Combining Graph Contraction and Strategy Generation for Green Security Games

Anjon Basak[1] Fei Fang[2] Thanh Hong Nguyen[2] and Christopher Kiekintveld[1]

[1] abasak@miners.utep.edu, cdkiekintveld@utep.edu
University of Texas at El Paso,
500 W University Ave, El Paso, TX 79902
[2] feifang@usc.edu, thanhhng@usc.edu
University of Southern California, 941 Bloom Walk, SAL 300
Los Angeles, CA, 90089

Abstract. Many real-world security problems can be modeled using Stackelberg security games (SSG), which model the interactions between a defender and attacker. *Green security games* focus on environmental crime, such as preventing poaching, illegal logging, or detecting pollution. A common problem in green security games is to optimize patrolling strategies for a large physical area such as a national park or other protected area. Patrolling strategies can be modeled as paths in a graph that represents the physical terrain. However, having a detailed graph to represent possible movements in a very large area typically results in an intractable computational problem due to the extremely large number of potential paths. While a variety of algorithmic approaches have been explored in the literature to solve security games based on large graphs, the size of games that can be solved is still quite limited. Here, we introduce abstraction methods for solving large graph-based security games and integrate these methods with strategy generation techniques. We demonstrate empirically that the combination of these methods results in dramatic improvements in solution time with modest impact on solution quality.

Keywords: security, green security, abstraction, contraction, game theory

1 Introduction

We face many complex security threats with the need to protect people, infrastructure, computer systems, and natural resources from criminal and terrorist activity. A common challenge in these security domains is making the best use of limited resources to improve security against intelligent, motivated attackers. The area of *green security* focuses on problems related to protecting wildlife and natural resources against illegal exploitation, such as poaching and illegal logging. Resource limitations are particularly acute in fighting many types of environmental crime, due to a combination of limited budgets and massive areas that need surveillance and protection. For example, it is common for small numbers of rangers, local police, and volunteers to patrol protected national parks that may cover thousands of square miles of rugged terrain [22].

Work on *green security games* [7, 11] has proposed formulating the problem of finding optimal patrols to prevent environmental crime as a Stackelberg security game [25]. In these games, the defender (e.g., park ranger service) must decide on a randomized strategy for patrolling the protected area, limited by the geographic constraints and the number of available resources. The attacker (e.g., poacher) selects an area of the park to attack based on the intended target and knowledge of the typical patrolling strategy (e.g., from previous observations and experience). Green security games are used to find randomized patrolling strategies that maximize environmental protection given the resources available.

Green security games typically model the movement constraints for the defender patrols using a graph representing the physical terrain. Unfortunately, this leads to a major computational challenge because the number of possible paths for the defender grows exponentially with the size of the graph. Enumerating all possible combinations of paths for multiple resources makes the problem even more intractable [29, 35]. Several algorithms have been proposed in the literature to solve these games more efficiently [24, 28]. Most of these rely on incremental strategy generation (known as double oracle algorithms, or column/constraint generation) to solve an integer programming formulation of the problem without enumerating the full strategy space. The most recent application called PAWS [10] approaches the scalability issue by incorporating cutting plane and column generation techniques.

Here, we take a new approach that *combines* strategy generation methods with automated game abstraction methods based on graph contraction. The idea of using automated abstraction has been very successful in solving other types of very large games, such as computer poker [16, 17, 19, 20, 40]. The basic idea of our game abstraction is motivated by graph contraction techniques used to speed up pathfinding and other computations on graphs. When we apply graph contraction to a green security game, it dramatically reduces the strategy space for the defender, leading to lower solving time. To improve scalability even further we integrate graph contraction with strategy generation to create a new class of algorithms capable of solving very large green security games. We evaluate our new algorithms on graph-based security games motivated by the problems encountered in green security domains, including some based on real world data sets. The experiments show that we can dramatically improve solution times by using abstraction in combination with strategy generation, leading to high-quality approximations within seconds even for graphs with a thousand nodes.

2 Related Work

The first approach to compute security resource allocations was to find a randomized strategy after enumerating all possible resource allocations [29], which is used by the Los Angeles Airport Police in an application called ARMOR [30]. A more compact form of security game representation was used [25] to develop a faster algorithm (IRIS [35]), which is used for scheduling by the Federal Marshal Service (FAMS). ASPEN [24] was introduced to deal with the exponential size of games with complex scheduling constraints by using a branch-and-price approach. Most recently, to tackle more massive games an approach based on cutting planes was introduced [38] to make

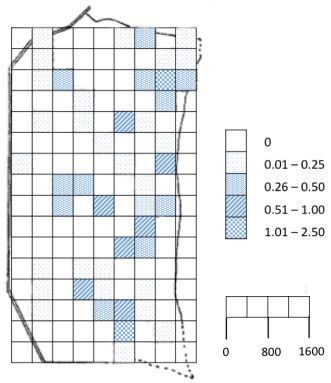
the solution space more manageable. Game theoretic algorithms are also used to secure ports [32] and trains [39]. Recently, successful deployment of game theoretic applications motivated researchers to use game theory in green security domains [7, 21, 37]. This led to new game model called GSG [11]. Assumptions about the attacker being able to fully observe the defender strategy can be unrealistic in some cases, so partial observability and bounded rationality have been introduced to make the attacker model better fit the practice. Defender payoff uncertainty has also been addressed with these issues in an algorithm called ARROW [28]. Despite the models and algorithms introduced, how to handle the large strategy space in GSGs remains a challenge. In this paper, we introduce abstraction techniques to address this problem. Many abstraction techniques have been developed for extensive form games with uncertainty including both lossy [31] and lossless [18] abstraction. There has been some work which gives bounds on the error introduced by abstraction [26]. There are also imperfect recall abstractions that consider hierarchical abstraction [8] and Earth mover's distance [13].

Graph contraction techniques [14] have been used to achieve fast routing in road networks, where contraction acts as a pre-processing step. This method has been improved using fast bidirectional Dijkstra searches [34, 15]. A time-dependent contraction algorithm has also been introduced for time-dependent road networks [5]. Graph contraction has also been used in imperfect information security games with infinite horizon where the area is patrolled by a single robot [4]. In this paper, we leverage insights from graph contraction to handle the large strategy space in GSGs. Another recent closely related work [23] uses cut-based graph contraction and also column generation approach for restricting the strategy space, but for a different type of security model based on checkpoint placement for urban networks.

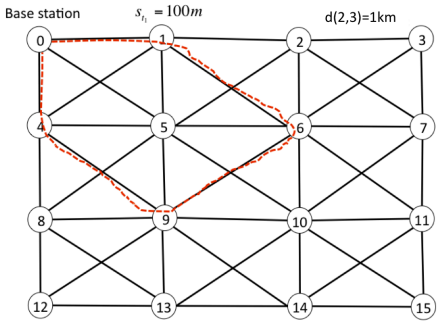
3 Domain Motivation

Illegal activities such as poaching pose a major threat to biodiversity across all types of habitats, and many species such as rhinos and tigers. A report [1] from the Wildlife Conservation Society (WCS) on May 2015 stated that the elephant population in Mozambique has shrunk from $20,000$ to $10,300$ over the last five years. Elephants were recently added to the IUCN Red List [2]. Marine species also face danger due to illegal fishing and overfishing, causing harm to the people of coastal areas who depend on fishing for both sustenance and livelihood. According to World Wide Fund for Nature (WWF), the global estimated financial loss due to illegal fishing is $\$23.5$ billion [3]. Organizations like WCS are studying strategies for combating environmental crime that include patrols of both land and sea habitats to detect and deter poaching. PAWS [10] is a new application based on green security games that helps to design patrolling strategies to protect wildlife in threatened areas. The area of interest is divided into grid cells that capture information about the terrain, animal density, etc. Each grid cell is a potential target for the poachers. The patroller plans a route to protect the targets along a path. However, if the grid cell is too large (e.g., 1km by 1km) or the terrain is complex, it is very difficult for the patroller to patrol even a single grid cell without any detailed path provided in the cell. Therefore, a fine-grained discretization is often required, leading to a large number of targets and a exponential number of patrol routes that existing solvers

cannot handle. PAWS handles this problem by pre-defining a limited set of routes based on domain knowledge of features like ridgelines and streams, which can be found based on elevation changes. We also observe that in many green security domains, there is a high variance in the importance of the targets. For example, Figure 1(a) shows the mean number of elephants in each area of a grid representing the Queen Elizabeth National Park in Uganda [12]. There are many cells that have no animal count at all, and if there is minimal activity it is very inefficient to consider these areas as targets to patrol (or poach). This motivates our abstraction-based approach to make it computationally feasible to directly analyze high-fidelity maps for green security without preprocessing.



(a) Mean numbers of elephants/$0.16km^2$ in Queen Elizabeth National Park, Uganda



(b) A graph representation of a grid-based GSG (a patrolling path is shown in red).

Fig. 1. Domain example and game model.

4 Game Model and Basic Solution Technique

A typical green security game (GSG) model is specified by dividing a protected wildlife area into grid based cells, as shown in Figure 1(a). Each cell is considered a potential target t_i where an attacker could attempt a poaching action. We transform this grid-based representation into a graph as shown in Figure 1(b). Each node represents a target t_i.

Definition 1. *A GSG Graph is a graph $G = (V, E)$ where each node $t_i \in V$ is associated with a patrolling distance s_{t_i} and each edge $e_{ij} \in E$ is associated with a traveling distance $d(i, j)$. There exists a base node $B \in V$. A feasible patrolling path is a sequence of consecutive nodes that starts and ends with B, with a total distance that does not exceed the distance limit d_{max}.*

For example, in Figure 1(b), $s_{t_1} = 100m$. This means that to protect target t_1, the patroller needs to patrol for a distance $100m$ within target t_1. $d(2, 3) = 1km$ indicates

the distance from target t_2 to t_3. The defender patrols to protect every target on the patrolling path. Therefore, the total distance of a path is the sum of patrolling and travel distance. Typically the patrol starts in a base station and ends in the same base station. For example, a patrolling path is shown in Figure 1(b) where the patrol starts at t_0 and traverses through targets $t_1 \rightarrow t_6 \rightarrow t_9 \rightarrow t_4$ and ends back in target t_0.

The defender has a limited number of resources R, each of which can be assigned to at most one patrolling path that covers a set of targets $t \in T$. So the defender's pure strategies are the set of joint patrolling paths $J_m \in J$. Each joint patrolling path J_m assigns each resource to a specific path. We denote a patrolling path by p_k and the base target by t_b. The length of p_k is constrained by d_{max}.

We use a matrix $P = P_{J_m t} = (0,1)^n$ to represent the mapping between joint patrolling paths and the targets covered by these paths, where $P_{J_m t}$ represents whether target t is covered by the joint patrolling path J_m. We define the defender's mixed strategy x as a probability distribution over the joint patrolling paths J where x_m is the probability of patrolling a joint patrolling path J_m. The coverage probability for each target is $c_t = \sum_{J_m} P_{J_m t} x_m$.

If target t is protected then the defender receives reward $U_d^c(t)$ when the attacker attacks target t, otherwise a penalty $U_d^u(t)$ is given. The attacker receives reward $U_a^u(t)$ if the attack is on an area where the defender is not patrolling, or penalty $U_a^c(t)$ if the attack is executed in a patrolled area. These values can be based on the density of the animals in the area attacked, as a proxy for the expected losses due to poaching activities. We focus on the zero-sum game case where $U_d^c(t) = U_a^c(t) = 0$ and $U_d^u(t) = -U_a^u(t)$. In the rest of the paper, we also refer to $U_a^u(t)$ as the utility of target t.

We use the Stackelberg model for GSG. In this model, the patroller, who acts as defender, moves first and the adversary observes the defender's mixed strategy and chooses a strategy afterwards. The defender tries to protect targets $T = t_1, t_2, ..., t_n$ from the attackers by allocating R resources. The attacker attacks one of the T targets. We focus on the case where the attacker is perfectly rational and compute the Strong Stackelberg Equilibrium (SSE) [27, 6, 36], where the defender selects a mixed strategy (in this case a probability distribution x over joint patrolling paths J_m), assuming that the adversary will be able to observe the defender's strategy and will choose a best response, breaking ties in favor of the defender. Given a defender's mixed strategy x and the corresponding coverage vector c, the expected payoff for the attacker is

$$U_a(c,t) = \max_{t \in T} \{(1 - c_t) U_a^u(t)\} \tag{1}$$

It is possible to solve this problem by enumerating all feasible joint patrolling paths [24]. In the case of zero-sum games, the optimal patrolling strategy for the defender can be determined by solving the following linear program (LP).

$$\min_{x,k} \quad k \tag{2}$$

$$(1 - Px) U_a^u \leq k \tag{3}$$

$$\sum\nolimits_i x_i \leq 1 \tag{4}$$

$$x \geq 0 \tag{5}$$

Equation 2 represents the objective function, which minimizes the expected payoff for the attacker, or equivalently, maximizes the expected payoff for the defender. Constraint 4 makes sure that the probability distribution over the joint patrolling paths does not exceed one. The solution of the LP is a probability distribution x over the joint patrolling paths J, and this is the strategy the defender commits to. The attacker will choose the target with highest expected utility, as shown in Constraints 3. This formulation does not scale well to large games due to the exponential number of possible joint paths as the graph grows larger.

5 Solving GSG with Abstraction

Our approach combines the key ideas in double oracle methods and graph contraction. There are often relatively few important targets in a GSG. For example, the key regions of high animal density are relatively few, and many areas have low density, as shown in Figure 1(a). This suggests that many targets in the game can be removed to simplify the analysis while retaining the important features of the game.

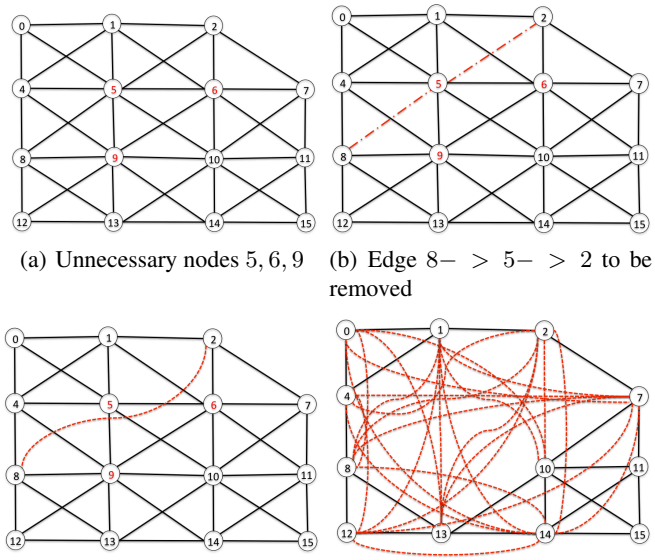
We describe our approach in three stages. First, we describe our method for contracting a graph by removing nodes and calculating a new set of edges to connect these nodes that retains the shortest path information. This contracted graph can be solved using any existing algorithm for GSG; as a baseline, we use the LP on the full set of paths. Second, we describe a single-oracle approach for finding the set of targets that must be included in the contracted game. This method restricts the set of targets to a small set of the highest-valued targets, and iteratively adds in additional targets as needed. Finally, we describe the double-oracle algorithm. This uses the same structure as the single oracle, but instead of solving each restricted game optimally, we restrict the defender's strategy space and use heuristic oracles to iteratively generate paths to add to the restricted game.

5.1 Graph Contraction

We first describe how we construct an abstracted (simplified) graph for a restricted set of target nodes. Essentially, we remove all of the nodes except the restricted set, and then add additional edges to make sure the shortest paths are preserved.

Many graph contraction procedures used in pathfinding remove nodes one by one, but we use a contraction procedure that removes the nodes in one step. Suppose we have decided to remove the set of nodes $T_u \in T$. We find all the neighbors of set T_u, denoted as V. Next we try to find the shortest paths between each pair of nodes $(v_i, v_j) \in V$ that traverse through nodes T_u where v_i and v_j are not adjacent. We use Floyd-Warshall algorithm [9] to find the shortest paths for all the nodes in V using only nodes T_u. If the length of the shortest path does not exceed d_{max}, we add an edge (v_i, v_j) in the contracted graph, with distance equals the length of the shortest path.

Theorem 1. *The contraction process described in Algorithm 1 preserves the shortest paths for any pair of nodes that are not removed in the original graph. Formally, given a graph $G = (T, E)$ and a subset of nodes T_u, Algorithm 1 provides a contracted*



(a) Unnecessary nodes $5, 6, 9$ (b) Edge $8- > 5- > 2$ to be removed

(c) New shortcut path $8- > 2$ (d) Final graph after contraction of node $5, 6, 9$

Fig. 2. Instant Contraction procedure for different nodes

graph $G' = (T \setminus T_u, E')$ *and the length of the shortest path for any pair of nodes* $(v_i, v_j) \in T \setminus T_u$ *in* G' *is the same as in* G*.*

Proof sketch: First, $\forall (v_i, v_j) \in T \setminus T_u$, the shortest path in G' can be easily re-mapped to a path in G, and thus is a candidate for the shortest path in G. Therefore, the shortest path in G' is no shorter than that in G. Second, $\forall (v_i, v_j) \in T \setminus T_u$, it can be shown that the shortest path in G can also be mapped to a path in G'. Let $P = t_1 \rightarrow t_2 \rightarrow ... \rightarrow t_K$ be the shortest path between v_i and v_j in G ($t_1 = v_i$, $t_K = v_j$). Let t_{k1} and t_{k2} be any two nodes in P such that $t_{k1} \in V$, $t_{k2} \in V$ and $t_k \in T_u, \forall k1 < k < k2$. Then $\bar{P} = t_{k1} \rightarrow t_{k1+1} \rightarrow \ldots \rightarrow t_{k2}$ has to be a shortest path linking t_{k1} and t_{k2}. Since t_{k1} and t_{k2} are in V and $\bar{P}$ only traverses through nodes in T_u, an edge (t_{k1}, t_{k2}) with the same length of $\bar{P}$ is added to G' according to Algorithm 1. Therefore, P can be mapped to a path P' in G' with the same length. As a result, the shortest path in G is no shorter than that in G'. Combine the two statements, the length of the shortest path for any pair of nodes $(v_i, v_j) \in T \setminus T_u$ in G' is the same as in G. □

Figure 2 shows how the contraction works. Figure 2(a) shows the removed nodes $T_u = (5, 6, 9)$. The neighbor set of T_u is $V = (0, 1, 2, 4, 7, 8, 10, 12, 13, 14)$. For convenience we show a breakdown of the step in Figure 2(b) where the edge $(8 \rightarrow 5 \rightarrow 2)$ is shown and in Figure 2(c) where the edge $(8 \rightarrow 5 \rightarrow 2)$ is replaced with shortcut $8 \rightarrow 2$. Figure 2(d) shows the final stage of the graph after contracting nodes $5, 6, 9$. Algorithm 1 shows pseudocode for the contraction procedure.

Algorithm 1 Instant Contraction Procedure

procedure INSTANTCONTRACTGRAPH ▷
  $G \leftarrow Graph()$ ▷ Initiate the graph to contract
  $n_d \leftarrow ContractedNodes()$ ▷ Get the nodes to contract
  $n_{nei} \leftarrow ComputeNeighbors(n_d)$
  $apsp \leftarrow AllPairShortestPath(G, n_d, paths)$
  for $v \leftarrow neighbors.pop()$ **do**
    for $v' \leftarrow neighbors.pop()$ **do**
      if $v \neq v^{'}$ $\&notadjacent(v, v')$ **then**
        $d \leftarrow apsp[v][v']$
        $path \leftarrow getPath(paths, v, v')$
        if $d \leq dmax$ **then** ▷ if d is less than the distance limit
          $UpdateNeighbors(v, v', path, d)$
          $v.AddNeighbor(v', path)$
          $v'.AddNeighbor(v, path)$
  $RemoveAllContractedNodes(G, n_d)$

Reverse Mapping When we solve a GSG with a contracted graph (e.g., using the standard LP), the paths found in the solution must be mapped back to the paths in the original graph so they can be executed. This is because a single edge in the abstract path can correspond to a path of several nodes in the original graph. In algorithm 1, when the contracted graph is constructed, the corresponding path in the original graph of each edge being added is already recorded, and is the basis the reverse mapping.

5.2 Single-Oracle Algorithm Using Abstraction

Algorithm 2 Single Oracle With Abstraction (SO)

Input: original graph G, target utility $U_i, \forall i \in V$
Output: defender mixed strategy x and coverage vector c

$\bar{T}$=GreedyCoverR(G) ▷ Find initial set of targets to be considered in the restricted graph
Set current graph $G_c = G$
repeat
  $G_t =$`Contract`$(G_c, \bar{T})$ ▷ Contract graph
  $(u, x_t, c_t) = Solve(G_t)$ ▷ Solve restricted graph, get attacker's expected utility u, defender strategy x_t, coverage vector c_t
  $v = AttEU(G_c, c_t)$ ▷ Calculate actual attacker's expected utility on current graph
  if $v == u$ **then**
    Break
  $G_c =$`ContractWithThreshold`(G_c, u) ▷ Remove targets with utility $< u$
  if G_c is small enough **then**
    $(u, x, c) = Solve(G_c)$ ▷ Solve G_c directly
    Break
  Add at least one additional target into $\bar{T}$
until $1 < 0$

We begin by describing a basic "single oracle" algorithm that restricts only the attacker's strategy space (i.e., the number of targets). The basic observation that leads to this approach is based on the notion of an attack set. In the Stackelberg equilibrium solution to a security game, there is a set of targets that the attacker is willing to attack; this is the set that the defender must cover with positive probability. All other target have maximum payoffs lower than the expected payoff for the attacker in the equilibrium solution, so the attacker will never prefer to attack one of these targets, even though it is left unprotected. If we could determine ahead of time which set of targets must be covered in the solution, we could simply apply graph contraction to this set of targets, solve the resulting game, and be guaranteed to find the optimal solution.

Our approach is to start by considering only a small set of targets $\bar{T}$, perform contraction, and solve the abstracted game for this small set of targets. If the attacker expected value in the solution is lower than the value the attacker can get from attacking the best target that was not included in the restricted game, we add at least one (and possible more than one) additional target to the restricted game and repeat the process. Targets are added in decreasing order of the attacker's payoff for attacking the target if it is not protected at all. If we solve a restricted game and the attacker's expected value is greater than the unprotected values of all remaining targets, we can terminate having found the correct attack set and the optimal solution.

The initial set of targets to be considered is determined by *GreedyCoverR* (GCR). First consider the case where there is only one patroller. We use an algorithm named GC1 to find a greedy patrolling path. GC1 greedily inserts targets to the path and asks the patroller to take the shortest path to move from one target to the next target. The targets are added sequentially in a descending order of the target utility. GC1 terminates when the distance limit constraint is violated. GCR calls GC1 R times to find greedy paths for R patrolling resources. If the greedy paths can cover the top K targets, GCR returns the set of targets whose utility is no less than the utility of the $(K+1)^{th}$ target. This is because a restricted graph with the top K targets can be perfectly protected given the greedy paths, and therefore the patroller can try to protect more targets.

Algorithm 2 shows psuedocode for this procedure. Clearly, u is non-decreasing and v is non-increasing with each iteration. For a value of u in any iteration, we can claim that any target whose utility is smaller than u can be safely removed as those targets will never be attacked (attacker will not deviate if those targets are added to the small graph). The function $\texttt{Contract}(G, \bar{T})$ completes two tasks. First, it removes targets that are not in $\bar{T}$, and second, refine the graph by removing dominated targets. In each iteration, u provides a lower bound of the attacker's expected utility in the optimal solution (optimal defender strategy) and v provides an upper bound. If $v == u$, it means current solution is the optimal. Line 13 adds at least one target to the set $\bar{T}$. Figure 3 illustrates the algorithm on an example graph. Figure 3 illustrates Algorithm 2 with an example.

5.3 Double Oracle Graph Contraction

The single oracle methods can prevent us from having to solve the full graph with the complete set of targets. However, it still assumes that we use an exact, exhaustive method to solve the smaller abstracted graphs. For very large problems, this may still

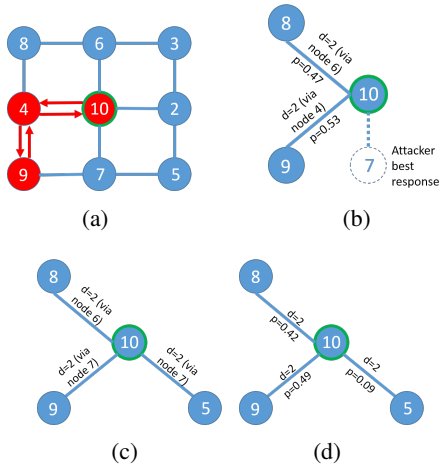


Fig. 3. Example of Single-oracle Algorithm. The numbers shown in the nodes represent the index and the utility of the target. Node 10 is the base node and the defender has only one patrol resource. 3(a): Original graph (distance limit= 4), which is also the initial current graph G_c. Red lines indicate the greedy route, which determines $\bar{T} = \{10, 9, 8\}$. 3(b): First restricted graph G_t and the corresponding optimal defender strategy (taking the route $10 \rightarrow 8 \rightarrow 10$ with probability 0.47), which leads to $u = 4.23$ and $v = 7$. 3(c): Updated current graph G_c, which is achieved by removing all nodes with utility $\leq u$ (i.e., nodes 2,3,4) and then removing dominated targets (node 7 is dominated by node 9 and node 6 is dominated by node 8). 3(d): Second restricted graph G_t with updated $\bar{T} = \{10, 9, 8, 5\}$, which leads to u=v=4.58 and the termination of the algorithm.

be too slow and use too much memory. To address this we introduce the Double Oracle method that also restricts the defender's strategy space when solving the abstracted graphs. This basic idea (a version of column generation) has been widely used in security games literature [24, 33]. Algorithm 3 outlines the procedure.

The outer loop is based on the single oracle method, and gradually adds targets to the restricted set. However, each time we solve the problem for a new contracted graph, we also start from a restricted set of possible paths for the defender. We then solve the "Master" problem (i.e., the original LP), but only with this restricted set of paths. If the solution to this restricted problem already implies that we need to add more targets (because the attacker's payoff is lower than the next best target), we do so and start over with a new, larger contracted graph. Otherwise, we solve a "Slave" problem to find at least one new path to add to the restricted problem, and then go back to solve the Master again. This process terminates when we cannot add any additional paths to the Master that would improve the payoff for the defender (and lower it for the attacker).

To guarantee that we have found the optimal solution, the slave should always return a new path to add that has the minimum reduced cost. The reduced cost of a new joint path J_m is $r_{J_m} = -\sum_i y_i U_a^u(i) P_{J_m,i} - \rho$, where y_i refers to the dual variable of the i^{th} constraint in the original LP (3), and ρ is the dual variable of constraint 4. The joint path with the most negative reduced cost improves the objective the most. If the reduced cost of the best new joint path is non-negative, then the current solution is optimal. In fact,

Algorithm 3 Double Oracle With Abstraction (DO)

Input: original graph G, target utility $U_i, \forall i \in V$

Output: defender mixed strategy x and coverage vector c

1: Sort the targets according to attacker's reward T_{srt}=`sortTargets()`
2: Get the list of initial targets using GCR from T_{srt}, T_{cur} = `GreedyCoverR()`
3: **repeat**
4: Set temporary graph where G_t and all targets $t_i \in G_t$ is also in T_{cur}
5: Generate initial set of paths using GreedyPathR, $s_{cur} = GPR(G_t)$
6: **repeat**
7: Solve SSG for G_t, get mixed strategy x_t, coverage vector c_t, and attacker's expected utility $u = AttEU(G_t, c_t)$
8: Calculate actual attacker's expected utility on original graph $v = AttEU(G, c_t)$
9: **if** $u < v$ **then**
10: Break
11: Generate paths using $s_t = GreedyPathR()$
12: Append paths $s_{cur} = s_{cur} U s_t$
13: **if** $s_t == 0$ **then**
14: Break
15: **until** $1 < 0$
16: Find attack target in G $attackTarget(G, c_t)$
17: Add next n e.g. $n = 5$ targets to T_{cur} from $T_{srt} - T_{cur}$
18: **until** $u >= v$ and no more path can be added to s_{cur}

finding the joint path with the lowest reduced cost is equivalent to solving the following combinatorial optimization problem:

Definition 2. *In the coin collection problem, a GSG graph $G = (V, E)$ is given, and each node t_i is associated with a number of coins, denoted as Y_i. When a node is covered by a patrolling path, the coins on the node will be collected and can be collected at most once. The goal is to find a feasible joint path that collects the most number of coins.*

When $Y_i = y_i U_a^u(i)$, the optimal solution of the coin collection problem is the joint path with the lowest reduced cost. The coin collection problem is NP-hard based on a reduction form the hamiltonian cycle problem (details omitted for space). Designing efficient algorithms for finding the optimal or a near-optimal solution of the coin collection problem can potentially improve the scalability of using the double oracle method to find the exact optimal solution to GSG. However, here we are interested in maximizing scalability for the DO approach combined with abstraction, so we designed heuristic methods for the slave that are very fast, but will not necessarily guarantee the optimal solution. More specifically, we use Algorithm 4 as a heuristic approach for solving the coin collection problem.

6 Experimental Evaluation

We present a series of experiments to evaluate the computational benefits and solution quality of our solution methods. We begin by evaluating the impact of abstraction in

Algorithm 4 GreedyPathR (GPR)

procedure GREEDYCOVER-COINCOLLECTION
  Initialize best joint path set J_{best}
  for $iter = 0$ to 99 **do**
    if $iter == 0$ **then**
      $Tlist \leftarrow sort(T \setminus B, Y)$ ▷ Get a sorted list with decreasing Y_i
    else
      $Tlist \leftarrow shuffle(T \setminus B)$ ▷ Get a random ordered list
    $Y_r \leftarrow Y$ ▷ Initialize the coins remained
    for $j = 1$ to R **do**
      Initialize the current patrol route Q_j
      for each target t_i in $Tlist$ with $Y_r(i) > 0$ **do** ▷ Check all uncovered targets
        Insert t_i to Q_j while minimizing the total distance
        if total distance of Q_j exceeds d_{max} **then**
          remove t_i from Q_j
      for each target t_i in Q_j **do**
        $Y_r(i) = 0$
    if $\{Q_1, ..., Q_R\}$ collects more coins than J_{best} **then**
      update J_{best}
  return J_{best}

isolation, and then provide a comparison of many different variations of our methods on synthetic game instances. Finally, we test our best algorithms on large game instances using real-world data, demonstrating the ability to scale up to real world problems.

6.1 Graph Abstraction

We begin by isolating the effects of abstraction from the use of strategy generation (using either the single or double-oracle framework). The baseline method solves a graph-based security game directly using the standard optimization formulation, enumerating all joint patrolling paths directly on the full graph. We compare this to first applying our graph abstraction method to the game, and then using the same solver to find the solution to the abstracted graph. We compare the methods on both solution quality and runtime. To measure the amount of error introduced we introduce an error metric denoted by epsilon(ϵ) $= \frac{[U_d(c,a) - U'_d(c,a)]}{U_d(c,a) * 100}$, where $U'_d(c, a)$ is the expected payoff for defender when using contraction and $U_d(c, a) \geq U'_d(c, a)$.

For our experiments we used 100 randomly generated, 2-player security games intended to capture the important features of green security games. Each game has 25 targets (nodes in the graph). Payoffs for the targets are chosen uniformly at random from the range -10 to 10. The rewards for the defender or attacker are positive and the penalties are negative. We set the distance constraint to 6. In the baseline solution the is no contraction. For different levels of abstraction the number of contracted nodes (#CN) varies between the values: $(0, 2, 5, 8, 10)$. Figure 4 shows us how contraction affects contraction time (CT), solution time (ST) and reverse mapping time (RMT). CT only consider the contraction procedure, ST considers the construction of the P matrix

and the solution time for the optimization problem, and RMT considers time to generate the P matrix for the original graph from the solution to the abstracted game.

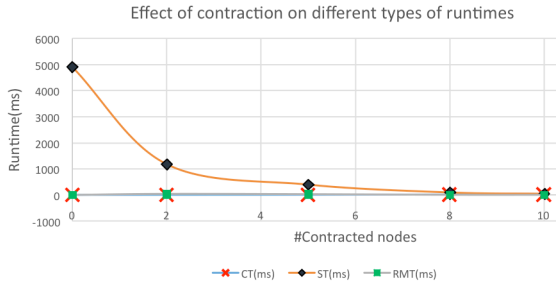


Fig. 4. Effect of contraction on times CT, ST and RMT

We first note that as the graph becomes more contracted ST takes much less time, as shown in Figure 4. The next experimental result presented in Figure 5 shows how much error is introduced as we increase the amount of contraction and the amount of time we can save by using contraction.

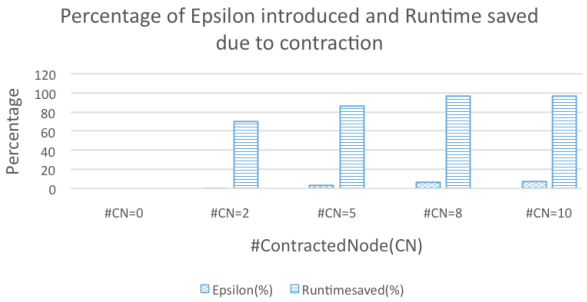


Fig. 5. Effect of contraction on Epsilon and runtime saved

6.2 Comparison of Solution Algorithms

We now present results comparing the solution quality and runtimes of different versions of our solution algorithm on graph-based security games of increasing size. We focus on grid-based graphs, which are typical of open-area patrolling problems like those in wildlife protection domains. For the experiments we generated 20 sample games for each size of game. For simplicity, the distance between every node and it's neighbors is set to 1. The patroller has two resources to conduct patrols in each case, and the distance constraint on the paths varies depending on the game size.

All of the games are zero-sum. We randomly assign payoffs to the targets. In wildlife protection, it is typical for there to be a relatively small number of areas with high densities of animal/poaching activity. To reflect this low density of high-valued targets, we partition the targets into high and low value types, with values uniformly distributed in the ranges of [0, 4] and [8, 10], respectively. We assign 90% of the targets values from the low range, and 10% values from the high range.

We break down the runtime into three different components: 1) The time to contract graphs, ContractionTime (CT), 2) The time to solve optimization problems, SolvingTime (ST), and 3) the total runtime, TotalTime (TT). All runtimes are given in milliseconds. EPd denotes the expected payoff for defender.

We compare out methods to two baselines that solve the original optimization problem with no contraction by enumerating joint patrolling paths. The first one enumerates all paths and directly solves the problem, while the second algorithm uses column generation to iteratively add joint paths (but does not use contraction). All algorithms that use the path sampling heuristic generate 1000 sample paths. We considered different combinations of the heuristics for both the Single Oracle (SO) and Double Oracle (DO) formulations. In Double Oracle, there are three modules where heuristic approaches can be used: 1) selecting the initial set of targets for the restricted graph; 2) selecting initial paths for solving the restricted graph; 3) in column generation, adding paths that can improve the solution for the restricted graph. The first two modules are also needed in Single Oracle. We discuss the heuristic approaches tested for these three modules. First, for selecting the initial set of targets, we test *GreedyCover1* (GC1) and *GreedyCoverR* (GCR). Second, for selecting initial paths for the restricted graph, we enumerate all the paths (denoted as All paths) for small scale problems. In addition, we test *GreedyPathR* (GPR) and *GreedyPath3* (GP3). When using GPR for selecting initial paths, we use target utility as the number of coins on the targets. *GreedyPath3* (GP3) initialize the set of paths by listing the shortest paths from the base to a target and back to base for each target. Third, to add new paths in column generation, we test GPR and random sampling of paths (denoted as sample path).

We present the runtime and solution quality data for our algorithms as we increase the size of the game, considering game sizes of 25, 50, 100 and 200 targets. Table 1 2 3 and Table 4 show the results for each of these four cases, respectively. We had a memory limitation of 16 GB, and many of the algorithms were not able to solve the larger problems within this memory limit. We include only the data for algorithms that successfully solved all of the sample games for a given size within the memory limit.

We note that the baseline algorithms are only able to solve the smallest games within the memory limit. Even for these games, the single and double oracle methods using abstraction are all dramatically faster, and many of the variations come close to finding the optimal solutions. As we scale up the game size, the single oracle methods are not able to solve the game within the memory limit. For the largest games, the double oracle methods without sampled paths are still able to solve the problems to find good solutions, and do so very quickly. The third and fourth variation consistently show the best overall performance, with a good tradeoff between solution quality and speed.

We conduct a second experiment on large, 200-target graphs with the same distance and resource constraints but a different distribution of payoffs. For this experiment we

Algorithm	#Targets	dmax	#remaining targets	EPd	CT	ST	TT
DO + GC1 + GPR + LP + GPR	25	8	13	6.759	2	11	69
DO + GCR + GPR + LP + GPR	25	8	14	5.8845	2	12	65
DO + GC1 + GP3 + LP + GPR	25	8	12	7.2095	3	22	44
DO + GCR + GP3 + LP + GPR	25	8	10	7.1865	2	15	38
DO + GC1 + GPR + LP + Sample Paths	25	8	14	7.481	2	14	165
DO + GCR + GPR + LP + Sample Paths	25	8	14	7.3955	2	14	205
DO + GC1 + GP3 + LP + Sample Paths	25	8	14	7.605	3	97	267
DO + GCR + GP3 + LP + Sample Paths	25	8	14	7.587	2	99	283
SO + GC1 + IC + All paths + LP	25	8	12	7.702	1	105	632
SO + GCR + IC + All paths + LP	25	8	14	7.702	2	135	827
SO + GCR + IC + GP3 + LP	25	8	11	2.05	4	10	33
No contraction + No column generation	25	8	25	7.702	0	1417	14140
No contraction + Column generation	25	8	25	7.702	0	1480	14661

Table 1. Performance comparison, #target=25 and dmax = 8

Algorithm	#Targets	dmax	#remaining targets	EPd	CT	ST	TT
DO + GC1 + GPR + LP + GPR	50	20	30	5.018	9	1313	1981
DO + GCR + GPR + LP + GPR	50	20	29	5.8195	7	461	790
DO + GC1 + GP3 + LP + GPR	50	20	28	7.4945	14	187	292
DO + GCR + GP3 + LP + GPR	50	20	27	7.6415	8	162	261
DO + GC1 + GPR + LP + Sample Paths	50	20	30	5.794	9	280	4154
DO + GCR + GPR + LP + Sample Paths	50	20	29	6.4185	6	167	3925
DO + GC1 + GP3 + LP + Sample Paths	50	20	20	6.8935	6	2194	4499
DO + GCR + GP3 + LP + Sample Paths	50	20	23	6.777	6	1570	4330
BA + GCR + IC + GP3 + LP	50	20	22	0.75	5	26	1113

Table 2. Performance comparison, #target=50 and dmax = 20

Algorithm	#Targets	dmax	#remaining targets	EPd	CT	ST	TT
DO + GC1 + GPR + LP + GPR	100	29	51	6.5135	51	5753	8433
DO + GCR + GPR + LP + GPR	100	29	51	6.193	38	2170	3392
DO + GC1 + GP3 + LP + GPR	100	29	48	7.0545	52	766	1084
DO + GCR + GP3 + LP + GPR	100	29	47	7.2435	37	659	1017
DO + GC1 + GPR + LP + Sample Paths	100	29	50	6.098	46	1792	25017
DO + GC1 + GP3 + LP + Sample Paths	100	29	20	5.4735	13	2200	4420
DO + GCR + GP3 + LP + Sample Paths	100	29	30	5.0745	12	2596	5864

Table 3. Performance comparison, #target=100 and dmax = 29

Algorithm	#Targets	dmax	#remaining targets	EPd	CT	ST	TT
DO + GC1 + GPR + LP + GPR	200	45	85	6.5355	345	10360	17904
DO + GCR + GPR + LP + GPR	200	45	83	6.501	287	5307	9657
DO + GC1 + GP3 + LP + GPR	200	45	72	6.551	270	2658	4156
DO + GCR + GP3 + LP + GPR	200	45	70	6.656	189	2274	3603

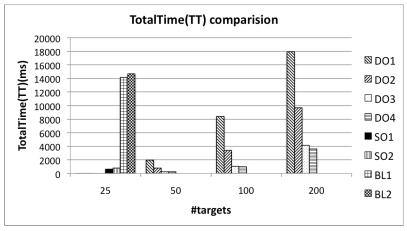
Table 4. Performance comparison, #target=200 and dmax = 45

have three payoff partitions, with the value ranges: $[0, 1], [2, 8], [9, 10]$. The ratio of target values in these ranges is 80%, 10% and 10%, respectively. Table 5 shows the results. In comparison with Table 5, the DO algorithms (especially variations 3 and 4) are even faster, though in this case variation 1 and 2 do result in higher solution qualities. The distribution of payoffs has a significant effect on the algorithm performance, and as expected, the DO variations with abstraction are most effective when there is a relatively small fraction of important targets and a large number of unimportant ones.

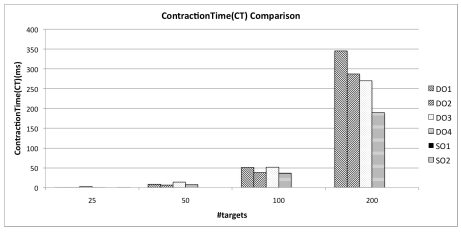
Algorithm	#Targets	dmax	#remaining targets	EPd	CT	ST	TT
DO + GC1 + GPR + LP + GPR	200	45	44	8.621	110	8177	12363
DO + GCR + GPR + LP + GPR	200	45	43	8.6085	73	3796	5275
DO + GC1 + GP3 + LP + GPR	200	45	40	7.7445	96	595	906
DO + GCR + GP3 + LP + GPR	200	45	40	7.7075	70	721	1058

Table 5. Performance comparison with 3 partition in payoff, #target=200 and dmax = 45

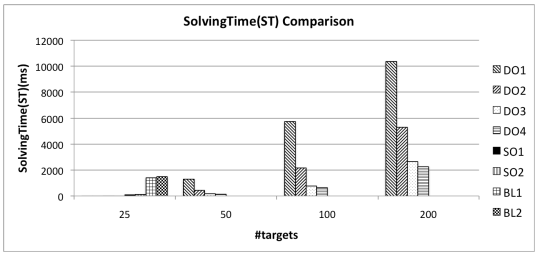
Next we present figures to visualize the runtime differences among different solution algorithms. Again, only algorithms that were able to complete within the memory bound are shown. Figures 6(a) 6(b) and 6(c) show the TotalTime, ContractionTime and SolvingTime comparison respectively among Double Oracle methods and Basic Abstraction Methods with the baseline algorithms. The figures show the same patterns of scalability discussed previously.



(a) TotalTime(TT).



(b) ContractionTime(CT).



(c) SolvingTime(ST).

Fig. 6. Runtime comparison among solvers.

Next we visualize the solution quality of our proposed algorithms in comparison with the baseline algorithms. The experiment setup is the same as the previous experiment. Figure 7 shows that we were able to compare the solution quality properly for $\#target = 25$ since the baseline algorithms were able to finish. The Basic Abstraction methods except the one which uses GP3 were able to compute the exact solution. All of the Double Oracle methods are suboptimal, but typically provide good approximations.

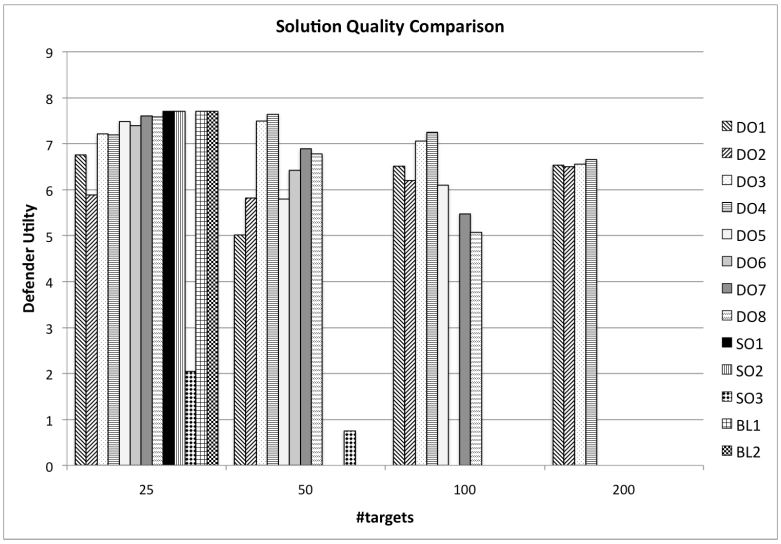


Fig. 7. Solution quality evaluation

For the final experiments we used real world data. We test our algorithms on grid-based graphs constructed from elevation and animal density information from a conservation area in Southeast Asia. The area is discretized into a grid map with each grid cell of size 50m by 50m. The problem has a large number of targets and feasible patrol routes when considering a practical distance limit constraint (often 5km-20km). We tested with four different game sizes, and the result shows that the proposed algorithm can solve real-world scale GSGs efficiently (see Table 6). Only DO4 was used for this experiment since it provides superior performance than others. The payoff range for the targets were $[0, 90]$.

7 Conclusion

Green security games are being used to help combat environmental crime by improving patrolling strategies. However, the applications of GSG are still limited due to the computational barriers of solving large, complex games based on underlying graphical structures. Existing applications require manual pre-processing to come up with suitably abstract games that can be solved by existing solvers. We address this problem

#targets	dmax	#remaining targets	EPd	CT	ST	TT
100	5000	56	25.83	121	303	789
200	8000	88	25.79	67	926	1678
500	15000	92	18.56	1928	1107	4403
1000	18000	100	16.29	12302	2072	18374

Table 6. Results of using abstraction in real world data

by designing the first algorithm for solving graph-based security games that integrates automated abstraction techniques with strategy generation methods. Our algorithm is the first to be able to provide high-quality solutions to very large green security games (thousands of nodes) in seconds, potentially opening up many new applications of GSG while avoiding the need for some of the arbitrary, manual abstraction stages when generating game models. With additional work to develop fast exact slave algorithms, we should also be able to provide exact solutions using this approach to large GSG. We also plan to investigate approximate slave formulations with performance bounds, using abstraction to compute solution concepts from behavioral game theory such as quantal response equilibrium, and applying our algorithms to real-world applications in green security games.

Acknowledgement

We would like to thank to our partners from Rimba and Panthera for providing the real world data set. This work was supported by the NSF under Grant No. IIS-1253950.

References

1. Govt of Mozambique announces major decline in national elephant population, May 2015.
2. The IUCN Red List of threatened species, April 2015.
3. Estimate of global financial losses due to illegal fishing, February 2016.
4. N. Basilico, N. Gatti, and F. Amigoni. Patrolling security games: Definition and algorithms for solving large instances with single patroller and single intruder. *Artificial Intelligence*, 184:78–123, 2012.
5. G. V. Batz, R. Geisberger, S. Neubauer, and P. Sanders. Time-dependent contraction hierarchies and approximation. In *Experimental Algorithms*, pages 166–177. Springer, 2010.
6. M. Breton, A. Alj, and A. Haurie. Sequential Stackelberg equilibria in two-person games. *Journal of Optimization Theory and Applications*, 59(1):71–97, 1988.
7. M. Brown, W. B. Haskell, and M. Tambe. Addressing scalability and robustness in security games with multiple boundedly rational adversaries. In *Decision and Game Theory for Security*, pages 23–42. Springer, 2014.
8. N. Brown, S. Ganzfried, and T. Sandholm. Hierarchical abstraction, distributed equilibrium computation, and post-processing, with application to a champion no-limit Texas hold'em agent. Technical report, 2014.
9. T. H. Cormen, C. E. Leiserson, and R. L. Rivest. The floyd-warshall algorithm. *Introduction to Algorithms*, pages 558–565, 1990.

10. F. Fang, T. H. Nguyen, R. Pickles, W. Y. Lam, G. R. Clements, B. An, A. Singh, M. Tambe, and A. Lemieux. Deploying PAWS: Field Optimization of the Protection Assistant for Wildlife Security. In *Proceedings of the Innovative Applications of Artificial Intelligence*, 2016.
11. F. Fang, P. Stone, and M. Tambe. When security games go green: Designing defender strategies to prevent poaching and illegal fishing. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2015.
12. C. Field and R. Laws. The distribution of the larger herbivores in the Queen Elizabeth National Park, Uganda. *Journal of Applied Ecology*, pages 273–294, 1970.
13. S. Ganzfried and T. Sandholm. Potential-aware imperfect-recall abstraction with earth mover's distance in imperfect-information games. In *Conference on Artificial Intelligence (AAAI)*, 2014.
14. R. Geisberger, P. Sanders, D. Schultes, and D. Delling. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In *Experimental Algorithms*, pages 319–333. Springer, 2008.
15. R. Geisberger, P. Sanders, D. Schultes, and C. Vetter. Exact routing in large road networks using contraction hierarchies. *Transportation Science*, 46(3):388–404, 2012.
16. A. Gilpin and T. Sandholm. A competitive Texas Hold'em poker player via automated abstraction and real-time equilibrium computation. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, volume 21, page 1007, 2006.
17. A. Gilpin and T. Sandholm. Better automated abstraction techniques for imperfect information games, with application to texas hold'em poker. In *AAMAS*, page 192, 2007.
18. A. Gilpin and T. Sandholm. Lossless abstraction of imperfect information games. *Journal of the ACM (JACM)*, 54(5):25, 2007.
19. A. Gilpin, T. Sandholm, and T. B. Sørensen. Potential-aware automated abstraction of sequential games, and holistic equilibrium analysis of texas hold'em poker. In *Proceedings of the Conference on Artificial Intelligence (AAAI)*, volume 22, page 50, 2007.
20. A. Gilpin, T. Sandholm, and T. B. Sørensen. A heads-up no-limit Texas Hold'em poker player: discretized betting models and automatically generated equilibrium-finding programs. In *AAMAS*, pages 911–918, 2008.
21. W. B. Haskell, D. Kar, F. Fang, M. Tambe, S. Cheung, and E. Denicola. Robust Protection of Fisheries with COmPASS. In *AAAI*, pages 2978–2983, 2014.
22. T. Holmern, J. Muya, and E. Røskaft. Local law enforcement and illegal bushmeat hunting outside the Serengeti National Park, Tanzania. *Environmental Conservation*, 34(01):55–63, 2007.
23. H. Iwashita, K. Ohori, H. Anai, and A. Iwasaki. Simplifying Urban Network Security Games with Cut-based Graph Contraction. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 205–213, 2016.
24. M. Jain, E. Kardes, C. Kiekintveld, F. Ordónez, and M. Tambe. Security Games with Arbitrary schedules: A Branch and Price Approach. In *AAAI*, 2010.
25. C. Kiekintveld, M. Jain, J. Tsai, J. Pita, F. Ordóñez, and M. Tambe. Computing optimal randomized resource allocations for massive security games. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 689–696, 2009.
26. C. Kroer and T. Sandholm. Extensive-form game abstraction with bounds. In *Proceedings of the fifteenth ACM conference on Economics and computation*, pages 621–638, 2014.
27. G. Leitmann. On generalized stackelberg strategies. *Journal of Optimization Theory and Applications*, 26(4):637–643, 1978.
28. T. H. Nguyen, F. M. Delle Fave, D. Kar, A. S. Lakshminarayanan, A. Yadav, M. Tambe, N. Agmon, A. J. Plumptre, M. Driciru, F. Wanyama, et al. Making the most of our regrets:

Regret-based solutions to handle payoff uncertainty and elicitation in green security games. In *Decision and Game Theory for Security*, pages 170–191. Springer, 2015.
29. P. Paruchuri, J. P. Pearce, J. Marecki, M. Tambe, F. Ordonez, and S. Kraus. Playing games for security: An efficient exact algorithm for solving Bayesian Stackelberg games. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2*, pages 895–902, 2008.
30. J. Pita, H. Bellamane, M. Jain, C. Kiekintveld, J. Tsai, F. Ordóñez, and M. Tambe. Security applications: Lessons of real-world deployment. *ACM SIGecom Exchanges*, 8(2):5, 2009.
31. T. Sandholm and S. Singh. Lossy stochastic game abstraction with bounds. In *Proceedings of the 13th ACM Conference on Electronic Commerce*, pages 880–897, 2012.
32. E. Shieh, B. An, R. Yang, M. Tambe, C. Baldwin, J. DiRenzo, B. Maule, and G. Meyer. Protect: A deployed game theoretic system to protect the ports of the united states. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 13–20, 2012.
33. E. Shieh, M. Jain, A. X. Jiang, and M. Tambe. Efficiently solving joint activity based security games. In *AAAI*, pages 346–352. AAAI Press, 2013.
34. S. Skiena. Dijkstra's algorithm. *Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica, Reading, MA: Addison-Wesley*, pages 225–227, 1990.
35. J. Tsai, C. Kiekintveld, F. Ordonez, M. Tambe, and S. Rathi. Iris-a tool for strategic security allocation in transportation networks. 2009.
36. B. Von Stengel and S. Zamir. Leadership with commitment to mixed strategies. 2004.
37. R. Yang, B. Ford, M. Tambe, and A. Lemieux. Adaptive resource allocation for wildlife protection against illegal poachers. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, pages 453–460, 2014.
38. R. Yang, A. X. Jiang, M. Tambe, and F. Ordonez. Scaling-up Security Games with Boundedly Rational Adversaries: A Cutting-plane Approach. In *IJCAI*, 2013.
39. Z. Yin, A. X. Jiang, M. Tambe, C. Kiekintveld, K. Leyton-Brown, T. Sandholm, and J. P. Sullivan. TRUSTS: Scheduling randomized patrols for fare inspection in transit systems using game theory. *AI Magazine*, 33(4):59, 2012.
40. M. Zinkevich, M. Johanson, M. Bowling, and C. Piccione. Regret minimization in games with incomplete information. In *Advances in neural information processing systems*, pages 1729–1736, 2007.