strategies. Finally, extensive experimental results show that our algorithms lead to significantly better solution quality than that of other algorithms in the literature and CDOG scales up well.
2 Motivating Scenario

Figure 1(a) shows the landscape of the Great Barrier Reef Marine Park in Australia. There is an authority (defender) responsible for the protection of the park, with offices located on the coast, shown as red stars in Figure 1(a). The defender can divide the MPA into several zones to design patrol strategies. Figure 1(b) shows an example division. The defender can stay at a zone or take a path among zones. Each zone is a potential attack target. The effect of damaging different zones can be different and time-dependent. The attacker enters the park at some time, takes a path to his target zone and determines how long to perform activities at this zone. Both agents may be limited to start and finish the path at certain zones, e.g., office locations, peripheral zones in the park (zones except 6 in Figure 1(b)). Strategic attackers can target a single zone. We did not consider how the attacker escapes after attacking since the coral reefs are damaged anyway after the attack. In addition, before the attacker finishes the attack, he needs to look for a location and operate the equipment, which can make him suspicious; while after the attack, he can easily camouflage himself and flee fast, which makes it difficult to catch him.

3 Model

We model the problem as a Stackelberg game [Tambe, 2011], then the attacker conducts surveillance and chooses the optimal strategy to respond to the defender’s strategy. We first construct a transition graph with a timeline. Denote an MPA as a collection of n zones \( Z = \{1, 2, \cdots, n\} \). We evenly discretize a day as a sequence of \( \tau \) time points \( t = (t_1, \cdots, t_\tau) \) with interval \( \delta \). Assume that the time needed to travel between two adjacent zones is a multiplier of \( \delta \) (this assumption holds as long as \( \delta \) is small enough). Assume that the defender and the attacker travel at the same speed and they only move at time points \( t_k \in t \). Let \( D = \{d_{ij} \in \{1, 2, \cdots\} \} \) with \( d_{ij} \) representing that the time needed to move from zone \( i \) to adjacent zone \( j \) is \( d_{ij} \cdot \delta \). To represent players’ strategies, we construct a directed transition graph \( G = (V, E) \) where a vertex \( v = (i, t_k) \) corresponds to zone \( i \) and time \( t_k \). There is an edge \( e = (v = (i, t_k), v' = (j, t_{k'})) \) if one of the following two conditions holds:

1. \( j = i, k' = k + 1 \). We call such edges stay edges.
2. \( i \) and \( j \) are adjacent zones and \( k' = k + d_{ij} \). We call such edges moving edges.

Consider a simple MPA graph in Figure 2(a) which includes 3 zones. Let \( t = (t_1, t_2, t_3) \) and \( d_{ij} = 1, \forall i, j \in \{1, 2, 3\}, i \neq j \). We can get a transition graph in Figure 2(b). The edge between \((1, t_1)\) and \((1, t_2)\) indicates that the defender can patrol in zone 1 and the attacker can perform activities in zone 1 during \((t_1, t_2)\). An edge connecting \((1, t_1)\) and \((2, t_2)\) indicates that if a player moves from zone 1 at time \( t_1 \), he will arrive at zone 2 at time \( t_2 \).

![Figure 1: The Great Barrier Reef Marine Park](image)

![Figure 2: MPA graph and transition graph](image)

**Defender strategies.** Assume that the defender has \( m \) resources, i.e., \( m \) patrol boats, and each resource can keep patrolling for time duration \( \theta \delta \). Let \( Z^d \subset Z \) be zones that the defender can start and end her patrol\(^1\). Since the patrol can start at any time before \( t_{\tau - 0} \), we add a collection of virtual source vertices to the transition graph, i.e., \( S = \{S_1, S_2, \cdots, S_{\tau - 0}\} \). For \( S_k \in S \), we add an edge from \( S_k \) to vertex \((i, t_k) (\forall i \in Z^d) \). Similarly, we add a collection of virtual terminal vertices \( T = \{T_1, T_2, \cdots, T_{\tau - 0}\} \) such that \( \forall T_k \in T \), there is an edge from vertex \((i, t_k + \theta) (\forall i \in Z^d) \) to \( T_k \). Therefore, a patrol strategy \( P_r \) of a resource \( r \) is a flow from \( S_k \) to \( T_k \). A pure strategy of the defender is a set of \( m \) ‘patrol strategy flows’, i.e., \( P = \{P_r : r \in \{1, \cdots, m\}\} \). A mixed strategy of the defender is a distribution over pure strategies, i.e., \( x = \{x_P\} \) where \( x_P \) represents the probability of \( P \) being used.

Consider the example in Figure 2(b). Assume that \( Z^d = \{1, 3\} \) and \( \theta = 1 \), thus the source vertices and terminal vertices can be added as is shown in Figure 3(a), i.e., \( S_1 \) is connected to \((1, t_1)\) and \((3, t_1)\), meaning that the defender can start the patrol from zone 1 or 3 at time \( t_1 \), while \( T_1 \) is connected to \((1, t_2)\) and \((3, t_2)\), meaning that patrols starting from \( S_1 \) should end in zone 1 or 3 at time \( t_2 \). Any flow from \( S_k \) to \( T_k \) is a feasible patrol strategy, e.g., \( S_2 \rightarrow (1, t_2) \rightarrow (3, t_2) \rightarrow T_2 \) represents that the defender patrols on the way from zone 1 to zone 3.

**Attacker strategies.** Assume that the attacker can also start at a subset of the zones \( Z^a \subset Z \). An attacker’s strategy includes two parts: a path in the transition graph to go to his target zone, and how long he attacks at the target zone. We assume that the attack duration is a multiplier of \( \delta \) (this holds when \( \delta \) is small enough) and denote an attacker’s strategy as \( Y = (H_Y, A_Y) \), where \( H_Y \) is a path leading to his target vertex \((i, t_k)\) and \( A_Y \) is a path consisting of \( l \) adjacent stay edges, representing that the attacker performs activity at zone \( i \) from

\(^1\)Our model can be easily expanded to handle cases in which the defender starts from and finishes at different sets of zones.
time $t_k$ to time $t_{k+i}$. Consider the previous 3-zones example. Assume that $Z^a = \{1\}$. Figure 3(b) shows a feasible attacker strategy where the attacker enters the MPA from zone 1 at $t_1$, arrives at his target zone 2 at $t_2$, then attacks zone 2 till $t_3$. As previous work in security games, we restrict attacker strategies to pure strategies [Kiekintveldt et al., 2009].

Utilities and equilibrium. We assume that each stay edge $e = (i, t_k), (i, t_{k+1})$ in the transition graph has a value of $V_e$, representing the attacker’s payoff of successfully performing activities in zone $i$ during time $(t_k, t_{k+1})$. If an attacker successfully plays strategy $Y = (H_Y, A_Y)$, he gains a utility of $U(Y) = \sum_{i \in A_Y} V_e$, while the defender gets a utility of $-U(Y)$. If the attacker is detected by the defender, he fails and both agents get a utility of 0. The attacker may be detected by the defender if their strategies share same edges. Specifically, if their paths share a stay edge, the attacker may be detected when he is staying at some zone; if their paths share a moving edge, the attacker may be detected when he is moving from one zone to another. Naturally, the defender may not be able to perform detection every time she meets a boat which could be a potential attacker. To describe the probability that the defender detects a boat, we make the following two assumptions, which are realistic enough to describe most real world scenarios.

**Assumption 1.** If the defender and the attacker first meet on edge $e$, the probability that the defender detects the attacker, i.e., the detecting factor of edge $e$, is $f_e \in [0, 1]$.

The next assumption is about the probability that the defender detects the attacker after they meet for the first time. Since patrol areas are usually somewhere prone to be maliciously damaged but not popular travel sites, a boat appearing in such areas frequently is very suspicious. Therefore, if the defender has met a boat for many times on her way before she arrives at edge $e$, then the boat seems more suspicious than a boat which is seen for the first time on edge $e$, thus should be detected with a higher probability than $f_e$.

**Assumption 2.** Assume that the defender and the attacker have been on same edges $e_1, e_2, \cdots, e_{k-1}$ and the attacker has not been detected. If they meet on edge $e_k$, the probability that the defender detects the attacker is $\min\{1, \frac{f_{e_k}}{1 - \sum_{i=1}^{k-1} f_{e_i}}\}$.

The probability shown in Assumption 2 ranges in $[f_{e_k}, 1]$, which satisfies the intuition that more suspicious boat should be detected with a higher probability. Based on Assumption 2, if the defender’s strategy and the attacker’s strat-
attacker, the expected attacker utility can be represented as
\[
U^a(c, Y) = (1 - \min \{1, \sum_{e \in Y} c_e f_e \} ) U(Y).
\] (3)

Our problem now lies in computing the marginal coverage which corresponds to the optimal mixed strategy of the defender. First, we need to construct marginal coverages corresponding to feasible mixed strategies. One challenge of the construction lies in that a mixed defender strategy consists of pure strategies starting from different time points. Yin et al. [2012] have proven that the problem can be solved by constructing an extended version \( E^G \) of the transition graph \( G \), and considering the marginal coverage as the sum of several flows on \( E^G \). Technically, \( E^G \) is composed of multiple restricted copies of \( G \) (i.e., subgraphs of \( G \)), corresponding to different possible starting time points for the defender. For the copy corresponding to starting time points \( t_k \), we only keep the subgraph \( G_k \) on vertices \( \langle S_k, T_k, v = \langle i, t_k \rangle : i \in Z, k' \in \{k, k+1, \ldots, k+\theta \} \rangle \). Therefore, any defender patrol strategy starting at \( t_k \) can be represented as an \( S_k - T_k \) flow in subgraph \( G_k \). Let \( z_k(e) \) represent the expected number of patrollers on edge \( e \) which come from patrol strategies starting at time point \( t_k \) (\( z_k(e) \geq 0 \)). Let \( \Gamma(e) \) represent the set of subgraphs which include edge \( e \). Let \( c_e = \sum_{k:G_k \in \Gamma(e)} z_k(e) \).

Yin et al. [2012] show that if \( z_k(e) \) satisfies conservation of flow, a defender mixed strategy which leads to the same utility as corresponding \( e = \langle c_e : e \in E \rangle \) can be constructed in polynomial time. Now we present a compact linear program CLP to compute the optimal marginal coverage.

\[
\text{CLP}(G, \ Y) : \quad \max_{c, z} U \tag{4}
\]
\[
U \leq -(1 - \min \{1, \sum_{e \in Y} c_e f_e \}) U(Y), \forall Y \in \Y \tag{5}
\]
\[
c_e = \sum_{k:G_k \in \Gamma(e)} z_k(e), \forall e \in G \tag{6}
\]
\[
\sum_{(v',v) \in G_k} z_k((v',v)) = \sum_{(v,v') \in G_k} z_k((v,v')), \forall G_k \tag{7}
\]
\[
\sum_{S_k \in S} \sum_{t \in T} z_k(\langle i, t_k \rangle) = m \tag{8}
\]
\[
\sum_{T \in T} z_k(\langle (i, t_k+\theta), T_k \rangle) = m \tag{9}
\]

Constraint (7) enforces conservation of flow, which is clearly satisfied by any mixed patrol strategy. Constraints (8) and (9) bound the total flow entering and exiting the transition graph by \( m \), the number of patrollers. Constraint (5) indicates that the attacker will best respond by choosing the strategy \( Y \) which leads to the best utility, or equivalently, the least utility for the defender \(- (1 - \min \{1, \sum_{e \in Y} c_e f_e \}) U(Y) \). This is an overestimate of the defender’s utility since the expression \( 1 - \min \{1, \sum_{e \in Y} c_e f_e \} \) is an overestimate of the probability that the attacker is detected. This is because \( \min \{1, \sum_{e \in E} c_e f_e \} \) only caps the expectation (over its pure strategies) of the detection probability at 1, but allows a pure strategy \( P = \langle P_i \rangle \) in its support to achieve \( \sum_{i=1}^m \sum_{e \in P_i} f_e > 1 \), whereas according to Eq.(1), the detection probability of each pure strategy should be at most 1. As a result, the solution of this LP provides an upper bound of the optimal defender utility. Fortunately, once we generate the patrols from the marginals we are able to compute the actual best-response utilities of the attacker. Our experiments show that the differences between the actual utilities and the upper-bounds given by the LP formulation are small.

5 CDOG Algorithm

The number of constraints in Eq.(5), being the same as the number of attacker strategies, increases exponentially with the game size. This leads to the poor scalability of the LP formulation. To deal with the scalability issue, we propose CDOG, a compact-strategy double-oracle algorithm on graphs. CDOG is based on the widely used double oracle framework (e.g., Jain et al., 2011)). The main idea is to find an equivalent small-size sub-game to avoid solving the original exponentially large game. Specifically, the framework starts from solving a sub-game involving only a very small subset of each player’s pure strategy set. The solution obtained, being an equilibrium of the sub-game, is not necessarily an equilibrium of the original game since the players may have incentive to deviate by choosing strategies not in the current sub strategy sets. Thus the framework expands the players’ strategy sets based on the current equilibrium and gets a larger sub-game. The process is repeated until no player can benefit from expanding their strategy sets. It usually ends with a sub-game with reasonable (instead of exponential) size, and the final solution is provably also an equilibrium to the original game (McMahan et al. 2003).

A traditional double oracle framework can take a long time to converge for large games, since sub-game is expanded slowly, while once it gets large, solving a sub-game is also very time-consuming. For example, Jain et al.’s algorithm [Jain et al., 2013] solves games with around 200 vertices, corresponding to a 10 zones, 20 time points case in our model, in around 9 hours. To scale it up, CDOG exploits the graph structure of our problem, uses a subgraph of the transition graph (instead of a pure strategy set) to characterize the defender’s strategy space, and solves each sub-game through compactly representing defender strategies as coverage on edges in the subgraph. The main structure of CDOG is depicted in Algorithm 1. Here Line 1 initializes the sub-game with a random subgraph \( G' \) of \( G \) and a random subset \( Y' \) of the attacker’s pure strategies. Using the LP formulation, Line 3 computes the equilibrium of the sub-game, where defender patrols on \( G' \) and attacker plays with strategies in \( Y' \). Notably, \( c = \langle c_e \rangle \) is the solution to LP, while \( y \) is an attacker mixed strategy over pure strategies in \( Y' \), which is obtained from the dual variable associated with Constraints in Eq.(5). Then through Lines 4–6, CDOG implements the core of dou-
ble oracle framework by calling two oracles—defender oracle (DO) and attacker oracle (AO)—to obtain the players’ best responses for expanding the sub-game. Next, we present in detail how these two oracles are implemented.

**Defender oracle.** The defender’s best response is an m-unit flow on G, which maximizes her utility against the current attacker strategy y. DO computes a compact representation of the best response pure strategy with the following MILP.

$$\max_{e,z} \sum_{Y \in \mathcal{Y}} -Y_i(1 - \min \{1, \sum_{e \in Y} c_e f_e \}) U(Y) \quad (10)$$

$$z_k(e) \in \{0, 1, \ldots, m\} \quad \forall z_k(e) \quad (11)$$

Constraints (6)-(9).

Constraint (11) ensures that the coverages on edges are integers, so that coverage vector c corresponds to a pure strategy P. Constraints (6)-(9) can be directly used here since the conservation of flow is the same for pure strategies as that for mixed strategies in CLP. The objective of DO is to maximize the defender’s expected utility given attacker’s mixed strategy y = \{Y_i\} where Y_i indicates the probability that the ith attacker pure strategy in Y’ is used. The value of Y_i comes from the dual variable of the ith inequality corresponding to Constraint (5) of CLP (G’, Y’). If the pure defender strategy corresponding to c computed by DO goes through edges not in G’, we expand G’ by including these edges and associated vertices.

**Attacker oracle.** The attacker oracle cannot be efficiently solved by an MILP since we cannot compactly represent the attacker strategies. AO computes the attacker’s best response through three steps: First, for each v \in V, we compute the attacker’s optimal path H(v) to v. Following H(v), the attacker is the least likely to be detected among all paths leading to v. Second, based on H(v), we compute the optimal time duration for the attacker to perform activities if he starts the activity at v. Finally, choose the vertex, path, and time duration which together lead to the optimal attacker utility. We now introduce the details of the three steps.

For step (1), let \gamma(v) = \sum_{e \in H(v)} c_e represent the probability of being detected if the attacker sails through path H(v). Apparently, if v = (i, t_k) and i \in Z^a, then the attacker can directly enter the transition graph at vertex v. Thus H(v) only consists of vertex v and \gamma(v) = 0. If i \notin Z^a, let \Lambda(v) represent the set of vertices v’ \in S such that (v’, v) is an edge in G, then the attacker has to arrive at some v’ and take edge (v’, v) to arrive at vertex v. Therefore, we have

$$\gamma(v) = \min_{v’ \in \Lambda(v)} \{1, H(v’) + c_{(v’, v)} f_{(v’, v)}\}. \quad (13)$$

Let \nu_{pre} = \arg\min_{v’ \in \Lambda(v)} \{H(v’) + c_{(v’, v)}\}, thus H(v) = H(\nu_{pre}) \cup \nu_{pre}, v, and H(v) can be computed by the recursive function shown in Algorithm 2. Note that to find H(v) for all vertices, every edge in the transition graph only needs to be visited at most once, thus the time complexity of step (1) is O(|E|).

For step (2), if the attacker starts to perform activity at vertex v = (i, t_k) and chooses a time duration l, the total probability of being detected depends on \gamma(v) and the probability of being deleted when performing the activity, i.e.,

**Algorithm 2: Find optimal path (c, v)**

1. Input: c, v; Output: H(v), \gamma(v);
2. if i \in Z^a then H(v) = v, \gamma(v) = 0;
3. else
   4. \Lambda(v) \leftarrow predecessors of v, pre \leftarrow -1, \min \leftarrow \infty;
   5. for v’ \in \Lambda(v) do
      6. if H(v’) + c_{(v’, v)} f_{(v’, v)} < min then
         7. min = H(v’) + c_{(v’, v)} f_{(v’, v)}; pre = v’;
   8. H(pre), \gamma(pre) \leftarrow Find optimal path (c, pre);
   9. H(v) \leftarrow H(pre) \cup (pre, v), \gamma(v) \leftarrow min;

\nu(v, l) = \min \{1, 1 - \gamma(v) - \sum_{j \in \{1, 2, \ldots, |E|\}} c_{e_j} f_{e_j}\}. Here e_j = \{(i, t_{k+1}), (i, t_{k+1})\}. The value of performing the activity is the sum of values of edges e_j, i.e., \ell(v, l) = \sum_{j \in \{1, 2, \ldots, \ell\}} \nu_{e_j}. Thus the expected utility is U(v, l) = (1 - \nu(v, l)) \cdot \ell(v, l). Therefore, the optimal time duration for vertex v is \nu^{opt}_{(v)} = \arg\max_{c \in (0, \infty)} U(v, l).

To find the optimal time duration for each vertex v = (i, t_k), we need to iterate all possible time durations for v, i.e., \{1, 2, \ldots, \tau - k\}. Thus the time complexity of Step (2) is O(\tau^2). The third step is to straightforwardly choose the vertex v leading to the optimal attacker utility, i.e., v = \arg\min_{v \in S} U(v, l^opt_{(v)}), and construct the attacker strategy Y based on H(v) and l. If strategy Y is not in the current set Y’, we expand Y’ by adding Y.

### 6 Experimental Evaluation

We evaluate the proposed algorithms in terms of (1) solution quality, (2) scalability, and (3) robustness. The CLP is solved by Knitro 9.0.0. Each point in the figures is the average value over 30 sample games. We test the algorithms on graphs based on the geography of the Great Barrier Reef Marine Park as is shown in Figure 1. Given that the MPA can be divided differently due to different purposes [Watts et al., 2009], in each game in the experiments, we generate an MPA graph based on a random division of the park. We randomly generate the time needed to move between adjacent zones, i.e., d_{ij}, in [1, \frac{t}{2}], where t is the number of time points in the game. Transition graphs are then constructed based on the MPA graphs and d_{ij}s, in which each zone has \frac{t}{2} copies. We randomly choose the value of each stay edge in (0, 100).

**Solution quality.** Solution quality of algorithms is measured by attacker utility. Given the zero sum assumption, higher attacker utility indicates lower defender utility. We compare our algorithms with two baseline algorithms ANP and AND. ANP assumes that the attacker does not take paths, but directly attacks anywhere at any time instantaneously, which is similar as in Fang et al. [2013]. AND assumes that the attacker can take paths in the graph to arrive at a target, but still attacks instantaneously, which is similar as in Yin et al. [2012]. The baseline algorithms are under extra assumptions since no previous algorithms can exactly solve our problems. We assume 3 patrollers, 9 zones and divide the timeline into 12 points unless otherwise specified.

In Figures 4(a), 4(b) and 4(c), the y-axis indicates the attacker utility, while the x-axis indicates the patrol duration of
a defender’s resource (i.e., the value of $\theta$), the number of starting zones for the attacker, and the number of starting zones for the defender respectively. Since CLP and CDOG lead to the same attacker utility, their results are represented by one single bar. The solution quality of CLP and CDOG is significantly better than that of ANP and AND despite the value of the three parameters. It is unsurprising that as the patrol duration increases (Figure 4(a)) or the number of defender’s starting zones increases (Figure 4(b)), the attacker utilities computed by all algorithms decrease. As the number of attacker’s starting zones increases (Figure 4(c)), ANP’s performance is not affected since it does not consider attacker’s paths, while other algorithms show an increasing trend in attacker utilities.

Figure 4(d) depicts the percentage of the true optimal defender utility v.s. the theoretical upper bound returned by CLP and CDOG. The x-axis indicates the maximum value of the detecting factor $f_e$, i.e., 0.3 indicates that $f_e$ is randomly chosen in $(0, 0.3]$. Eq.(1) indicates that with smaller $f_e$, CLP and CDOG are less likely to overestimate the detection probability. Figure 4(d) shows a decreasing trend in the percentage of the true defender utility v.s. CLP and CDOG’s results. Fortunately, even when $f_e = 1$, the percentage is still around 90%, indicating that the upper bound computed by CLP and CDOG is very close to the true utilities.

Scaling.

In Figures 5(a) and 5(b), the y-axis indicates runtime while the x-axis respectively indicates the number of zones and time points. In both figures, the runtime of CLP shows a much more obvious increasing trend. Actually, CLP cannot solve games with more than 20 targets and 20 time points due to the RAM limit, while CDOG can solve games with 40 targets in around 6 minutes and games with 50 time points in less than 2 minutes. We also evaluate the runtime of the defender oracle, attacker oracle, and CLP in CDOG in detail. Figure 6 shows an example of the runtime of the three parts in the CDOG algorithm, which solves a game after 10 iterations. The runtime of the CLP shows an increasing trend. The size of the DO and the AO is barely affected by the iteration, thus their runtime does not change much.

Robustness.

We first consider observation noise of the attacker. We add 0-mean Gaussian noise with standard deviations chosen randomly from $U[0, 0.5]$ to the coverage on each edge observed by the attacker. Figure 5(c) shows the attacker utilities considering observation noise for the same class of games considered in Figure 4(a). Compared with Figure 4(a), all algorithms lead to lower attacker utilities in Figure 5(c) since the observation noise prevents the attacker from responding the best. CLP and CDOG still significantly outperform ANP and AND. We also consider payoff noise of the defender. For each stay edge in the transition graph, we add the same Gaussian noise as the previous setting to the defender’s knowledge on the value of the edge. Figure 5(d) considers the class of games in Figure 4(a) with payoff uncertainties. Compared with Figure 4(a), all algorithms lead to higher attacker utilities in Figure 5(d) since the payoff noise affects the defender’s judgement on the attacker’s action. The advantage of CLP and CDOG over ANP and AND is still significant.

Figure 4: Solution quality

Figure 5: Scalability and robustness

Figure 6: Runtime (ms) of CLP and oracles in CDOG

7 Conclusion

This paper models the problem of patrolling MPAs to protect coral reef ecosystems as a defender-attacker Stackelberg game, in which both players’ strategies are time-dependent paths, and the payoffs are affected by the duration of the attack. We propose a linear program (CLP) to solve the game, in which defender strategies are compactly represented as flows on graphs. We also propose a more scalable algorithm, CDOG, which combines techniques of compactly representing defender strategies and incrementally generating strategies. Experimental results show that our algorithms lead to significantly better solution quality than that of baseline algorithms, and the CDOG algorithm can scale up.
References


