Chapter 9
Security Games with Restricted Strategies: An Approximate Dynamic Programming Approach

C.M. Laan, A.I. Barros, R.J. Boucherie and H. Monsuur

Contents

Abstract In this chapter we consider a security game between an agent and an intruder to find optimal strategies for patrolling against illegal fishery. When patrolling large areas that consist of multiple cells, several aspects have to be taken into account. First, the current risk of the cells has to be considered such that cells with high risk are visited more often. Moreover, it is important to be unpredictable in order to increase the patrolling effectiveness countering illegal fishery. Finally,

C.M. Laan (✉) · H. Monsuur
Netherlands Defence Academy, Den Helder, The Netherlands
e-mail: c.m.laan@utwente.nl

H. Monsuur
e-mail: h.monsuur@mindef.nl

C.M. Laan · A.I. Barros
TNO, The Hague, The Netherlands
e-mail: ana.barros@tno.nl

C.M. Laan · R.J. Boucherie
University of Twente, Enschede, The Netherlands
e-mail: r.j.boucherie@utwente.nl

H. Monsuur et al. (eds.), *NL ARMS Netherlands Annual Review of Military Studies 2018*, NL ARMS, https://doi.org/10.1007/978-94-6265-246-0_9

patrolling strategies have to be chosen in such a manner that they satisfy given operational requirements. For example, the agent might be required to patrol some cells more often than others imposing extra restrictions on the agent strategies. In this chapter, we develop a dynamic variant of the security game with restrictions on the agent's strategy so that all these requirements are taken into account. We model this game as a stochastic game with a final penalty to ensure that the operational requirements are met. In this way, strategies are formed that both consider past actions and expected future risk levels. Due to the curse of dimensionality, these stochastic games cannot be solved for large scale instances. We develop an approximate dynamic programming algorithm to find approximate solutions.

Keywords Stochastic games · Security · Patrolling · Restricted strategies · Approximate dynamic programming · Aggregation

9.1 Introduction

The coast guard is responsible for patrolling the coastal waters. Patrolling strategies should be unpredictable, cover the entire region, and must satisfy operational requirements on e.g. the frequency of visits to certain vulnerable parts of the region (cells). We develop a special security game dealing with the protection of a large area in which the agent's strategy set is restricted. This area consists of multiple cells that have to be protected during a fixed time period. The agent has to decide on a patrolling strategy, which is constrained by governmental requirements that establish a minimum number of visits for each cell. Some cells have to be visited more often than others because these regions are more vulnerable. For example, cells close to a port have to be visited more often. A static version of this model is discussed in Laan et al.,[1] where a strategy for the complete time period is identified before the game starts. The requirements are modeled in such a way that they are met with high probability. However, this model does not allow patrolling strategies adjusted to the current situation. In this chapter, we consider a dynamic approach to the security game with restricted strategies in which the agent decides on his strategy for each day taking into account expected future rewards. This allows finding a more flexible strategy for the agent, where current payoffs and number of visits to each cell can be taken into account.

An example application of this model lies in countering illegal or unreported and unregulated fishing. These illicit activities endanger the economy of the fishery sector, fish stocks and the marine environment and require the monitoring of large areas with scarce resources subject to national regulations. To support the development of patrols against illegal fishing, in Haskell[2] a decision support system is developed. This system models the interaction between different types of illegal fishers and the

[1] Laan et al. 2017.

[2] Haskell et al. 2014.

patrolling forces as a repeated game. Moreover, in order to cope with the uncertainty on the adversary's strategy a robust optimization approach is used. More recently, Fang et al.[3] introduced a new game theoretical approach, the green security games, wherein a generalization of Stackelberg games is used to derive sequential agent strategies that learn from adversary behavior. However, in these papers restrictions to the patroller's strategy are not considered.

We model the dynamic variant of the security game with restricted strategies as a finite-time stochastic game in which the state depends on both the current payoff matrix and the remaining minimum number of visits left to each cell. The direct reward is given by the intruder's payoff and at the end of the time period a penalty is given if the operational requirements are not met. Solving stochastic games can be done by iterating over all states and time periods. However, the state space grows exponentially in the number of cells and we are unable to solve realistic sized games. Therefore, we develop an approximate dynamic programming (ADP) approach to find approximate solutions.

Due to the curse of dimensionality, many stochastic optimization models cannot be solved by iterating over all possible states. ADP is a technique that can be used to solve large scale Markov decision processes (MDPs). We develop an ADP framework to find approximate solutions for our stochastic game. A brief introduction in ADP can be found in Powell[4] and various examples are given in Mes and Rivera.[5] In the ADP framework, the optimal solutions are not found using standard backward dynamic programming, but by using a forward dynamic programming approach over only a fixed number of iterations. In this forward approach, different value function approximations can be used. In this chapter, we use multiple aggregation levels of the state space to approximate the value functions as discussed in George et al.[6] In the basic ADP algorithm, only a very limited number of states will be updated during each iteration. In our method using aggregation of the state space, multiple value function approximations are updated at the same time, possibly with different weight for different aggregation states. In this way, the value functions are updated more often and will converge faster.

Although most of the research in ADP focuses on solving MDPs, some models focus on solving games. In Perolat et al.,[7] the authors consider the error propagation for different approximation schemes of zero-sum stochastic games. However, this chapter does not provide a clear procedure that can be used to solve stochastic games using ADP. A solution technique that is very similar to ADP is reinforcement learning (RL), see for example Bucsoniu.[8] The main difference between ADP and RL is that RL is considered to be model-free, which means that information about transition probabilities is not necessarily required. In the field of RL, there is also limited

[3] Fang et al. 2015.

[4] Powell 2010.

[5] Mes and Rivera 2017.

[6] George et al. 2008.

[7] Perolat et al. 2015.

[8] Buşoniu et al. 2010.

research about applications to stochastic games. In Lin et al.,[9] the authors use RL to approximate unknown rewards and use an iterative algorithm to find a policy for both players, while we are interested in calculating this policy using approximation algorithms.

The main contribution of this chapter is twofold. First, we develop a model to solve security games with restrictions on the agent's strategy. Formulating this model as a stochastic game enables the agent to adjust the strategy to the current situation and actions that already have been chosen in the past. Second, we approximate optimal solutions of this stochastic game via an ADP approach. We adjust the standard ADP model that is often used to solve large scale MDPs to analyze stochastic games. Experimental results show that this method gives better payoffs for the agent's than using a static approach where strategies are fixed for the complete planning period.

The remainder of this chapter is organized as follows. In Sect. 9.2, we introduce the model and give the elements of the stochastic game. In Sect. 9.3, we first give a brief introduction in ADP and then describe our formulation for stochastic games. In Sect. 9.4, we give computational results and compare the static and dynamic approach. Finally in Sect. 9.5, we summarize the main findings and provide directions for future research.

9.2 Model Description

In this section, we give the formulation of the security game with restrictions on the agent's strategy. We first describe the basic model in Sect. 9.2.1. In Sect. 9.2.2, we briefly explain the solution method that is used to solve the game with a static strategy for the complete time period as studied in Laan et al.[10] In Sect. 9.2.3, we describe a new stochastic game approach which is used to enable strategies over the entire planning period.

9.2.1 Basic Model

The game is played between an agent and an intruder over a time period of N_D days. The area is given by a finite set of cells $C = \{1, \ldots, N_C\}$. Each day, an intruder selects one cell to attack while the agent chooses a route from a finite set of routes $R = \{1, \ldots, N_R\}$. The agent and intruder choose their action simultaneously. Routes consist of multiple cells where the agent is allowed to move between adjacent cells. The matrix A indicates which cells are visited by each route, such that a_{ij} equals 1 if route i visits cell j and 0 otherwise.

[9] Lin et al. 2017.

[10] Laan et al. 2017.

The risk of a cell is displayed in the payoff matrices. Cells with a high risk have higher payoffs than low risk cells. This payoff is interpreted as the intruders gain: the higher the gain for the intruder, the higher the probability that the intruder will attack there. The payoff matrix can change over time due to, e.g., weather conditions or seasonal fluctuations resulting in multiple payoff matrices. We assume that we have some information about how these payoff matrices change. Let $M^{(k)}$ be the k-th payoff matrix of size $N_R \times N_C$ out of a finite set of payoff matrices, $k = 1, \ldots, N_M$. The element $m_{ij}^{(k)}$ is the expected payoff if the agent uses route i and the intruder attacks cell j, $i = 1, \ldots, N_R$, $j = 1, \ldots, N_C$. We consider the payoff given by the intruder's expected gain:

$$m_{ij}^{(k)} = \left((1 - d_j)a_{ij} + (1 - a_{ij})\right) g_j^{(k)}, \quad i = 1, \ldots, N_R, \ j = 1, \ldots, N_C, k = 1, \ldots, N_M,$$

where d_j is the detection probability for cell j and $g_j^{(k)}$ is the intruder's gain if the intruder successfully attacks cell j. If the agent successfully intercepts the intruder, the payoff is 0.

There are operational requirements on the number of visits to the cells: the agent's strategy is restricted by the requirements that impose a minimum number of visits v_j for each cell j, $j = 1, \ldots, N_C$. During the time period, the agent has to decide on his actions such that cell j is visited at least j times.

Note that the model described in this chapter only describes a basic security game with one intruder. The methods developed in this chapter may be applied to extensions to matrix games obtained by changing the payoff matrices, such as including more (cooperating) intruders or detection probabilities depending on the cell or chosen action.

9.2.2 Static Approach

When using a static approach, the strategies are the same for each time period, but might be different for each payoff matrix $M^{(k)}$. The strategy of the agent is $p^T = (p^{(1)}, \ldots, p^{(N_M)})$, where $p_i^{(k)}$ is the probability that route i is chosen when the payoff matrix is $M^{(k)}$. The strategy of the intruder is $q^T = (q^{(1)}, \ldots, q^{(N_M)})$, where $q_j^{(k)}$ is the probability that cell j is attacked when the payoff matrix is $M^{(k)}$. The probability that the payoff matrix is $M^{(k)}$ equals $\mu^{(k)}$.

The restrictions to the agent's strategy are modeled by the constraint $f(p) \geq 1 - \epsilon$. The function $f(p)$ gives the probability that all the agent's restrictions on the minimum of visits for the cells are met, given the agent's strategy p. Randomized strategies are used to guarantee the unpredictability of the patrolling. Therefore, it is not possible to demand that the requirements are always met. By requiring $f(p) \geq 1 - \epsilon$, we guarantee that the requirements are met with high probability. For the experiments in this chapter, we use $\epsilon = 0.05$.

The value of the game is the expected payoff per day and can be found by solving the following optimization problem:

$$
\begin{aligned}
V = \min_{p} \max_{q} \quad & \sum_{k=1}^{N_M} \mu^{(k)} (p^{(k)})^T M^{(k)} q^{(k)} \\
\text{s.t.} \quad & f(p) \geq 1 - \epsilon, \\
& \sum_{i=1}^{N_R} p_i^{(k)} = 1, \quad k = 1, \ldots, N_M, \\
& \sum_{i=1}^{N_C} q_i^{(k)} = 1, \quad k = 1, \ldots, N_M, \\
& p, q \geq 0.
\end{aligned} \tag{9.1}
$$

Taking the dual of the inner linear program, $\max_q \{\sum_{k=1}^{N_M} \mu^{(k)} (p^{(k)})^T M^{(k)} q^{(k)} \mid \sum_{j=1}^{N_C} q_j^{(k)} = 1, k = 1, \ldots, n_M, q \geq 0\}$, the minmax formulation from (9.1) can be rewritten as a minimization problem to obtain the game value and optimal strategies for the agent.

We use approximations to determine $f(p)$. We explain the basic idea of these approximations and refer to Laan et al.[11] for a more detailed description. Consider the game with only one payoff matrix, so we omit the index k. Let $Y = (Y_1, \ldots, Y_{N_R})$ be the number of times that each route is chosen by the agent. The random variable Y is multinomially distributed with parameters p and N_D, where p is the probability distribution over the routes and N_D is the length of the planning period. For large N_D, Y can be approximated by a multivariate distribution with mean $N_D p_i$ and variance $N_D p_i (1 - p_i)$ and covariance $-N_D p_i p_{i'}$, $i, i' = 1, \ldots, N_R$. Let $X = (X_1, \ldots, X_{N_C})$ be the number of times that each cell is visited:

$$
X_j = \sum_{i=1}^{N_R} a_{ij} Y_i.
$$

Using the approximation of Y, X can be approximated by a multivariate normal distribution with mean $N_D a_j p$, variance $N_D a_j p (1 - a_j p)$ and covariance $\sum_{i=1}^{N_R} \sum_{i'=1}^{N_R} a_{ij} a_{i'j'} Cov(\tilde{Y}_i, \tilde{Y}_{i'})$, $j, j' = 1, \ldots, N_C$ (see Ross).[12] Now, the probability that the requirements are met can be calculated using the cumulative distribution for the multivariate normal distribution, but this function is difficult to implement. A lower bound for the probability that all requirements are met is:

[11] Laan et al. 2017.

[12] Ross 1996, Chapters 1.4 and 1.8.

$$f(p) \geq 1 - \sum_{j=1}^{N_C} \Phi\left(\frac{v_j - N_D a_j p}{\sqrt{N_D a_j p(1 - a_j p)}}\right),$$

where $\Phi(x)$ is the cumulative distribution of the standard normal distribution. Implementing this function in (9.1) gives an upper bound for the game value. Experimental results show that the error that is made by this lower bound is small ($\leq$2%).[13] A similar expression for $f(p)$ can be derived for the case with multiple payoff matrices.

9.2.3 Dynamic Approach

When considering a dynamic approach, strategies can change during the time window depending on the current payoff matrix and the number of times each cell already has been visited. We model this as a finite-time zero-sum stochastic game. We now describe the elements of this game.

The state space S of the game is given by the current payoff matrix and the number of visits that are still required for each cell:

$$S = \{s | s = (k, \bar{v}_1, \ldots, \bar{v}_{N_C}),\ k = 1, \ldots, N_M,\ 0 \leq \bar{v}_j \leq v_j,\ j \in C\}.$$

The action space of the agent and intruder are given by A_A and A_I. The intruder attempts to maximize the payoff by choosing which cell to attack, so the action set of the intruder is given by C. The agent tries to catch the intruder by selecting a route, so the action set of the agent is given by R:

$$A_A = R, \quad A_I = C.$$

The matrix T gives the transitions between the payoff matrices. These transitions do not depend on the actions of the agent and the intruder. If the current payoff matrix is $M^{(k)}$, then with probability t_{kl} the next payoff matrix is $M^{(l)}$. The transition matrix of the game P depends on both T and the action i of the agent:

$$P(s'|s, i) = \begin{cases} t_{kl}, & \text{if } \bar{v}'_j = \max\{\bar{v}_j - a_{ij}, 0\},\ \text{for all } j \in C, \\ 0, & \text{otherwise.} \end{cases}$$

where $s = (k, \bar{v}_1, \ldots, \bar{v}_{N_C})$ and $s' = (l, \bar{v}'_1, \ldots, \bar{v}'_{N_C})$ and i is the agents action.

The direct reward is given by R and depends on the agent's strategy i, the intruder's strategy j and the current payoff matrix $M^{(k)}$:

$$R(s, (i, j)) = m_{ij}^{(k)}.$$

[13] Laan et al. 2017.

To ensure that the requirements are met, we introduce a final reward which is a penalty for the requirements that are not met. This can either be a penalty for each requirement that is not met or one penalty if the requirements are not met. For the results considered in this chapter we consider the last option:

$$R_f(s) = \begin{cases} B, & \text{if } \sum_{j \in C} \bar{v}_j > 0, \\ 0, & \text{otherwise,} \end{cases}$$

where B is chosen large enough such that it is never beneficial to violate one of the requirements.

Optimal strategies can be found by solving the game iterative (see Owen).[14] Let $V_t(s)$ be the game value at time period t, when the game is in state s.

$$V_{N_D}(s) = \text{Val}\left(M^{(k)} + \sum_{s' \in S} P(s'|s, \cdot) R_f(s') \right), \tag{9.2}$$

$$V_t(s) = \text{Val}\left(M^{(k)} + \sum_{s' \in S} P(s'|s, \cdot) V_{t+1}(s') \right), \quad t < N_D. \tag{9.3}$$

where $s = (k, \bar{v}_1, \ldots, \bar{v}_{N_C})$, so $M^{(k)}$ depends on the first element of state s, and $P(s'|s, \cdot)$ is the matrix consisting of the values $P(s'|s, i)$ for all agent's actions. The expression between brackets defines a matrix game. Val gives the value of this matrix game, so this is the game value when both players choose a strategy corresponding to a Nash equilibrium.

Solving Eqs. (9.2) and (9.3) will give an optimal value of the game. However, the size of the state space is exponentially increasing in the number of cells and conditions and we are unable to solve these equations analytically. In the next section, we present a model to deal with this large state space.

9.3 Solution Approach: Approximate Dynamic Programming

In this section, we present a method that can be used to overcome the large state space of the stochastic game formulation in Sect. 9.2. Approximate dynamic programming (ADP) is a technique that is often used to solve large scale MDPs. In Sect. 9.3.1, we give a short introduction in ADP for solving MDPs based on Powell.[15] In Sect. 9.3.2, we develop ADP to solve our stochastic game.

[14] Owen 1995, Chapter V.3.

[15] Powell 2010.

9.3.1 Introduction to ADP

Consider an MDP over time horizon T, with states s_t, actions a_t, transition matrix P, and cost functions C_t. The value of an MDP can be found by solving the Bellman equations:

$$V_t(s_t) = \min_{x_t} \left(C_t(s_t, x_t) + \sum_{s_{t+1}} P(s_{t+1}|s_t, x_t) V_{t+1}(s_{t+1}) \right).$$

When the state space is large, solving the Bellman equations is too time consuming. The main idea of ADP is not to solve the model by enumerating over all possible solutions but only over a limited number of states using a forward dynamic programming approach over a fixed number of iterations N. For each iteration, the random information is sampled using Monte Carlo experiments.

The random information that is revealed after action a_t is chosen is given by w_{t+1}. Both the action and the random information define the next state. For ADP, the post-decision state s_t^a is introduced. A post-decision state is the state after an action a_t is chosen, but before the new random information w_{t+1} is revealed:

$$V_t(s_t) = \min_{a_t} \left(C_t(s_t, a_t) + V_t^a(s_t^a) \right),$$
$$V_t^a(s_t^a) = \sum_{w_{t+1}} P(w_{t+1}) V_{t+1}(s_{t+1}|s_t^a, w_{t+1}).$$

By the use of the post-decision state, we only have to evaluate the possible outcomes over w_{t+1} for each action and not over all possible states s_{t+1}. This decreases the number of possible outcomes that have to be evaluated during each iteration significantly.

The output of the algorithm is an approximation $\bar{V}_t^n(s_t^a)$ of the value of the post-decision states. During each step, the approximation $\bar{V}$ is updating using the following update rule:

$$\bar{V}_t^n(s_t) = \begin{cases} (1-\alpha)\bar{V}_t^{n-1}(s_t^n) + \alpha \hat{v}_t^n, & \text{if } s_t = s_t^n, \\ \bar{V}_t^{n-1}(s_t), & \text{otherwise.} \end{cases} \tag{9.4}$$

where α is a step size between 0 and 1. In the next section, we discuss the value of α.

The basic structure of an ADP is given by the following algorithm.

Algorithm 1 ADP algorithm

1: Initialize:

- Choose an initial approximation $\bar{V}_t^0(s_t^a)$ for each t.
- Set $n = 1$ and choose an initial state s_0^1.

2: Choose a sample path $w^n = (w_1^n, \ldots, w_T^n)$.
3: For $t = 0, \ldots, T$

- Solve

$$\hat{v}_t^n = \min_{a_t} \left(C_t(s_t^n, a_t) + \mathbb{E}^w \left(\bar{V}_{t+1}^{n-1}(s_t | s_t^{a_t}, w) \right) \right),$$

  and let a_t^n be the action that solves this minimization.
- Update $\bar{V}_t^n(s_t)$ using (9.4).
- Compute the next state to visit from the action a_t^n.

4: Set $n = n + 1$ and go to Step 2.

This algorithm is a basic outline and will in general not always give good approximation results. There are some methods for improving the algorithm, mainly in the step of choosing the next state (random or not), the choice of α and in the steps of the value function approximation. We discuss these methods in the next section.

9.3.2 ADP for a Stochastic Game

The ADP approach described in the previous section is used to solve large scale MDPs. In this section, we describe the adjustments we make to the ADP to solve the stochastic game in Sect. 9.2.3. The difference is that we deal with multiple players. Therefore, the Bellman equations are replaced by:

$$V_t(s) = \text{Val} \left(M^{(k)} + \sum_{s' \in S} P(s'|s, \cdot) V_{t+1}(s') \right).$$

As a consequence, we have to optimize over both the intruder's and the agent's actions. However, in our case, the next state does not depend on the intruder's action. Therefore, we can use an ADP algorithm similar to the ADP algorithm which is used to solve MDPs.

Due to the introduction of multiple players we are not dealing with discrete actions. Both agent and intruder choose as an action a probability distribution over the action spaces at each time step. Therefore, we are not able to calculate a value for each combination of actions and states. We modify the formulation and use of the post-decision state, which in our case only depends on the agents actions. Let s_t^i be the post-decision state at time t and state s when the agent chooses pure strategy $i \in R$:

$$\bar{V}_t(s_t^i) = \sum_{l=1}^{N_M} t_{kl}\bar{V}_{t+1}\left((l, (s_t(2) - a_{i1})^+, \ldots, (s_t(N_C+1) - a_{iN_C})^+)\right),$$

where $s = (k, \bar{v}_1, \ldots, \bar{v}_{N_C})$.

We now describe the basic ADP algorithm adjusted for our game:

Algorithm 2 ADP algorithm for stochastic game

1: Initialize:

- Choose an initial approximation $\bar{V}_t^0(s_t)$ for each t, s_t.
- Set $n = 1$ and choose an initial state s_0^1.

2: Choose a sample path w^n which describe the payoff matrices.
3: For $t = 0, \ldots, N_D - 1$

- Construct M, such that:

$$m_{ij} = m_{ij}^{(k)} + \bar{V}_{t+1}^{n-1}(s_t^i), \quad t < N_D,$$
$$m_{ij} = m_{ij}^{(k)} + R_f(s_t^i), \quad t = N_D.$$

- Solve

$$\hat{v}_t^n = \text{Val}\,(M),$$

  and let π_t^n be the agent's strategy that solves this minimization.
- Update $\bar{V}_t^n(s_t)$ using (9.4).
- Compute the next state to visit: w.p. β decide on the next state using π_t^n and with probability $1 - \beta$ choose a random action.

4: Set $n = n + 1$ and go to Step 2.

We now discuss the choice of α and β and introduce aggregation, which can be used to speed up the convergence of the algorithm.

Choice of α The value of the step step size α can be chosen in different ways. A review of different step sizes that are used in literature is given in George and Powell.[16] Two popular step sizes that are often used are the harmonic and the polynomial step size.[17] We use a harmonic step size where α depends on the iteration n:

$$\alpha_n = \max\left\{\frac{a}{a+n-1}, \alpha_0\right\},$$

where the value of α_n decreases in the number of iterations. In Sect. 9.4, we conduct experiments to decide on the value of a and α_0.

[16] George and Powell 2006.

[17] Powell 2010.

Choice of β In Step 3 of the ADP algorithm, the next state is chosen. If the next state only depends on the strategy π_t^n, it is possible that some states will never be visited and the algorithm does not converge to the best possible value. To avoid this, a random action is chosen with probability $1 - \beta$. In Sect. 9.4, we show the results of experiments with the value of β.

Aggregation To have a good approximation of the value function of a state, this specific state has to be visited often enough. During one iteration, the value function of only one state is updated and when the number of states is large, a lot of iterations are necessary to ensure a good approximation. There are different methods that can be used to speed up the convergence by updating multiple states per iteration. Two methods that are commonly used are aggregation and the use of basic functions.[18] We use aggregation with multiple aggregation levels which is proven to work well for large scale MDPs.[19] An example of an aggregation level is to only consider the requirements and not the payoff matrix.

Let G be the number of aggregation levels and $S^{(g)}$, $g = 0, \ldots, G$, be the state space corresponding to the g-th aggregation level ($S^{(0)} = S$). The state $s^{(g)}$ is the state corresponding to s in the g-th aggregation level and $\bar{V}^{(g)}(s^{(g)})$ is the value function approximation for this state. The value function approximation of each state s is given by a weighted combination of the value functions of all the corresponding states for the different aggregation levels:

$$\bar{V}^n(s) = \sum_{g=0}^{G} w^{(g,n)}(s)\bar{V}^{(g,n)}(s^{(g)}),$$

where $w^{(g,n)}(s)$ is the weight of the g-th aggregation level for state s. We choose the weight by inverse mean squared errors as described in George et al.[20] using the bias and variance of each estimator:

$$w^{(g,n)}(s) \sim \frac{1}{\frac{(\tilde{\sigma}^{(g)}(s))^2}{N_s^{(g)}} + (\tilde{\mu}_s^{(g)})^2},$$

where $\tilde{\sigma}^{(g)}(s))^2$ is the sample variance of all the observations corresponding to the estimate $\bar{V}^{(g)}(s^{(g)})$, $N_s^{(g)}$ is the number of all these observations and $\tilde{\mu}_s^{(g)}$ is the bias from the true value $\bar{V}^{(0)}(s)$. A detailed description of these computations can be found in George et al.[21] Experiments with different aggregation levels can be found in Sect. 9.4.

[18] Powell 2010.
[19] George et al. 2008.
[20] George et al. 2008.
[21] George et al. 2008.

In this section, we have described a framework to enable us to deal with large scale dynamic games. In the next section, we show computational results to illustrate the performance of the ADP framework.

9.4 Experiments

We have developed a model and solution approach to solve a dynamic variant of security games with restrictions on the agent's strategy. In this section, we perform experiments to see how our model performs. First, we compare the static and dynamic approach in Sect. 9.4.1. In Sect. 9.4.2, we experiment with different input variables of the ADP approach and give computational results. Finally, in Sect. 9.4.3, we explore the model for an instance of realistic size.

9.4.1 Benefits of the Dynamic Approach

To show the benefits of the dynamic approach studied in this chapter, we compare it with the static approach as described in Sect. 9.2.2.

Consider a game with 9 cells and 8 routes as described in Fig. 9.1. In this example, the routes are chosen in such a way that the agent moves right, left or diagonal. The numbers are the cell numbers and the color of the cells correspond to the intruder's gain. The darker the color, the higher the intruder's gain: white cells have a payoff of 1, light gray cells have a payoff of 2 and dark gray cells have a payoff of 3. The transition probabilities for the payoff matrices are:

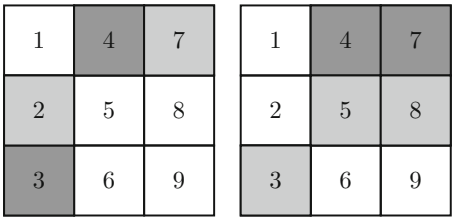
$$T = \begin{bmatrix} 0.7 & 0.3 \\ 0.3 & 0.7 \end{bmatrix}.$$

This means that on average both $M^{(1)}$ and $M^{(2)}$ occur with equal probability, so $\mu^{(1)} = \mu^{(2)} = 0.5$. The time period, N_D, equals 80.

In Table 9.1, the second and third columns show the game value for both the static and dynamic approach for different requirements on the agent's strategy. The value that is given is the expected value per day. The last column gives the running time for the stochastic game. The static game always runs within a second. All experiments in this section are implemented in Matlab version R2016b[22] on an Intel(R) Core(TM) i7 CPU, 2.4 GHz, 8 GB of RAM.

Both the static and the dynamic game are played over a time period of N_D days. Note that the strategies from the static game, can always be recreated using the dynamic approach. When there are no restrictions, the dynamic game gives almost the same strategies as in the static game because previous actions do not influence

[22] MATLAB 2016.



Routes	Cells visited by route
1	1, 5, 9
2	2, 3, 6
3	6, 8, 7
4	1, 4, 7
5	1, 2, 3
6	2, 6, 8
7	3, 5, 6
8	4, 5, 9

Fig. 9.1 Payoff matrices and routes [*Source* C.M. Laan, A.I. Barros, R.J. Boucherie, H. Monsuur]

Table 9.1 Expected payoff per day for different requirements [*Source* C.M. Laan, A.I. Barros, R.J. Boucherie, H. Monsuur]

Requirements	Static	Dynamic	
	Game value	Game value	Running time (s)
None	1.45	1.45	1.01
$v = (0,\ 0,\ 40,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0)$	1.64	1.55	10.38
$v = (0,\ 30,\ 0,\ 0,\ 0,\ 0,\ 20,\ 0,\ 0)$	1.88	1.52	143.81
$v = (0,\ 30,\ 40,\ 0,\ 0,\ 0,\ 20,\ 0,\ 0)$	2.31	1.58	6513.91
$v = (0,\ 0,\ 40,\ 0,\ 0,\ 0,\ 30,\ 0,\ 0)$	–	1.85	275.19

the outcome and, when N_D is large enough, the number of times each payoff matrix appears is approximately the expected value μ as used in the static game. As can be seen in Table 9.1, the dynamic game approach gives better results for the agent when there are restrictions. This is because he has more flexibility in planning his strategy. The agent does not have to plan his complete strategy in advance anymore and can adjust his strategy depending on which routes were chosen before. Moreover, using the stochastic game approach, it is guaranteed that the requirements are met. Another advantage is that we do not have to require that each payoff matrix occur often enough because we do not need to apply the law of large numbers. Also, some requirements give an infeasible solution for the static approach, while they can be met for the dynamic case. This follows from the fact that for the static approach we use randomized strategies that are the same for each time period. To meet the requirements with high probability, the cells have to be visited more often on average than the requirements require. This is not necessary for the dynamic approach, since the strategies can be adjusted to the number of visits in the past. The disadvantage of the stochastic game approach is that the running time increases exponentially in the number of visits.

9.4.2 Computational Results of ADP

In this section, we explore the performance of the ADP approach for the dynamic game. Also, we test different inputs parameters and multiple aggregation levels. Consider the game as described in Sect. 9.4.1 with the requirements $v = (0,\ 30,\ 40,\ 0,\ 0,\ 0,\ 20,\ 0,\ 0)$. This example will be used to illustrate our experiments.

The running times of the experiments in this section depend on the level of aggregations: the more levels of aggregations, the higher the running time. For the case without aggregation, the running time of the experiments is approximately 1000 s and the running time for the case with for levels of aggregation, the running time was approximately 1500 s. For all the experiments, we used 3000 as the number of iterations.

First, we test the model without aggregation for different input parameters: the step size parameters, a and α_0, and the probability that a random action is chosen $1 - \beta$. The value function approximation of the initial state is displayed in Fig. 9.2 for a selection of different combinations of these parameters without aggregation. The ADP algorithm gives value function approximations for each possible state. Also, for each possible state, a strategy is calculated in Step 3 of the algorithm. We test this strategy by simulating the game after different numbers of iterations. The game is simulated 100 times, where the value approximations and strategies obtained by the ADP algorithm are used.

Tables 9.2, 9.3, 9.4 and 9.5 show the results for the model with and without aggregation. Tables 9.2 and 9.4 show the percentage that the requirements are met. In general, it holds that the better the value function approximations are, the higher the probability that the requirements are met. For the dynamic game, it is guaranteed that the conditions are met if this is feasible and the penalty is high enough. However, when using the value function approximations to decide on the strategies, this is not always guaranteed if the approximations are still too far from the optimal values.

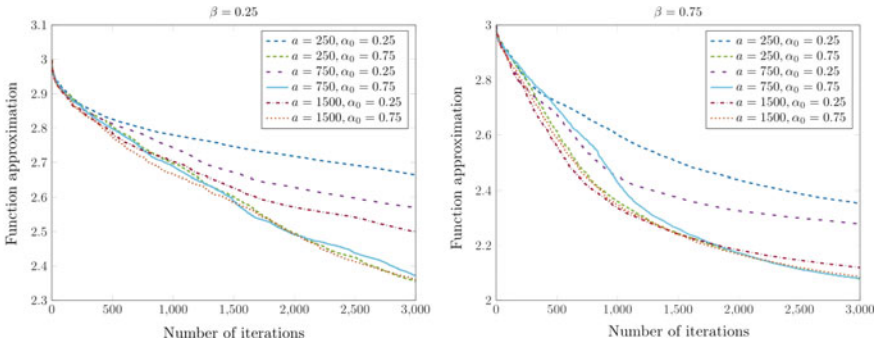


Fig. 9.2 Convergence of ADP for different values of α_0 and a and β, no aggregation [*Source* C.M. Laan, A.I. Barros, R.J. Boucherie, H. Monsuur]

Table 9.2 Percentage requirements satisfied, no aggregation [*Source* C.M. Laan, A.I. Barros, R.J. Boucherie, H. Monsuur]

β	a	250			750			1500		
	Iterations/α_0	0.25	0.5	0.75	0.25	0.5	0.75	0.25	0.5	0.75
0.25	1000	1%	0%	1%	1%	2%	0%	2%	2%	4%
	2000	4%	8%	5%	9%	5%	15%	16%	6%	6%
	3000	13%	8%	7%	17%	7%	5%	30%	14%	4%
0.5	1000	0%	9%	3%	3%	1%	1%	4%	1%	4%
	2000	2%	11%	9%	13%	20%	7%	29%	20%	3%
	3000	6%	20%	26%	18%	11%	68%	47%	18%	4%
0.75	1000	0%	25%	80%	9%	8%	43%	94%	34%	44%
	2000	77%	89%	63%	40%	91%	100%	99%	58%	98%
	3000	88%	100%	96%	99%	100%	99%	100%	99%	100%

Table 9.3 Average game value, no aggregation [*Source* C.M. Laan, A.I. Barros, R.J. Boucherie, H. Monsuur]

β	a	250			750			1500		
	Iterations/α_0	0.25	0.5	0.75	0.25	0.5	0.75	0.25	0.5	0.75
0.25	1000	1.73	–	1.76	1.87	2.04	–	1.92	1.84	2.11
	2000	1.84	1.87	1.98	1.97	1.99	1.93	1.97	1.93	1.82
	3000	1.93	1.91	1.92	1.99	1.90	1.86	1.94	1.92	1.85
0.5	1000	–	2.04	2.01	2.09	2.01	1.89	2.05	2.02	2.05
	2000	1.94	1.97	1.93	2.00	1.99	1.98	1.99	2.04	1.95
	3000	1.97	1.98	1.93	2.00	1.99	1.98	1.94	1.97	1.97
0.75	1000	–	2.20	2.17	2.17	2.07	2.22	2.16	2.22	2.16
	2000	2.23	2.13	2.06	2.13	2.14	2.07	2.07	2.15	2.06
	3000	2.16	2.12	2.00	2.13	2.09	2.00	2.00	2.10	2.00

Tables 9.3 and 9.5 show the average game value for the case that the requirements are met. These tables show that $\beta = 0.75$ gives the best results for all different step values. For the value of the step size, the results are less conclusive. However, higher step sizes gives better value function approximation. The choice of α_0 is hereby more important than the choice of a. This can also be seen in Fig. 9.2.

In Fig. 9.3 the value function approximation for the starting state is shown for both the ADP with and without aggregation ($a = 750$, $\alpha_0 = 0.75$, $\beta = 0.75$). For the case with aggregation, we use 4 aggregation levels. The first level considers the state without payoff matrix, the second level considers the state with only even number of visits left, the third level considers the state with the number of visits divided and rounded to the nearest integer above and the fourth level only considers the total number of visits. We use the first level for the case with one level, the first two for the case with two levels, etc.

Table 9.4 Percentage requirements satisfied, with 3 aggregation levels [*Source* C.M. Laan, A.I. Barros, R.J. Boucherie, H. Monsuur]

β	a	250			750			1500		
	Iterations/α_0	0.25	0.5	0.75	0.25	0.5	0.75	0.25	0.5	0.75
0.25	1000	0%	0%	2%	2%	0%	3%	2%	2%	2%
	2000	3%	9%	3%	7%	1%	9%	2%	5%	11%
	3000	18%	11%	9%	20%	9%	20%	10%	20%	21%
0.5	1000	3%	4%	8%	4%	1%	1%	4%	2%	3%
	2000	24%	29%	57%	14%	9%	6%	33%	27%	49%
	3000	48%	60%	88%	23%	27%	12%	62%	31%	72%
0.75	1000	83%	35%	84%	76%	85%	86%	32%	99%	80%
	2000	98%	94%	100%	41%	100%	100%	95%	100%	99%
	3000	100%	99%	99%	100%	100%	100%	100%	100%	100%

Table 9.5 Average game value, with 3 aggregation levels [*Source* C.M. Laan, A.I. Barros, R.J. Boucherie, H. Monsuur]

β	a	250			750			1500		
	Iterations/α_0	0.25	0.5	0.75	0.25	0.5	0.75	0.25	0.5	0.75
0.25	1000	–	–	2.06	2.04	–	1.97	1.89	2.06	1.93
	2000	1.85	1.88	1.86	1.98	1.91	1.97	1.93	1.91	1.98
	3000	1.91	1.97	1.90	2.01	1.92	2.01	1.96	1.99	1.92
0.5	1000	2.03	2.19	2.01	2.01	2.13	2.10	2.04	2.12	2.04
	2000	2.03	1.98	2.00	1.99	2.07	1.87	1.97	2.02	1.95
	3000	2.02	1.99	1.93	2.00	1.97	1.91	1.98	2.01	1.93
0.75	1000	2.26	2.21	2.11	2.16	2.18	2.13	2.19	2.12	2.17
	2000	2.16	2.09	1.98	2.06	2.08	1.98	2.09	2.01	2.03
	3000	2.11	2.00	1.92	2.03	2.01	1.92	2.07	1.94	1.99

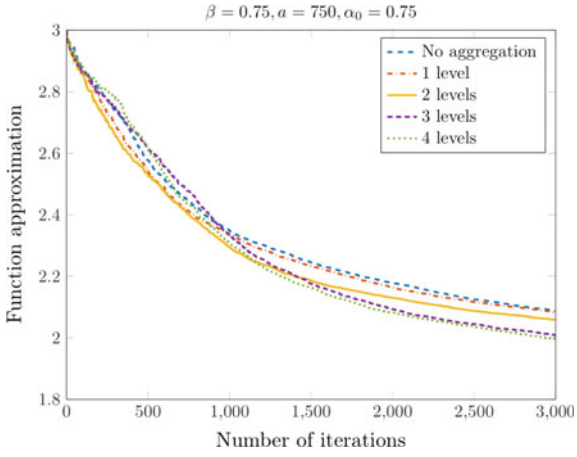


Fig. 9.3 Convergence of ADP for different aggregation levels [*Source* C.M. Laan, A.I. Barros, R.J. Boucherie, H. Monsuur]

Figure 9.3 shows that more aggregation levels give faster convergence. However in this example, the differences between the different aggregation levels are small. This can be explained by the fact that this is a relative small problem and the convergence for the case without aggregation is already fast. In the next section, we discuss a larger instance where it can be seen that the algorithm with aggregation converges significantly faster than without.

Recall the results in Sect. 9.4.1 as shown in Table 9.1. For the game with requirements $v = (0,\ 30,\ 40,\ 0,\ 0,\ 0,\ 20,\ 0,\ 0)$, the game value for the static approach is 2.31 and for the dynamic approach 1.58. In this section, we approximated the dynamic approach solution by using ADP, where we were able to obtain game values of 1.92 (see Table 9.5). These results show that the expected reward of when using the ADP approach is a higher than the optimal value of the stochastic game. However, it still outperforms the static approach. Moreover, this method can be used for larger instances where the stochastic game approach is too computationally expensive.

We tested the ADP approach for different instances, which gave similar results. The ADP approach usually outperforms the static approach, but not always in the cases where the game value of the static and dynamic case are close. This can explained by the fact that in these cases, the requirements do not have a large impact on the optimal strategy, so the static game already gives a solution close to the solution of the dynamic game. Since some approximation error is made in the ADP approach, it might occur that the static game gives a better value. Also, the optimal choice of input parameters vary a bit for different instances, so these have to be chosen carefully depending on the instance. From our computational results, we can say that a high value of β always gives good results and that the value of a needs to be chosen higher for larger instances.

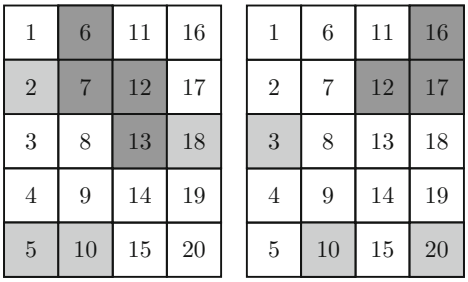
9.4.3 Numerical Results for a Realistic Sized Instance

In this section, we give numerical results of a larger instance of the security game for which we cannot solve the stochastic game to optimality. The size of this instance is comparable to real world sized problems. However, we still consider a limited number of payoff matrices such that we can compare the game values with the game value of the static game as described in Sect. 9.2.2.

Consider the game as described in Fig. 9.4 with two payoff matrices and with requirements on Cells 1–5: $v = (10,\ 30,\ 30,\ 30,\ 10,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0)$. The time period, N_D, is 100. The transition probabilities for the payoff matrices are:

$$T = \begin{bmatrix} 0.3 & 0.7 \\ 0.4 & 0.6 \end{bmatrix},$$

which means that on average, the payoff matrix is $M^{(1)}$ with probability 0.36 and $M^{(2)}$ with probability 0.64. Solving the static game gives a game value of 2.57.



Routes	Cells visited by route
1	1, 2, 3, 8, 13, 18
2	5, 10, 14, 15, 19
3	4, 5, 6, 7, 8, 11
4	4, 9, 12, 13, 14, 17
5	16, 17, 18, 19, 20
6	1, 2, 6, 7, 11, 12
7	3, 8, 11, 12, 13
8	4, 5, 10, 15, 20
9	1, 2, 6, 11, 16
10	3, 8, 14, 19, 20
11	6, 7, 12, 16, 17

Fig. 9.4 Payoff matrices and routes for realistic sized scenario [*Source* C.M. Laan, A.I. Barros, R.J. Boucherie, H. Monsuur]

We ran the ADP algorithm with 4000 iteration with multiple aggregation levels in different configurations. We used the same aggregation levels as described in Sect. 9.4.2 with one additional aggregation level. The fifth aggregation level only considers the maximum number of visits over all cells.

The results for a selection of aggregation level configurations are shown in Fig. 9.5. This figure shows that aggregation ensures convergence a lot faster in this game. For this instance, the best convergence is obtained with levels 1, 2 and 4 combined. With these levels combined, the problems converge faster than without aggregation levels. This can also be seen in Table 9.6. However, not for all aggregation configurations outperform the algorithm without aggregation. For example, only using levels 3, 4 and 5 decreases the convergence speed. This can be explained by the fact that an error is made when aggregation multiple states. Aggregating many states will speed up the convergence, but may also lead to approximations far from the optimal solution. The right choice of aggregation levels depends on the instance and has to be chosen carefully.

Table 9.6 Percentage and average realistic sized scenario ($\beta = 0.75$, $a = 150$, $\alpha_0 = 0.75$) [*Source* C.M. Laan, A.I. Barros, R.J. Boucherie, H. Monsuur]

Iterations	No aggregation		3 levels	
	Average	Percentage (%)	Average	Percentage (%)
1000	–	0	2.47	98
2000	–	0	2.33	100
3000	2.08	83	2.25	100
4000	2.03	82	2.17	100
5000	2.01	98	2.07	100

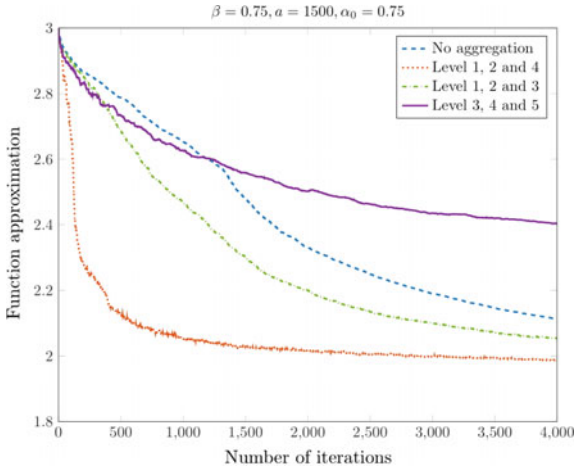


Fig. 9.5 Realistic sized scenario with and without aggregation [*Source* C.M. Laan, A.I. Barros, R.J. Boucherie, H. Monsuur]

9.5 Conclusion

In this chapter, we have developed a model for the dynamic decision making of an agent when his strategy is restricted by operational requirements. We have formulated the problem as a stochastic game and have shown that the use of a dynamic formulation outperforms the model in which strategies cannot be adjusted to the current situation: better game values for the agent can be obtained. Also, the stochastic game formulation can yield a feasible solution when more operational requirements are considered.

The disadvantage of the stochastic game formulation is that the solving time grows exponentially in the number of cells with requirements. This means that we cannot solve the game to optimality for real world instances. For that reason, we have developed an ADP approach to find approximate solutions. ADP is often used to solve large scale MDPs. With a limited number of adjustments, we have been able to develop a similar approach for our stochastic game.

Experimental results show that the game value which is found by the ADP algorithm is about 25% worse than the optimal solutions. However, using this algorithm we can solve much larger instances than for the full stochastic game. We also compared the ADP approach with a static approach and this showed that the ADP approach outperforms the static approach in our computational experiments.

For large instances, the convergence of the value function approximation can be slow, because states have to be visited multiple times before a good approximation can be given. We have used state space aggregation to speed up this convergence. For small instances, we do not gain a lot from this aggregation, because the algorithm without aggregation is already fast. However, for large instances, the speed of convergence is increased considerably with this aggregation.

The convergence of the ADP algorithm also depends on different input parameters which define the step size and the level of randomness. The optimal value of these parameters can vary for each instance and may also depend on the aggregation level. From our computational experiments, we can say that a large step size and a small number of randomness performs the best for our instances.

In this chapter, we have assumed that the evolution of the payoff matrices is defined by a given transition matrix and that at the beginning of each day, the payoff for that day is known. For future research, it would be interesting to investigate the case where not all payoff matrices are knows and only predictions for each day are given.

References

Buşoniu L, de Schutter B, Babuška R (2010) Approximate dynamic programming and reinforcement learning. Interactive collaborative information systems, pp 3–44

Fang F, Stone P, Tambe M (2015) When security games go green: Designing defender strategies to prevent poaching and illegal fishing. IJCAI, pp 2589–2595

George AP, Powell WB (2006) Adaptive stepsizes for recursive estimation with applications in approximate dynamic programming. Machine Learning, 65(1):167–198.

George AP, Powell WB, Kulkarni SR (2008) Value function approximation using multiple aggregation for multiattribute resource management. Journal of Machine Learning Research, 9(Oct):2079–2111

Haskell W, Kar D, Fang F, Tambe M, Cheung S, Denicola E (2014) Robust protection of fisheries with compass. Twenty-Sixth IAAI Conference

Laan CM, Barros AI, Boucherie RJ, Monsuur H (2017) Security games with probabilistic constraints on the agents strategy. International Conference on Decision and Game Theory for Security, pp 481–493

Lin X, Beling PA, Cogill R (2017) Multi-agent inverse reinforcement learning for zero-sum games. IEEE Transactions on Computational Intelligence and AI in Games (published online)

MATLAB (2016) version 9.1 (R2016b). The MathWorks Inc., Natick MA

Mes MRK, Rivera AP (2017) Approximate dynamic programming by practical examples. In: Boucherie RJ, van Dijk NM (eds) Markov Decision Processes in Practice, Springer, pp 63–101.

Owen G (1995) Game theory, 3rd edn. Academic Press

Perolat J, Scherrer B, Piot B, Pietquin O (2015) Approximate dynamic programming for two-player zero-sum Markov games, Proceedings of the 32nd International Conference on Machine Learning (ICML-15), pp 1321–1329

Powell WB (2010) Approximate dynamic programming: Algorithms. Wiley Encyclopedia of Operations Research and Management Science

Ross S (1996) Stochastic processes, 2nd edn. John Wiley & Sons, Inc