

Abstraction Methods for Solving Graph-Based Security Games

Anjon Basak^{1(✉)}, Fei Fang², Thanh Hong Nguyen²,
and Christopher Kiekintveld¹

¹ University of Texas at El Paso,
500 W University Ave, El Paso, TX 79902, USA
abasak@miners.utep.edu, cdkiekintveld@utep.edu

² University of Southern California,
941 Bloom Walk, SAL 300, Los Angeles, CA 90089, USA
{feifang, thanhng}@usc.edu

Abstract. Many real-world security problems can be modeled using Stackelberg security games (SSG), which model the interactions between 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. We demonstrate empirically that these abstraction methods can result in dramatic improvements in solution time with modest impact on solution quality.

Keywords: Security · Green security · Abstraction · Contraction · Game theory

1 Introduction

As a society, we face a wide variety of security challenges in protecting people, infrastructure, computer systems, and natural resources from criminal activity. A common challenge across all of these different security domains is making the best use of limited resources to improve security, even in the face of intelligent, highly motivated attackers. Green security domains focus particularly on the problem of 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 physical areas that need to be protected. 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.

Recent work on *green security games* [8, 11] has proposed formulating the problem of finding optimal patrols to prevent environmental crime as a Stackelberg security game [23]. 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 the patrolling strategy. For example, a poacher will try to target areas of high animal density where patrols are the least likely to occur. The goal in solving the green security game is to find the patrolling strategy for the defender that maximizes the environmental protection.

Green security games typically model the movement constraints for the defender patrols using a graph to capture the physical terrain. Unfortunately, this leads to a major computational challenge because the number of possible paths grows exponentially with the size of the graph, and enumerating all possible combinations of paths that could be followed by the available resources makes the problem even more intractable [27, 33]. Several algorithms have been proposed in the literature to solve these games more efficiently [22, 26]. 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. While these methods have improved scalability dramatically, it is still not possible to solve real-world problems covering large areas with a fine-grained resolution on the terrain.

In this paper we introduce a new direction for solving large security games based on graphs by using abstraction to reduce the size of the problem before applying an equilibrium solver. This direction is motivated by the success of automated abstraction methods in solving other very large games, most notably computer poker [16, 17, 19, 20, 38]. In addition, there has been work on using abstraction methods for graphs in the literature on pathfinding algorithms (e.g., [4, 14, 15, 32]). We introduce a new solution technique for green security games that first applies a graph abstraction method to simplify the graph used to define the strategy space, and then apply an equilibrium solver to the reduced game. Once the simplified game has been solved, we map the solution back into the original game. We note that this overall approach can be used in combination with previous work on incremental strategy generation to further improve scalability, since the faster incremental algorithms can be used as equilibrium solvers in our method.

We empirically evaluate our abstraction techniques on graph-based security games motivated by the problems encountered in green security domains. Our experiments demonstrate that abstraction has great potential to scale up algorithms for green security games to much larger problem sizes. In particular,

we show that applying our graph abstraction methods dramatically improves computation time, while introducing relatively modest reductions in the solution quality. Future work to improve the abstraction methods should yield even greater advantages.

2 Related Work

The first approach to compute strategic resource allocations for security was to find a randomized strategy after enumerating all possible resource allocations [27], which is used by the Los Angeles Airport Police as an application called ARMOR [28]. After that a more compact form security game representation was used [23] to achieve a faster algorithm (IRIS [33]), which is used by Federal Marshal Service (FAMS). Another algorithm, ASPEN [22], was introduced to prevent the exponential explosion of the variables since all the possible resource allocation were used to get the strategy for the defender. ASPEN uses branch-and-price approach to alleviate problems of previous algorithms. In this algorithm, joint scheduling was used to get a probability distribution over the pure strategy space. Most recently, to tackle more massive games an approach based on cutting planes was introduced [36] to make the solution space more manageable. Game theoretic algorithms are also used to secure ports [30] and trains [37].

Recently, successful deployment of game theoretic applications motivated researchers to use game theory in green security domains [8, 21, 35]. The transition to green security domain provided new challenges and research questions. For example rather than using the SSG game model a new game model called GSG [11] has been introduced. Assumptions about attacker being able to fully observe the defenders strategy was also not suitable with the real world scenario, where the poacher doesn't have the capability to observe the complete patrolling strategy. So, partial observability and bounded rationality have been introduced to make the attacker model more realistic. The defenders also have the issue of limited observability which results in payoff uncertainty. To address this issues an algorithm called ARROW [26] was introduced.

Many abstraction techniques have also been developed for extensive form games with uncertainty including both lossy [29] and lossless [18] abstraction. There also has been some work which gives bounds on the error introduced by abstraction [24]. There is also imperfect recall abstraction which considers hierarchical abstraction [9] and earth mover's distance [13].

Contraction technique [14] has been used to achieve fast routing in road networks, where contraction acts as a pre-processing step. One issue with the contraction algorithm was finding the shortest path. To alleviate the issue a bidirectional Dijkstra [15, 31] algorithm is used which is fast. A time dependent contraction algorithm also has been introduced for time dependent road network [4].

Another area where game abstraction is used extensively is in making computer poker agents. A simple version of full scale Texas hold'em poker, 2 player

limit Texas hold'em poker, has $O(10^{18})$ nodes [5]. Without any kind of abstraction solving poker is extremely hard. Even though recently heads-up limit hold'em poker is solved [6] without any abstraction, it required gigantic amount of data processing.

3 Domain Motivation

Wildlife plays a major role in our ecosystem by thriving and balancing every aspect of the environment. Illegal activities such as poaching pose a major threat to biodiversity in wildlife and marine areas, and many endangered species such as rhinos and tigers are under extreme threat of extinction due to human activity. 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 in last five years. Recently, elephants have been added to the IUCN Red List [2].

Marine species are also facing danger because of overfishing and illegal fishing. This is not only hampering the biodiversity of our ecosystem but also causing harm to the people of coastal areas where people depend on fishing for both

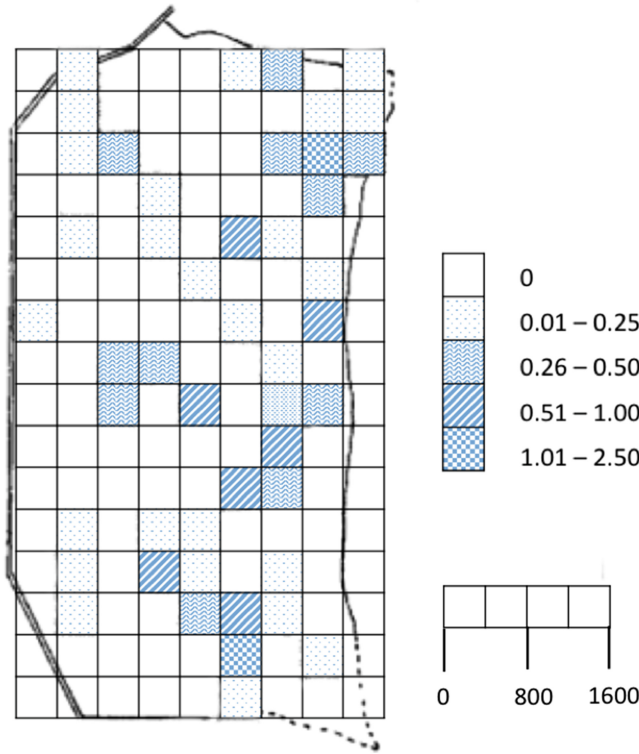


Fig. 1. Mean numbers of elephants/ 0.16 km^2 in Queen Elizabeth National Park, Uganda

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]. Different organizations like WCS are providing strategies which involve patrolling of forest areas to prevent poaching and protect wildlife habitats.

PAWS [10] is a new application based on solving green security games which helps to design patrolling strategies to protect wildlife in threatened areas. It is based on a kind of virtual street map to define the area the needs to be patrolled. In the application, the area of interest is divided into grid cells that capture information about the terrain, as well as other important information such as the animal density. For example, Fig. 1 shows the mean number of elephants in each area of the grid representing the Queen Elizabeth National Park in Uganda [12].

Each cell is 400 *m* by 400 *m*. The first thing we notice is there are many cells which have no animal count at all, and if there is minimal activity it is very inefficient to consider these areas as targets to patrol. Even if we give less priority to the cells with low animal activity, it is still very difficult for the patroller to patrol even a single target without any path strategy inside those cells, because each cell is 0.16 *km*². PAWS solve this problem by having a finer grained structure. For example each grid cell in Fig. 1 can be divided into 50*m* by 50*m* cells. However, this results in large number of targets, so the existing algorithms are not able to compute a strategy directly on the full problem with the large number of small cells. Our goal in considering abstraction methods is to make it computationally feasible to directly analyze games with this fine-grained structure.

4 Game Model

A typical green security game (GSG) model is specified by dividing a protected wildlife area into grid based cells, as shown in the example in Fig. 1. 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 representation as shown in Fig. 2. Each node represents a target t_i . There are utilities on both the nodes and edges in our graphs. $U_d^c(i, j)$ represents the utility on the edge between target i and j .

To patrol a cell t_i the patroller needs to cover a certain distance s_{t_i} . We use $d(i, j)$ to represent the distance from target i to j , as shown in Fig. 2, where $s_{t_1} = 100$ *m* and $d(2, 3) = 1$ *km*. A patrolling path is a sequence of consecutive targets covered by the defender. Inside a target (i.e., grid region) the defender covers some distance then he moves to the next target. The defender has a constraint on the maximum length of a patrol, so each path must be shorter than a given maximum value d_{max} . Typically the patrol starts in a base station and ends in the same base station. For example, a patrolling path is shown in Fig. 3 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 .

In a GSG the defender's pure strategies are the set of joint patrolling paths $J_m \in J$. The defender has a limited number of resources R , each of which can be

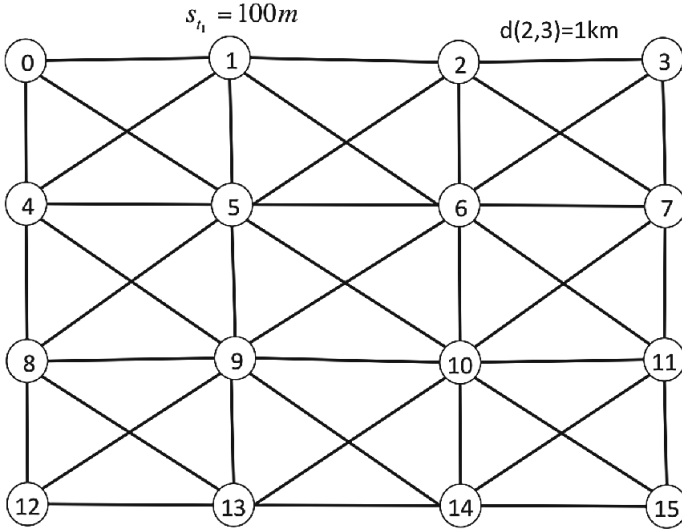


Fig. 2. A graph representation of a grid based GSG

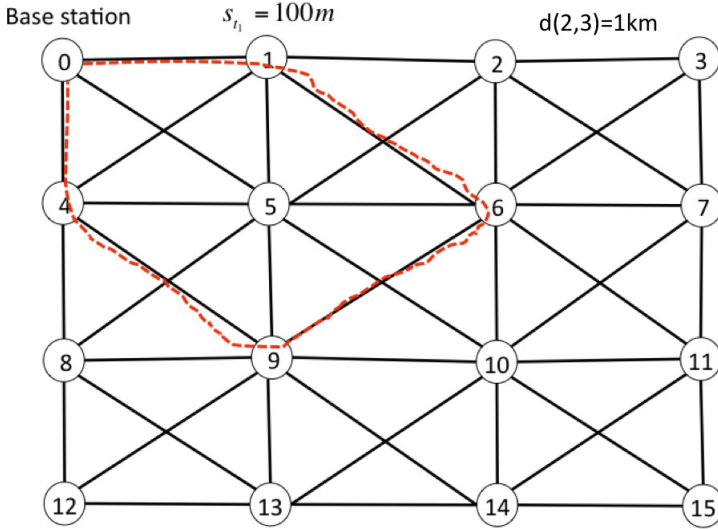


Fig. 3. A patrolling path

assigned to at most one patrolling path. Each joint patrolling path J_m assigns each resource to a specific path. Each path $p_k \in J_m$ in a joint patrolling path covers a set of targets $t \in T$. Every path p_k is unique and is generated using breadth first search where we have a base target t_b . The length of each patrolling path must be less than the distance constraint 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 $c_t = \sum_{J_m} P_{J_m t} x_m$. Table 1 shows an example of matrix P . This matrix P grows exponentially with the number of targets and resources, which makes performing computations on this representation intractable as the size of the game gets very large.

Table 1. Matrix P for joint patrolling paths

	J_1	J_2	...	J_n
t_1	1	1	...	1
t_2	1	1	...	0
t_3	1	0	...	1
...
...
...
t_n	0	1	...	1

If target t is protected then the defender receives reward $U_d^c(t)$ when attacker attacks target t , otherwise a penalty $U_d^u(t)$ is given. In the domain, these values are based on the density of the animals in the area attacked, as a proxy for the expected losses due to poaching activities. We assume that the number of losses decreases when the defender is patrolling the area. The attacker receives reward $U_a^u(t)$ if the attack is on an uncovered area, where the defender is not patrolling, or penalty $U_a^c(t)$ if the attack is executed in a patrolled area. We also assume $U_d^c(t) > U_d^u(t)$ and $U_a^u(t) > U_a^c(t)$. The expected payoffs for defender and attacker are given by:

$$U_d(c, t) = \sum_{t \in T} a_t (c_t U_d^c(t) + (1 - c_t)(U_d^u(t))) \quad (1)$$

$$U_a(c, t) = \sum_{t \in T} a_t (c_t U_a^c(t) + (1 - c_t)(U_a^u(t))) \quad (2)$$

We use the Stackelberg model for the GSG, as in previous work on 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 $T = t_1, t_2, \dots, t_n$ from the attackers by allocating R resources. The attacker attacks one of the T targets. a_i is the binary attack vector for attacking target t_i . The solution concept used to find equilibrium in this game model is called Strong Stackelberg Equilibrium (SSE) [7, 25, 34]. In this equilibrium 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 his strategy and will choose a best response and break ties in favor of the defender.

5 Solution Approach

We first present a basic method for solving our games using a mixed-integer programming formulation identical to baselines presented in previous work [22]. Solving this mixed integer linear program (MILP) will determine the optimal patrolling strategy for the given game as a probability distribution over the joint patrolling paths. This formulation is somewhat naïve, and it does not scale well to large games due to the exponential number of possible joint paths as the graph grows larger (result in a very large P matrix). We could improve this formulation with any of the more advanced methods based on incremental strategy generation to solve larger problem instances. However, our focus here is on investigating the benefits of abstraction, rather than improving the base algorithm for equilibrium computation, so this basic formulation is sufficient for our purposes. The MIP is represented by the following equations:

$$\max \mathbf{d} \quad (3)$$

$$\mathbf{d} - DPx - U_d^u \leq (1 - a)M \quad (4)$$

$$k - APx - U_a^u \leq (1 - a)M \quad (5)$$

$$APx + U_a^u \leq k \quad (6)$$

$$\sum_{x_i} = 1 \quad (7)$$

$$\sum_{a_i} = 1 \quad (8)$$

$$x, a \geq 0 \quad (9)$$

Equation 3 represents the objective function, which maximizes the expected payoff for the defender. Equations 4–6 enforce the players to choose mutual best responses. Equation 7 makes sure that the probability distribution over the joint patrolling paths sums to one. Equation 8 enforces the attacker to attack only one target.

This MIP solves for the probability distribution x over the joint patrolling paths J . This is the strategy defender commits to. In an SSG, attacker observes the defender strategy then attacks a target. We find this attacked target by taking the target with the maximum expected payoff for the attacker using Eq. 2. The attacker breaks tie in favor of defender, as specified in the Strong Stackelberg Equilibrium solution concept. That means if there are multiple targets with same expected payoff for the attacker, the attacker chooses the target with the maximum expected payoff for the defender. Then we can easily calculate the defender expected payoff using Eq. 1.

6 Abstraction for Graph-Based Security Games

Abstraction techniques play a vital role in solving large game models in game theory. Typically, abstraction techniques exploit action space similarity to shrink

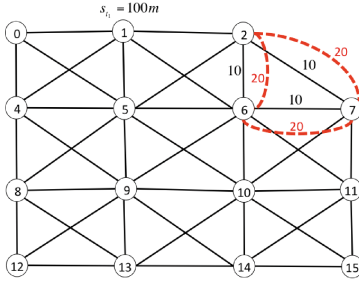
the game model before applying a solver (such as an equilibrium finding algorithm) to the simpler game. We have already seen successful computer poker agents which uses abstraction techniques where an original game tree can have approximately 10^{71} nodes.

Our approach here applies this idea of game abstraction to green security games. The main idea we exploit in our abstraction is that there are often relatively few important targets in a GSG. The key regions of high animal density are relatively few, and many areas have low density, as shown in Fig. 1. Based on this observation, we believe that many targets in the game can be effectively removed from the analysis to reduce the complexity of large GSG while retaining the important features of the game.

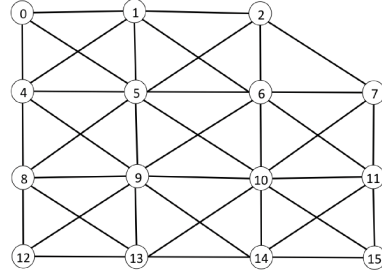
The main idea of our abstraction methods is based on the idea of graph contraction, a version of which has been developed for pathfinding in road networks [14, 15]. The idea in contraction is to systematically remove unnecessary nodes from a graph by introducing edges and maintaining the shortest path structure from the complete graph. However, we must also consider the utilities of the nodes when performing contraction in a GSG, since we are interested not only in the shortest path but also the value of the targets on the path. We contract nodes depending on their utility to the defender, relying on the observation that there may be many targets with very low utilities.

The first step in our contraction process is selecting the targets we want to contract. The selection process works according to Pareto optimality. This Pareto optimality will give us a notion about which targets give the defender less utility even after traveling a long distance compared with other targets. Each target in the graph has reward $U_d^c(t), t \in T$ if the total distance s_t of the target is covered. A target t_i is not Pareto optimal if $\exists j \in N, i \neq j, U_d^c(t_i) < U_d^c(t_j)$ and $s_i \geq s_j$. Since we are contracting the graph to get a patrolling strategy for the defender, we only use the utility for the defender to find targets which are not Pareto optimal. Pareto optimality is one of the possible ways of selecting targets to be contracted. Contracting a dominated target may lead to a loss in the solution quality. We defer a further investigation of the the selection criteria to future work.

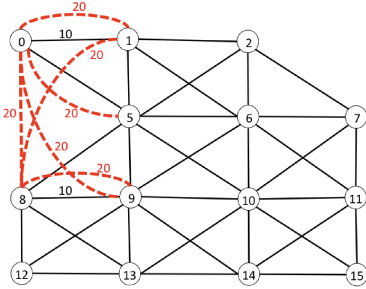
Contraction of a node t is done by replacing edge (v, t, v') with a shortcut (v, v') , where $v \neq v'$ and v, v' are neighbors of t . Next we evaluate the edges to determine whether to keep them or not. We do not need direct edge (v, v') , if $d(v, t) + d(t, v') \geq d(p)$, where p is another shortcut. To find the shortest path we use a brute force approach. We check if there exists another path (v, t', v') where $d(v, t') + d(t', v') \leq d(v, t) + d(t, v')$. We keep expanding node t' if $d(v, t') < d(v, t) + d(t, v')$. If $t' = v'$ and $d(v, t') < d(v, t) + d(t, v')$ then we do not need edge (v, v') . If the shortcut (v, v') is necessary then we assign the node's utility on that new edge. Since shortest path computation can become expensive we can limit our search for shortest path where can consider only the neighbors of the contracted node. This search space limit does not affect the correctness of the graph since we are introducing the shortcut (v, v') if we do not find another shortcut path.



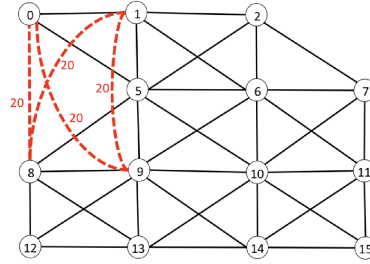
(a) Contracting node 3



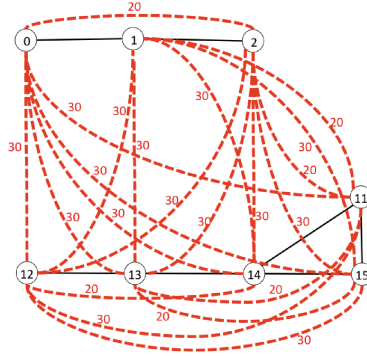
(b) Removing unnecessary edges after contracting node 3



(c) Contracting node 4



(d) Removing unnecessary edges after contracting node 4



(e) Final graph after contraction of node 3, 4, 5, 6, 7, 8, 9, 10

Fig. 4. Contraction procedure for different nodes

In this problem, adding shortcut for neighbors of node t is sufficient to make sure the shortest paths in the original graph are preserved. If there exist two nodes v_1, v_2 between which the shortest path must include node t , then in this shortest path, the node prior to t (denoted as v_3) and the node right after t (denoted as v_4) are neighbors of t and a shortcut (v_3, v_4) will be added for sure according to our approach. Otherwise the optimality of the shortest path

between v_1 and v_2 is violated. Therefore, after contraction, the shortest path between v_1 and v_2 can be replaced by one containing the shortcut (v_3, v_4) and the distance is unchanged.

Figure 4 shows how the contraction procedure works on the example graph shown in Fig. 2. The number on the edge represents the distance of the two nodes which are connected by that edge. Figure 4a shows the new edges introduced as red dotted curve after removing target 3. After that we compute the shortest path for each of the new edge. Figure 4b shows that there were already existing shortest paths. So all of the new edges were pruned because they are unnecessary. Similarly, Fig. 4c shows the new edges introduced as red dotted lines after contracting target 4. Then Fig. 4d shows what happened to the graph as we compute shortest path calculation. As we notice that some new edges remained (e.g. $t_0 \rightarrow t_8$) because those nodes are in the shortest path. Figure 4e shows the final graph after contracting nodes $(3, 4, 5, 6, 7, 8, 9, 10)$.

Algorithm 1. Contraction Procedure

```

1: procedure CONTRACTGRAPH ▷
2:    $G \leftarrow \text{Graph}()$  ▷ \text{Initiate the graph to contract}
3:    $n_d \leftarrow \text{ContractedNodes}()$  ▷ \text{Get the nodes to contract}
4:   for  $t \leftarrow n_d.\text{pop}()$  do ▷ \text{for every node } t \text{ in } n_d
5:     for  $v \leftarrow n_c.\text{neighbors}()$  do
6:       for  $v' \leftarrow n_c.\text{neighbors}()$  do
7:         if  $v \neq v'$  then
8:            $d_{mid} \leftarrow \text{Dist}(v, t, v')$ 
9:            $d_s \leftarrow \text{ShortestDist}(v, v')$ 
10:          if  $d_{mid} \leq d_s$  then ▷  $t$  in shortest path
11:             $\text{AdjustNeighbors}(v, t, v')$ 
12:          end if
13:        end if
14:      end for
15:    end for
16:  end for
17: end procedure

```

Algorithm 1 shows how the contraction procedure works. Lines 2–3 initialize the variables. Lines 4 – 16 check whether a contracted node t falls into the shortest path between node v and v' . In line 8 $\text{Dist}(v, t, v')$ returns the distance of edge $v \rightarrow t \rightarrow v'$. In line 9 $\text{ShortestDist}(v, v')$ finds the shortest path between v and v' . If t is in the shortest path (line 10), we add a direct edge between v and v' using method $\text{AdjustNeighbors}()$ shown in Algorithm 2. Algorithm 2 also makes sure that there is a path between v and v' which consists of node t . Lines 2–4 in Algorithm 2 builds an edge that consists of node t . $\text{Path}(v, t)$ returns the intermediate nodes to reach t from v including v . We do this because there might be previously contracted nodes in the edge from v or v' to t . Suppose $\text{Path}(v, t_5)$ returned $p_v^{t_5} = (v \rightarrow t_1 \rightarrow t_2)$. $\text{Path}(v', t_5)$ returned $p_{v'}^{t_5} = v' \rightarrow t_6 \rightarrow t_7$. After

Algorithm 2. Adjust Neighbor Procedure

```

1: procedure ADJUSTNEIGHBORS( $v, t, v'$ ) ▷
2:    $p_v^t \leftarrow \text{Path}(v, t)$ 
3:    $p_{v'}^t \leftarrow \text{Path}(v', t)$ 
4:    $p \leftarrow \text{Merge}(p_v^t, t, p_{v'}^t)$ 
5:    $v.\text{AddNeighbor}(v', p)$ 
6:    $v'.\text{AddNeighbor}(v, p)$ 
7:    $v.\text{RemoveNeighbor}(t)$ 
8:    $v'.\text{RemoveNeighbor}(t)$ 
9: end procedure

```

that $\text{Merge}(p_v^{t_5}, t_5, p_{v'}^{t_5})$ will merge these two paths and return $v \rightarrow t_1 \rightarrow t_2 \rightarrow t_5 \rightarrow t_7 \rightarrow t_6 \rightarrow v'$. Next in line 5 – 6 v and v' adds each other as neighbor inside method $\text{AddNeighbor}(v, p)$.

Once we have the contracted graph we can follow the same procedure described in Sect. 5 to solve the game based on the smaller. The only difference is that we have fewer nodes and the edges can represent paths that cover multiple underlying nodes. Solving for the optimal solution of this contracted game will not necessarily result in finding the optimal solution to the original game, but we will show in our experimental results that it is a good approximation.

6.1 Reverse Mapping

The MIP returns a probability distribution over the joint patrolling paths J . But the P' matrix used by the MIP was constructed from the contracted graph, where patrolling paths do not reflect the original graph: each edge in the contracted graph can represent multiple nodes and edges in the original graph. We need to find the P matrix which will consider those contracted nodes. Given the defender's strategy, found from the MIP, using a P matrix which considers all the targets including the contracted node, we can find the target with the maximum expected payoff for the attacker. What we present now is a reverse mapping function which transform the P' matrix into P . For each of the joint patrolling paths this function $\text{ReverseMap}(P') \rightarrow P$ checks if two consecutive nodes have edges with contracted nodes. If so, the node is marked as covered in the P matrix. There is an alternative of first mapping the routes in the contracted graph to routes in the original graph, and then solve a MIP. Either way is correct as the reversemapping does not depend on solving the MIP.

The reverse mapping function $\text{ReverseMap}(P') \rightarrow P$ gives us joint patrolling paths for the original graph. Algorithm 3 shows how the ReverseMap method works. For every joint patrolling path J_m (line 4) and for every path p_m in J_m (line 5) we mark two consecutive nodes as covered (line 9–10). Then we check if two consecutive nodes have any edge consists of contracted nodes. If so, we mark those targets covered (line 11–14). Once we have the joint patrolling paths P for the original graph, we can compute the attack vector and also defender's expected payoff.

Algorithm 3. Reverse Mapping Procedure

```

1: procedure REVERSEMAP
2:    $J \leftarrow \text{JointPaths}()$ 
3:    $index \leftarrow 0$ 
4:   for  $J_m \leftarrow J.pop()$  do
5:     for  $p_k \leftarrow J_m.pop$  do
6:       for  $i \leftarrow p_k.len - 1$  do
7:          $t \leftarrow p_k(i)$ 
8:          $t_{next} \leftarrow p_k(i + 1)$ 
9:          $P[t][index] \leftarrow 1$ 
10:         $P[t_{next}][index] \leftarrow 1$ 
11:         $e_{nodes} \leftarrow \text{EdgeNodes}(t, t_{next})$ 
12:        for  $t_e \leftarrow e_{nodes}.pop()$  do
13:           $P[t_e][index] \leftarrow 1$ 
14:        end for
15:      end for
16:    end for
17:     $index \leftarrow index + 1$ 
18:  end for
19: end procedure

```

7 Experiments

We begin by showing what the solution of the example game shown in Fig. 4e looks like. We computed this solution based on Fig. 2 after contracting targets (3, 4, 5, 6, 7, 8, 9, 10). We show the reduced P' matrix for the contracted graph is formed, and then we show how we find the P matrix after reverse mapping. For simplicity we use only two joint patrolling paths which have a positive probability distribution. We used the maximum distance 6 to generate 9 patrolling paths without any duplicates for the contracted graph. The joint patrolling paths are $J_0 = (p_6, p_8)$ and $J_1 = (p_6, p_7)$ where $p_6 = (t_0 \rightarrow t_{15} \rightarrow t_0)$, $p_7 = (t_0 \rightarrow t_2 \rightarrow t_1 \rightarrow t_0)$, $p_8 = (t_0 \rightarrow t_{11} \rightarrow t_1 \rightarrow t_0)$. The MIP returns a probability distribution x where $x_0 = 0.076$ and $x_1 = 0.250$. There are other joint patrolling paths for which x_i has a positive probability, but we are only showing part of the solution. A part of P' matrix is given in Table 2 for the contracted graph shown in Fig. 4e.

Next we use the reverse mapping function $\text{ReverseMap}(P') \rightarrow P$ to obtain the P matrix (Table 3) for the graph shown in Fig. 2. The $\text{ReverseMap}(P') \rightarrow P$ checks whether a target t_i is in an edge between two consecutive target in a patrolling path which belongs to a joint patrolling path. If so, t_i is marked as covered. For example, t_5, t_{10} are marked as covered because they are part of an edge from t_0 to t_{15} for joint schedule J_0 and J_1 .

Now we present the results of our initial experimental evaluation of the the effectiveness of abstraction based on graph contraction on security games. The baseline solution provides us with a defender strategy on joint patrolling paths for any given graph (contracted or not). We did our analysis by comparing the baseline case with different levels of contraction. The comparisons are done

Table 2. A part of the P' matrix for example game in Fig. 4e,

	$x_0 = 0.0769$	$x_1 = 0.2507$...
	J_0	J_1	...
t_0	1	1	...
t_1	1	1	...
t_2	0	1	...
t_{11}	1	0	...
t_{12}	0	0	...
t_{13}	0	0	...
t_{14}	0	0	..
t_{15}	1	1	...

Table 3. A part of P matrix for example game in Fig. 2

	$x_0 = 0.0769$	$x_1 = 0.2507$...
	J_0	J_1	...
t_0	1	1	...
t_1	1	1	...
t_2	0	1	...
t_3	0	0	...
t_4	0	0	...
t_5	1	1	...
t_6	1	0	...
t_7	0	0	...
t_8	0	0	...
t_9	0	0	...
t_{10}	1	1	...
t_{11}	1	0	...
t_{12}	0	0	...
t_{13}	0	0	...
t_{14}	0	0	..
t_{15}	1	1	...

for two metrics: solution quality and runtime. For each level of contraction we measured the level of error introduced and how much time we could save by introducing that error. We denote the error by $\text{epsilon}(\epsilon)$, which can be defined as follows, where $U'_d(c, a)$ is the expected payoff for defender when using contraction where $U_d(c, a) \geq U'_d(c, a)$:

$$\epsilon = \frac{[U_d(c, a) - U'_d(c, a)]}{U_d(c, a) * 100} \quad (10)$$

For our experimental setup we used 100 randomly generated, 2-player security games intended to capture the important features of typical green security games. Each game has 25 targets (nodes in the graph). Payoffs for the targets are chosen randomly from the range -10 to 10 . The rewards for defender or attacker

are positive and the penalties are negative. We set the distance constraint to 6. In the baseline solution we have no contraction. For different level of abstraction the number of contracted nodes ($\#CN$) varies between the values: (0, 2, 5, 8, 10). Figure 5 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 MIP, and RMT considers time to generate the P matrix for the original graph from the solution to the abstracted game.

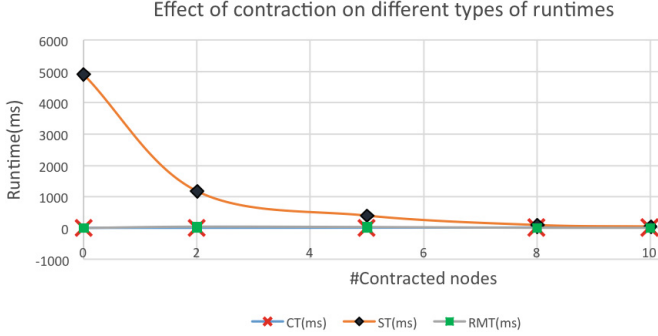


Fig. 5. Effect of contraction on contraction time(CT), solving time(ST) and reverse mapping time(RMT)

We first note that as the graph becomes more contracted ST takes much less time, as shown in Fig. 5. The next experimental result presented in Fig. 6 shows how much error is introduced as we increase the amount of contraction and the amount of time we can save by using contraction. The amount of time saved is very significant compared with the amount of error introduced.

In the next experiment we tried to see how the density of high-valued targets in the graph affects the expected utility of defender if we use contraction. If a

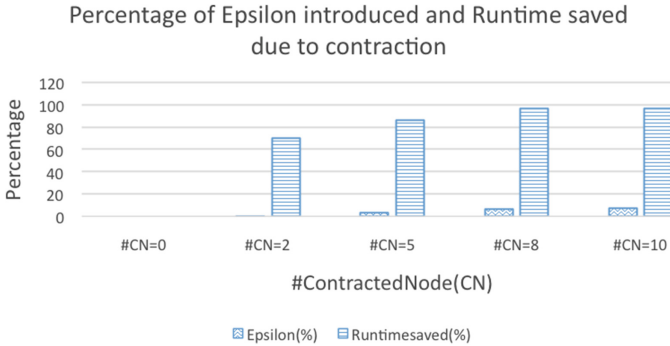


Fig. 6. Effect of contraction on Epsilon and runtime saved

graph has $y\%$ density, we set the same utility for the defender and attacker for 10 % of targets in a graph. The rest of the targets have 0 utility, meaning that those targets are insignificant. Figure 7 shows that when we have a small number of significant targets contraction results in better solution quality. This indicates that contraction will be a particularly effective method for games with relatively few important areas to protect relative to the size of the geographic area, which is often the case for GSG. The reason why the expected utility decreased as we increased density is we kept the distance limit for the patrolling fixed.

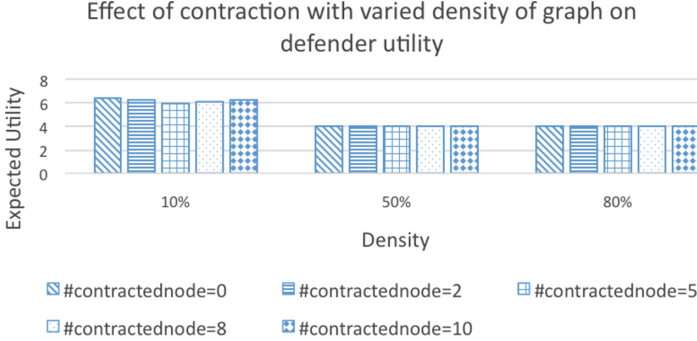


Fig. 7. Effect of contraction for different graph densities

In the next experiment result shown in Figs. 8 and 9 we consider how increasing patrolling distance can affect the solution quality with different levels of contractions and how much time was saved by using contraction. For different level of abstraction the number of contracted nodes used were : (0, 2, 5, 8, 10). The number of resources was fixed at 3. For the baseline algorithm, there was no epsilon because it provides the optimal solution and is what we compare the solution quality of the graph contraction approach to. One key observation is as the distance increases, the percentage of epsilon increased. This is because our resource was fixed but when maximum distance increased we had more joint patrolling paths to consider. This had the effect of introducing more error, but we were able to save a huge amount of time as shown in Fig. 9.

For our final experiment we kept the same experimental setup as the previous experiment except this time we kept the maximum distance fixed at 6 and we varied the number of resources. For the number of resources we considered (1, 2, 3). The results are shown in Figs. 10 and 11. We notice in Fig. 8 that epsilon decreased as we increased our resources. This is expected behavior: as we increase our resources there will be less error introduced, and we were able to save more time(Fig. 9) when resources were increased.

7.1 Discussion and Future Work

Our experimental results demonstrate that using graph contraction as a method for abstraction in green security games has significant potential to increase the

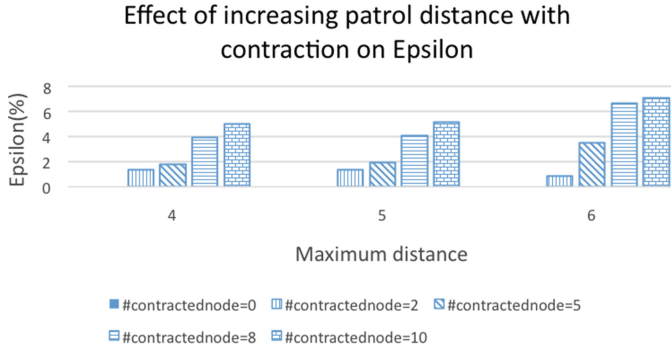


Fig. 8. Effect of contraction on Epsilon with increasing distance

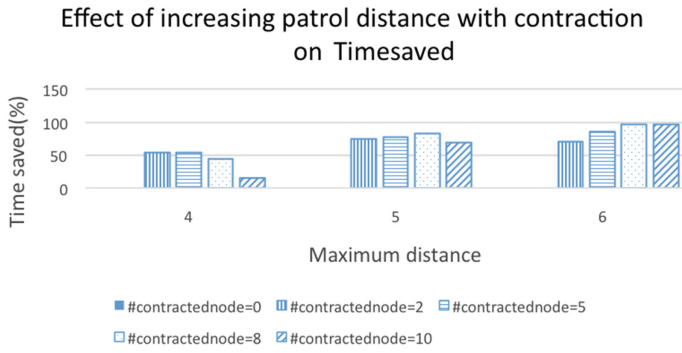


Fig. 9. Effect of contraction on runtime saved with increasing distance

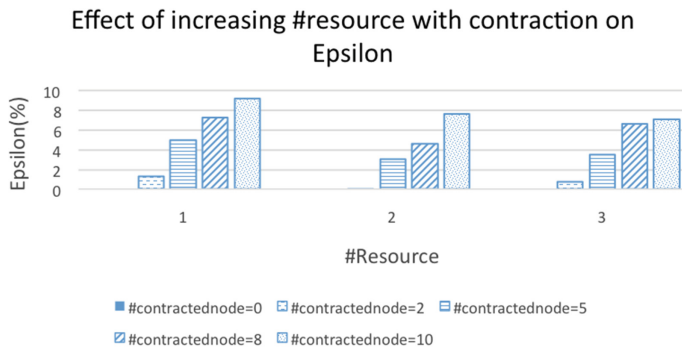


Fig. 10. Effect of increasing patrolling resource with contraction in epsilon

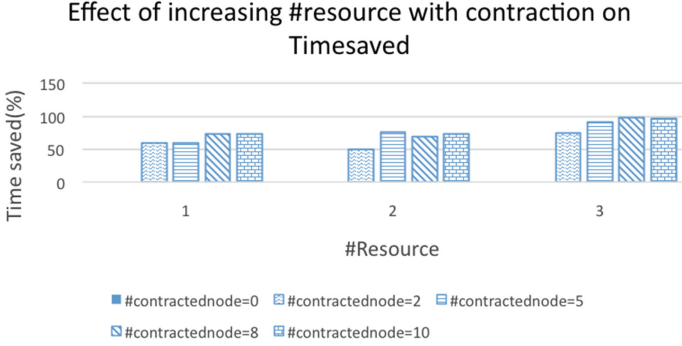


Fig. 11. Effect of increasing patrolling resource on runtime

scalability of solution methods for this class of games. However, we also note several issues with the current methods, and opportunities for improvement in future work.

One of the important issues in the contraction procedure is the selection of targets we want to contract. Suppose a target has been contracted. It can happen in the current procedure that the target was not used in any edge between two targets, which are not contracted. That means the target will not be covered when we use the reverse mapping function. In this scenario for every joint patrolling path J_m with positive probability attacker will receive the reward, and the defender will always have penalty since the target is not covered at all. This kind of situation may be avoidable if we design a more sophisticated method for determining which targets to contract.

Another issue is there might be contraction which can result in very long edges. The difficulty arises if all the edges from base target are further away than the distance constraint. We will not be able to generate any legal path in this case. This kind of scenario can be avoided by not doing contraction if the edge results an edge longer than a threshold.

A final part of our contraction method that could be improved is when we need to find out whether (v, t, v') is the shortest path. We use a shortest path finding algorithm to find another shortest path which does not use t . This shortest path algorithm continues until v' can be reached using less distance than (v, t, v') . As the number of nodes grows, this becomes a problem for runtime. One solution of this problem is to reduce the search space. For example we can only compare edges between neighbors of the contracted node.

8 Conclusion

Stackelberg security games are currently being used in real-world green security domains. In this domain the defenders try to find out an efficient patrolling strategy to prevent illegal wildlife poaching. One of the big issues of this approach, modeling the scenario with GSG, is scalability. In reality, wild animals

spread over very large area. As a result, the problem space becomes huge to solve within feasible time. In this paper we focused on improving the scalability of methods for solving a general class of graph-based security games that includes green security games in particular. We showed that using graph contraction as a method of abstraction can speed up the solution process by a huge amount. With further work to refine the abstraction method, we believe that we can achieve a very large improvement in the scalability for green security games.

References

1. Govt of Mozambique announces major decline in national elephant population (May 2015). <http://press.wcs.org/News-Releases/articleType/ArticleView/articleId/6760/Government-of-Mozambique-Releases-Elephant-Population-Numbers.aspx>
2. The IUCN Red List of threatened species, April 2015. <http://www.iucnredlist.org/>
3. Estimate of global financial losses due to illegal fishing, February 2016. <http://www.worldwildlife.org/threats/illegal-fishing>
4. Batz, G.V., Geisberger, R., Neubauer, S., Sanders, P.: Time-dependent contraction hierarchies and approximation. In: Festa, P. (ed.) SEA 2010. LNCS, vol. 6049, pp. 166–177. Springer, Heidelberg (2010). doi:[10.1007/978-3-642-13193-6_15](https://doi.org/10.1007/978-3-642-13193-6_15)
5. Billings, D., Burch, N., Davidson, A., Holte, R., Schaeffer, J., Schauenberg, T., Szafron, D.: Approximating game-theoretic optimal strategies for full-scale poker. In: The International Joint Conference on Artificial Intelligence (IJCAI), pp. 661–668 (2003)
6. Bowling, M., Burch, N., Johanson, M., Tammelin, O.: Heads-up limit hold'em poker is solved. *Science* **347**(6218), 145–149 (2015)
7. Breton, M., Alj, A., Haurie, A.: Sequential Stackelberg equilibria in two-person games. *J. Optim. Theory Appl.* **59**(1), 71–97 (1988)
8. Brown, M., Haskell, W.B., Tambe, M.: Addressing scalability and robustness in security games with multiple boundedly rational adversaries. In: Poovendran, R., Saad, W. (eds.) GameSec 2014. LNCS, vol. 8840, pp. 23–42. Springer, Heidelberg (2014). doi:[10.1007/978-3-319-12601-2_2](https://doi.org/10.1007/978-3-319-12601-2_2)
9. Brown, N., Ganzfried, S., Sandholm, T.: Hierarchical abstraction, distributed equilibrium computation, and post-processing, with application to a champion no-limit Texas hold'em agent. Technical report (2014)
10. Fang, F., Nguyen, T.H., Pickles, R., Lam, W.Y., Clements, G.R., An, B., Singh, A., Tambe, M., Lemieux, A.: Deploying paws: field optimization of the protection assistant for wildlife security. In: Proceedings of the Innovative Applications of Artificial Intelligence (IAAI) (2016)
11. Fang, F., Stone, P., Tambe, M.: When security games go green: designing defender strategies to prevent poaching and illegal fishing. In: International Joint Conference on Artificial Intelligence (IJCAI) (2015)
12. Field, C., Laws, R.: The distribution of the larger herbivores in the Queen Elizabeth National Park, Uganda. *J. Appl. Ecol.* 273–294 (1970)
13. Ganzfried, S., Sandholm, T.: Potential-aware imperfect-recall abstraction with earth mover's distance in imperfect-information games. In: Conference on Artificial Intelligence (AAAI) (2014)

14. Geisberger, R., Sanders, P., Schultes, D., Delling, D.: Contraction hierarchies: faster and simpler hierarchical routing in road networks. In: McGeoch, C.C. (ed.) WEA 2008. LNCS, vol. 5038, pp. 319–333. Springer, Heidelberg (2008). doi:[10.1007/978-3-540-68552-4_24](https://doi.org/10.1007/978-3-540-68552-4_24)
15. Geisberger, R., Sanders, P., Schultes, D., Vetter, C.: Exact routing in large road networks using contraction hierarchies. *Transp. Sci.* **46**, 388–404 (2012)
16. Gilpin, A., Sandholm, T.: 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), vol. 21, p. 1007 (2006)
17. Gilpin, A., Sandholm, T.: Better automated abstraction techniques for imperfect information games, with application to texas hold'em poker. In: International Foundation for Autonomous Agents and Multiagent Systems (AAMAS), p. 192 (2007)
18. Gilpin, A., Sandholm, T.: Lossless abstraction of imperfect information games. *J. ACM (JACM)* **54**(5), 25 (2007)
19. Gilpin, A., Sandholm, T., Sørensen, T.B.: 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), vol. 22, p. 50 (2007)
20. Gilpin, A., Sandholm, T., Sørensen, T.B.: A heads-up no-limit texas hold'em poker player: discretized betting models and automatically generated equilibrium-finding programs. In: International Conference on Autonomous Agents and Multiagent Systems (AAMAS), pp. 911–918 (2008)
21. Haskell, W.B., Kar, D., Fang, F., Tambe, M., Cheung, S., Denicola, E.: Robust protection of fisheries with compass. In: Association for the Advancement of Artificial Intelligence (AAAI), pp. 2978–2983 (2014)
22. Jain, M., Kardes, E., Kiekintveld, C., Ordóñez, F., Tambe, M.: Security games with arbitrary schedules: a branch and price approach. In: Association for the Advancement of Artificial Intelligence (AAAI) (2010)
23. Kiekintveld, C., Jain, M., Tsai, J., Pita, J., Ordóñez, F., Tambe, M.: Computing optimal randomized resource allocations for massive security games. In: Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-, vol. 1, pp. 689–696. International Foundation for Autonomous Agents and Multiagent Systems (AAMAS) (2009)
24. Kroer, C., Sandholm, T.: Extensive-form game abstraction with bounds. In: Proceedings of the Fifteenth ACM Conference on Economics and Computation, pp. 621–638 (2014)
25. Leitmann, G.: On generalized stackelberg strategies. *J. Optim. Theory Appl.* **26**(4), 637–643 (1978)
26. Nguyen, T.H., Fave, F.M.D., Kar, D., Lakshminarayanan, A.S., Yadav, A., Tambe, M., Agmon, N., Plumptre, A.J., Driciru, M., Wanyama, F., Rwetsiba, A.: Making the most of our regrets: regret-based solutions to handle payoff uncertainty and elicitation in green security games. In: Khouzani, M.H.R., Panaousis, E., Theodorakopoulos, G. (eds.) GameSec 2015. LNCS, vol. 9406, pp. 170–191. Springer, Heidelberg (2015). doi:[10.1007/978-3-319-25594-1_10](https://doi.org/10.1007/978-3-319-25594-1_10)
27. Paruchuri, P., Pearce, J.P., Marecki, J., Tambe, M., Ordóñez, F., Kraus, S.: 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, vol. 2, pp. 895–902. International Foundation for Autonomous Agents and Multiagent Systems (AAMAS) (2008)

28. Pita, J., Bellamane, H., Jain, M., Kiekintveld, C., Tsai, J., Ordóñez, F., Tambe, M.: Security applications: lessons of real-world deployment. *ACM SIGecom Exchanges* **8**(2), 5 (2009)
29. Sandholm, T., Singh, S.: Lossy stochastic game abstraction with bounds. In: *Proceedings of the 13th ACM Conference on Electronic Commerce*, pp. 880–897 (2012)
30. Shieh, E., An, B., Yang, R., Tambe, M., Baldwin, C., DiRenzo, J., Maule, B., Meyer, G.: Protect: a deployed game theoretic system to protect the ports of the united states. In: *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems*, vol. 1, pp. 13–20. International Foundation for Autonomous Agents and Multiagent Systems (AAMAS) (2012)
31. Skiena, S.: Dijkstra’s Algorithm. *Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica*, pp. 225–227. Addison-Wesley, Reading (1990)
32. Storandt, S.: Route planning for bicycles-exact constrained shortest paths made practical via contraction hierarchy. In: *International Conference on Automated Planning and Scheduling (ICAPS)*, vol. 4, p. 46 (2012)
33. Tsai, J., Kiekintveld, C., Ordóñez, F., Tambe, M., Rath, S.: Iris-a tool for strategic security allocation in transportation networks (2009)
34. Von Stengel, B., Zamir, S.: Leadership with commitment to mixed strategies (2004)
35. Yang, R., Ford, B., Tambe, M., Lemieux, A.: Adaptive resource allocation for wildlife protection against illegal poachers. In: *Proceedings of the 2014 International Conference on Autonomous Agents and Multi-agent Systems*, pp. 453–460. International Foundation for Autonomous Agents and Multiagent Systems (AAMAS) (2014)
36. Yang, R., Jiang, A.X., Tambe, M., Ordóñez, F.: Scaling-up security games with boundedly rational adversaries: a cutting-plane approach. In: *The International Joint Conference on Artificial Intelligence (IJCAI)* (2013)
37. Yin, Z., Jiang, A.X., Tambe, M., Kiekintveld, C., Leyton-Brown, K., Sandholm, T., Sullivan, J.P.: Trusts: scheduling randomized patrols for fare inspection in transit systems using game theory. *AI Mag.* **33**(4), 59 (2012)
38. Zinkevich, M., Johanson, M., Bowling, M., Piccione, C.: Regret minimization in games with incomplete information. In: *Advances in Neural Information Processing Systems (NIPS)*, pp. 1729–1736 (2007)

Autonomous Agents and Multiagent Systems

AAMAS 2016 Workshops, Visionary Papers, Singapore,

Singapore, May 9-10, 2016, Revised Selected Papers

Osman, N.; Sierra, C. (Eds.)

2016, XII, 197 p. 71 illus., Softcover

ISBN: 978-3-319-46839-6