Computing threat points in irreducible ETP-ESP games

Rogier Harmelink & Reinoud Joosten

IEBIS, School of Behavioral and Management Sciences, University of Twente, POB 217, 7500 AE Enschede, The Netherlands.

Abstract

In non-cooperative stochastic games, the threat point is the lowest possible Nash equilibrium when players force each other to receive the minimal reward possible. For repeated games the determination of the threat point seems trivial, but when looking at more complex stochastic games this does not hold. Frequency-dependent (FD) games are stochastic games for which the stage payoffs depend on a FD function, a so-called Endogenous Stage Payoff (ESP) game. Another type of FD games is when the transition probabilities between states are adjusted based on a FD function, these are called Endogenous Transition Probabilities (ETP) games. Incorporating these characteristics in stochastic games results in non-trivial determination of the threat point and therefore creates the need of an efficient threat point algorithm. In this paper we develop an algorithm which computes the threat point in irreducible ETP-ESP games with two players under the limiting average reward criterion. We start by describing an algorithm for a simple repeated game for which we later incorporate more complex elements in order to end up at an ETP-ESP game. Our algorithm can at best be described as an intelligent brute force method which computes a reasonably accurate threat point in a reasonable amount of computational time. The resulting threat point can be seen as a good approximation of the exact threat point while also giving an indication of the set of Nash equilibria.

Keywords: game theory, algorithmic game theory, stochastic games, Frequency-dependent games, jointly-convergent pure-strategies, ETP-ESP

1. Introduction

When John Nash Jr. defined the concept which is now known as the Nash equilibrium [1], he probably did not know the impact it would have on science and the world we are in nowadays. In this ever increasing complex world the need for decision making based on rationality seems to increase. The Nash equilibrium proved to be an useful rational concept for analyzing sometimes 'irrational' outcomes like the Prisoner's Dilemma [2]. Even game theoretical thought experiments like Newcomb's paradox [3] can be analyzed with the help of the Nash equilibrium [4].

The power of the Nash equilibrium also applies to more complex stochastic games. When Shapley in 1953 introduced the stochastic game in order to incorporate uncertainty of the state being in [5], the concept further increased the difficulty of the analysis of Nash equilibria. However, stochastic games are static to some extent, i.e. that the occurrence of play only has an effect on the chosen transition probabilities between states. In 2003 the introduction of frequency-dependent games by Brenner and Witt (2003), later extended by Joosten et al. (2003) made stochastic games more dynamic by adjusting the stage payoffs depending on the play up until and including the current stage.

The development of the domain of Frequency-Dependent (FD) games however has not stagnated. Now, they not only incorporate Endogenous Stage Payoffs (ESP), but also the transition probabilities may alter based on the history of play. These are called Endogenous Transition Probabilities (ETP) and first explored by Joosten and Meijboom (2018). Because of these developments, new applications of FD games were developed. Examples are Small Fish Wars by Joosten (2007) and rarity value in a small fish war [10]. The big question is how to analyze these types of FD games in terms of Nash equilibria.

Important in the analysis of Nash equilibria in stochastic games is the so-called threat point. The threat point is the lowest possible Nash equilibrium that players receive if both choose to force each other to this reward. We assume that in a two-player game, player 1 plays strategy π while player 2 plays strategy σ. Both players receive a reward denoted, for player 1 as an example, as $\gamma^1(\pi, \sigma)$. We define the threat point as follows:

$$v = (v^1, v^2) \qquad \text{with:}$$
$$v^1 = \min_{\sigma} \max_{\pi} \gamma^1(\pi, \sigma)$$

$$v^2 = \min_{\pi} \max_{\sigma} \gamma^2(\pi, \sigma)$$

Determination of this threat point is trivial in repeated games, in more complex stochastic games such as ETP-ESP games the threat point is retrievable analytically. However, these methods are time intensive and prone to human error. The need for an fast and accurate solution therefore increases. Algorithms can bridge this gap in order to cover human errors while computing reasonably accurate or even exact solutions.

1.1. Typology of stochastic games

In order to clarify the domain of stochastic games in which FD games play an important role, we refer to an ordering of the types of stochastic games by Joosten and Samuel (2018). They call this 'A computation-inspired ordering among stochastic games'.

		CTP	**ETP**
	Type I (p_0)	**Type II** (x, p_0)	**Type III** $(x, p(x))$
CSP (θ_0)	AIT games	Stochastic games	CAIT-games ETP-games
ESP $(\theta(x))$	FD games AIT FD games	Stochastic FD games	CAIT-ESP games ETP-ESP games

Table 1: 'A computation-inspired ordering among stochastic games' [11].

Table 1 shows three types of stochastic games. Type I games incorporate transition probabilities that do not depend on the history of play or actions chosen by the players in different stages of the game. Switching states is therefore completely independent of player interference. Type II games are regular stochastic games in which players' actions have an impact on the transition probabilities and therefore the switching of the state of play. Type III games are the so-called ETP class of stochastic games in which the actions of player not only impact the transition probabilities as a result of play, but also alter them based on the frequency of play.

All types of stochastic games have a Constant Stage Payoffs version but also an Endogeneous Stage Payoffs version. (C)AIT stands for (Current) Action Independent Transitions. The term stochastic FD games has been introduced by Mahohoma (2014), but Joosten and Samuel (2018) call this "a tautology in the broad interpretation". Our goal in this paper is to develop

a threat point algorithm which can compute the threat point on the whole range of stochastic games defined in table 1.

1.2. Earlier work

Algorithmic game theory has seen an increase in attention since the paper of Nisan and Ronen (2001). But this was not the first instance of algorithms being used within game theory as a mathematical discipline. In stochastic games there was already research in algorithms under the discounted reward criterion [14]. An algorithm for undiscounted stochastic games was developed by Vrieze (1981) based on work of Filar and Raghavan (1979). The problem with these algorithms is that it remains unclear if they are applicable to the domain of FD games and are able to find a threat point result.

The first research in the domain of algorithms of FD games was done by Mahohoma (2014). In his Master thesis he uses a simulation approach to determine equilibria, rewards but also threat points in stochastic FD games. This algorithm seems useful if non-stationary strategies are the only way to reach the threat point, but in games in which stationary strategies are guaranteed to give threat point solutions, the method seems cumbersome in computational complexity ($O(n^4)$).

Other work in the domain of algorithm development in FD games has been done by Joosten and Samuel (2017). They developed an algorithm which computes the set of feasible rewards in Type I, II and III games with or without ESP. The algorithm computes the threat point as a part of the set of feasible rewards but is unable to state this threat point as an actual result. This paper builds further on this work by computing the threat point and improving computational complexity. Part of the work on this algorithm has been done in the form of a master thesis at the University of Twente [18].

In this paper we first have introduced the concept of different types of stochastic games and the need for a threat point algorithm. Section 2 defines the specifications surrounding the game and algorithm under scrutiny. In section 3 we start with development of the algorithm on Type I games. Section 4 continues be extending the algorithm to incorporate regular stochastic game characteristics while in section 5 we incorporate the last step of the ETP characteristics. We end in section 6 with a conclusion and discussion of the algorithm.

2. Game and algorithm specifications

In this section we set the specifications and criteria that our games but also the algorithm should adhere to in order to successfully compute a threat point result. We first introduce the letters and symbols surrounding our algorithm. After this we denote the criterion of analysis of the rewards and state the concept of jointly-convergent pure-strategy rewards. Also we state specifications of the Markov chain and FD functions for which the algorithm is guaranteed to work. We discuss the choice of the programming language used and at last, we introduce an example game which we use throughout the paper as a basis to show the application of the algorithm.

2.1. Mathematical letters and symbols used

We first introduce a small table of mathematical letters and symbols which are used throughout this paper. This should reduce the chance of ambiguity on certain interpretations of the algorithm.

Mathematical Letters/Symbols	Description
γ^k	Average reward for player k
R_t^k	Expected payoff of player k at time t
π	Strategy played by player 1
σ	Strategy played by player 2
P	Strategy matrix of player 1
Q	Strategy matrix of player 2
t	Current period in time
T	Total amount of periods of time
x^t	Relative frequency vector at time t
X	Relative frequency matrix
$\liminf$	The limit inferior result
θ_{S_k}	Payoff matrix of state k
p^{S_k}	Transition probability matrix of state k
p_i	Transition probability vector
$\limsup$	The limit superior result
$Pr_{\pi,\sigma}$	Probability under strategy pair π, σ
FD_i^t	FD function at time t with game type i
v	The threat point result
$Beta(n, \alpha, \beta)$	Beta-distribution with number of points n
E	ETP matrix

2.2. Limiting average reward criterion

Stochastic games are played for a(n) (in)finite number of periods. So the total sum of payoffs can be analyzed via different criteria. Two of these criteria are dominant in literature, the discounted reward and the limiting average reward. The first one seems to be the most frequent criterion of use. However, in this paper we focus on the limiting average reward criterion. We think that it is essential in some games to value the far future equally to the present. Also discounting could possibly result in finite horizon effects occurring.

The expected payoff of player k at time t under strategy pair (π, σ) is denoted as $R_t^k(\pi, \sigma)$. So as described by Sorin (2003) we look at the limiting average reward $(\gamma^k(\pi, \sigma))$ for player k with:

$$\gamma^k(\pi,\sigma) = \liminf_{T \to \infty} \frac{1}{T} \sum_{t=1}^{T} R_t^k(\pi,\sigma)$$

For this paper we focus on the limiting average reward as the way to analyze rewards.

2.3. Jointly-convergent pure-strategy rewards

The development of this algorithm is based for a large part on the theory of jointly-convergent pure-strategy pairs. This theory was first described by Joosten et al. (2003). A player follows a pure strategy when a player always plays a pure action in a certain stage at time t in any state with probability 1. A strategy pair is jointly-convergent for the strategy of player 1 π, player 2's strategy σ and relative frequency vector x if and only if [7]:

$$\limsup_{t \to \infty} Pr_{\pi,\sigma}[|x_i^t \quad x_i| > \epsilon] = 0 \tag{1}$$

When the relative frequency vector for a strategy pair converges to a vector of fixed numbers with probability 1 when t goes to infinity [20] we call the strategy pair jointly-convergent. With the set of jointly-convergent pure-strategy pairs we can also compute a set of rewards, as was done by Joosten and Samuel (2017). Two main theorems surround the theory of jointly-convergent pure-strategy rewards.

Theorem: "Each pair of individually-rational jointly-convergent pure-strategy rewards can be supported by an equilibrium. Moreover, each pair of jointly-convergent pure-strategy rewards giving each player strictly more than the threat-point reward, can be supported by a subgame-perfect equilibrium. [7]"

The threat point plays an important role in this theorem. Also in the second theory, the threat point has a large impact.

Theorem: "Each pair of rewards in the convex hull of all individually-rational jointly-convergent pure-strategy rewards can be supported by an equilibrium. Moreover, each pair of rewards in the convex hull of all jointly-convergent pure strategy rewards giving each player strictly more than the threat-point reward, can be supported by a subgame-perfect equilibrium [7]."

The threat point therefore defines the boundaries of the set of equilibria that are covered by jointly-convergent pure-strategy rewards. Computing the threat point therefore, not only gives the absolute minimum players can reach as an equilibrium [7], but also solves the problem of determining the complete set of equilibria.

2.4. Specifications of the Markov chain

Transition probabilities in stochastic games are modeled as a Markov chain, i.e. the probability of switching to another state only depends on the current state being in. In order for a successful approximation of the threat point our algorithm demands certain properties of the Markov chain. These are: irreducibility and aperiodicity.

Definition Irreducible Markov Chains: *"A Markov chain* $(X_0, X_1, ...)$ *with state space* $S = \{s_1, ..., s_k\}$ *and transition matrix* P *is said to be* ***irreducible*** *if for all* $s_i, s_j \in S$ *we have that* $s_i \rightarrow s_j$*. Otherwise the chain is said to be reducible [21]."*

In short, we expect to have stochastic game in which the play at any time can switch to another state in the game. Temporarily absorbing states are therefore no issue, but for stochastic games in which the states are absorbing we cannot guarantee the workings of our algorithm. This however does not mean that our algorithm will not work on the games with absorbing states, further research on this is necessary.

Definition Aperiodic Markov Chains: *"A Markov chain is said to be* ***aperiodic*** *if all its states are aperiodic. Otherwise the chain is said to be periodic [21]."*

Aperiodicity means that the Markov chain does not have a fixed period of switching between states, the transition between states is random and does not incorporate any form of certainty of switching states after a certain amount of time. On the side we are looking into ways to make our algorithm use-able on games with a periodic Markov chain. For now we assume that the algorithm only works on games in which the Markov chain is irreducible and aperiodic, resulting in an useful property for use in the algorithm:

Theorem: *"For any irreducible and aperiodic Markov chain, there exists*

at least one stationary distribution [21].”

This stationary distribution can be linked with the convergence of the relative frequency x in Equation 1. Also, Markov chains that are irreducbile and aperiodic are sometimes refered to as ergodic.

2.5. Specifications of the FD functions

In frequency-dependent games we need to define a FD function which state the impact of the frequency of play on the stage payoffs or transition probabilities. The function used can be defined in any possible way, but we are not always able to guarantee the working of our algorithm. We have tested the algorithm on polynomials and linear functions and were able to receive accurate results. However, when the function is not monotonic, the algorithm is not able to generate accurate results, only an inaccurate estimation. We encountered these problems already in a paper in which the stage payoffs were analyzed with rarity value [10]. We plan on future research in order to incorporate non monotone functions within this algorithm.

2.6. Programming language

A programming language could be seen as just a method in order to deliver a certain output. In our opinion however, the choice of well suiting programming language could have a lasting impact on usage of algorithms. Earlier versions of our algorithm have been build in Excel and MATLAB. However, we think that Python is a more suitable alternative for game theoretical programs. Game theorists are mathematicians, not programmers, in our opinion a more easy readable syntax would enhance their analyzing capabilities. The goal of Python is to provide the user with easy and understandable syntax. On the other side, Python is in itself a slow language due to its dynamic nature. This we compensate by making use of the NumPy package in Python for highly optimized computations.

2.7. An example game

When writing the algorithm we tested it on a multitude of games. For didactic purposes we show the working of the algorithm on an example game developed by Samuel (2017). This example game, in our opinion, shows well what the effect is of the frequency-dependent functions while also offering a clear test case of the algorithm. The example game is a two-player frequency-dependent commons-pool game which is essentially modelled as an ETP-ESP

game. However, elements of the ETP-ESP game can be extracted in order to reflect Type I, Type II and Type III games.

2.7.1. Payoff matrices

Our example game is modelled as a frequency-dependent two-player two-state game. In both states, players have the option to act responsible or to gain a higher short-term payoff in favor of the depletion of the common-pool resource. The first state represents a situation in which there is a larger common-pool resource available. The upper-left corner of the payoff matrix is always the responsible option and results in no depletion of the common-pool resource. The bottom-right result always takes the largest hit on the common-pool resource within a state, the other two outcomes are moderately depleting.

$$\theta_{S_1} = \begin{bmatrix} 16,16 & 14,28 \\ 28,14 & 24,24 \end{bmatrix} \qquad \theta_{S_2} = \begin{bmatrix} 4.0,4.0 & 3.5,7.0 \\ 7.0,3.5 & 6.0,6.0 \end{bmatrix}$$

2.7.2. Transition probabilities

The transition probabilities guard the switching between the two states in this game. When in a certain state, the chance of staying in that same state is always weakly dominating the chance of switching states (except for when the FD function adjusts the transition probabilities in an ETP game). The sustainable (more responsible) left-upper corner always results in both states in a higher chance of ending up in the first state in which the resource availability is richer. Acting irresponsible results in a higher chance of staying in or transitioning to the second state in which the common-pool resource is already exhausted.

$$p^{S_1} = \begin{bmatrix} 0.8,0.2 & 0.7,0.3 \\ 0.7,0.3 & 0.6,0.4 \end{bmatrix} \qquad p^{S_2} = \begin{bmatrix} 0.5,0.5 & 0.4,0.6 \\ 0.4,0.6 & 0.15,0.85 \end{bmatrix}$$

2.7.3. FD functions

In our ETP-ESP game we have at least two FD functions. The use of FD functions is not limited to one per type of functionality, FD functions can also be intertwined like in [10]. However, the example game keeps it simple. We have two ESP functions which are denoted by FD_I^t for the Type I game and by $FD_{II,III}^t$ for the Type II and III games. The ETP function is denoted by $p(x)$. In this example game with an ESP function we see a larger depletion of the stage payoffs in case of reckless actions on the common-pool resource.

In case of the ETP function the transition probabilities tend to shift towards a higher probability of ending up in the second state when players choose to act irresponsible.

$$FD_I^t = 1 \quad \frac{1}{4}(x_2^t + 2x_3^t) \quad \frac{2x_4^t}{3}$$

$$FD_{II,III}^t = 1 \quad \frac{1}{4}(x_2^t + x_3^t) \quad \frac{x_4^t}{3} \quad \frac{1}{2}(x_6^t + x_7^t) \quad \frac{2x_8^t}{3}$$

$$p(x) = p_0 \quad [x \quad E] \qquad \text{with:}$$

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ .35 & .30 & .30 & .25 & .20 & .15 & .15 & .05 \\ .35 & .30 & .30 & .25 & .20 & .15 & .15 & .05 \\ .70 & .60 & .60 & .50 & .40 & .30 & .30 & .10 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ .35 & .30 & .30 & .25 & .20 & .15 & .15 & .05 \\ .35 & .30 & .30 & .25 & .20 & .15 & .15 & .05 \\ .70 & .60 & .60 & .50 & .40 & .30 & .30 & .10 \end{bmatrix}$$

3. Developing an algorithmic basis for Type I games

We introduced the specification and limitations on the type of game that can be used on our algorithm. Now we describe the building blocks of our algorithm. This we do in a structured fashion where we describe the different functions that are incorporated into the algorithm and execute specific computations. The algorithms themselves are described in pseudocode, but mathematical elements of the algorithms are introduced with a corresponding notation.

In this section we introduce a basis for the three types of games, applied to a Type I game. All types of games consists of three main functions, drawing

a random strategy for the player who threatens, create a frequency matrix for both players and later sorting this frequency matrix based on the pure best replies from the player under threat. These three elements are then combined into an algorithm for a Type I game.

3.1. Draw random strategy

The threat point for a player is defined as $\min\max \gamma^k(\pi, \sigma)$. So in order to compute a reward γ^k we start by fixing the strategy of the player who is threatening. In case of a two player game, the strategy π (σ) for player 1 (2) is fixed. However, we apply a technique called vectorization, which means that we do not draw single strategies or compute single rewards. We compute a whole set of rewards at once. This does mainly lower the computational complexity, on the other side, it requires a larger memory for computation.

In this case we draw random strategies from a so-called random number generator, usually random number generators follow a uniform distribution. But in this case we use the beta distribution as the source for generating a random number. We do this because the beta distribution has a property that tends to draw more values near the edges of the distribution when $0 < \alpha = \beta < 1$ [22]. A value closer to 0 results in more strategies that are closer to the edges of the rewards distribution. One should keep in mind that adjusting this value has a potential change on the approximation of the threat point. When plotting rewards we tend to go for $\alpha = \beta = 0.1$, when computing the threat point we use $\alpha = \beta = 0.5$.

$$P = Beta(n, \alpha, \beta) \tag{2}$$

or

$$Q = Beta(n, \alpha, \beta) \tag{3}$$

In Equation 2 (3) we draw a strategy matrix for player 1(2) containing n number of strategies. However, we need to normalize all these strategies in order to sum to one.

$$P = \frac{P_{n,i}}{\sum_{i=1}^{k}(P_{n,i})} \tag{4}$$

or

$$Q = \frac{Q_{n,i}}{\sum_{i=1}^{k}(Q_{n,i})} \tag{5}$$

Equation 4 (5) therefore normalizes the drawn strategies per strategy for player 1(2) so that they sum up to 1. Combining these equations we end up with a function which draws a random strategy matrix for the threatening player which is normalized. We describe this in Algorithm 1.

Algorithm 1 Function: Draw Random Strategy.

Input: Total points to generate, number of total actions for player who threatens

Output: Random strategy matrix

1: Draw *points* number of strategies with a length of *number of total actions* from a β-distribution with ($\alpha = \beta = 0.5$)
2: Normalize the drawn strategies, such that each individual strategy sums to one
3: Return the random strategy matrix

3.2. Create frequency vector based on the best replies

Now having generated a strategy matrix for the player who is threatening we have to make a match with the player who is being threatened. Therefore, we transform the strategy matrix into a frequency matrix. We base the creation of this frequency matrix on a result by Hordijk et al. (1983). They state that mixed stationary strategies are always best replied with a pure stationary strategy. Because we have guaranteed stationary strategies as optimal due to an ergodic Markov chain guarding the transition probabilities, the result of Hordijk et al. (1983) should hold. So a player being threatened in a two-player two state game with two actions per state always has four pure strategies to use in order to get a best response.

The result is Algorithm 2 which does the simple job of bookkeeping which strategies of the threatening player match to the pure strategies of the player under threat.

Algorithm 2 Function: Create Frequency Matrix.

Input: Total points generated, number of total actions for both players, random strategy matrix

Output: Frequency pairs based on pure best replies from player under threat

1: Initialize frequency vector with dimensions: (total number of points total number of actions Player 1 , total number of actions Player 1 total number of actions Player 2)
2: **for** i in range *total number of actions player under threat* **do**
3: **for** j in range *total number of actions threatening player* **do**
4: **if** v^1 is searched **then** frequency vector[total number of points $(i \quad 1)$:total number of points i,(number of actions player 2 $\quad i)+j$] = random strategy matrix[:,j]
5: **end if**
6: **if** v^2 is searched **then** frequency vector[total number of points $(i \quad 1)$:total number of points i,(number of actions player 1 $\quad j)+i$] = random strategy matrix[:,j]
7: **end if**
8: **end for**
9: **end for**
10: Return frequency matrix

3.3. Sort the payoffs for threat point selection

In order to complete the algorithm for (non)-FD Type I games we need a function which re-sorts the payoffs based on the frequency matrix generated. As an example, if the player threatening has a certain mixed strategy, the player who is being threatened has (in case of our example game) four best pure stationary replies. In the end, for threat point determination the player under threat always wants to play the pure stationary strategy which maximizes his reward. On the other side, the threatening player will always want to choose the mixed stationary strategies which gives the opponent the minimal reward possible. We therefore need a function which matches all pure stationary best replies to a threatening strategy. This is exactly which Algorithm 3 describes.

Algorithm 3 Function: Reward sort.

Input: Total points generated, rewards matrix, total number of actions
Output: Sorted rewards matrix

```
Initialize empty sorted reward matrix (number of points, number of actions)
for x in range total number of points do
    for i in range total number of actions do
        Sorted reward matrix[x,i] = reward matrix[points   i + x]
    end for
end for
Return sorted reward matrix
```

3.4. Combined into Type I algorithm

Now we have three basic functions which can be used to create a Type I algorithm for threat point determination. Algorithm 4 describes the steps that the algorithm takes, starting from line 5. We flatten the reward matrix into a reward vector, draw random strategies with help of Algorithm 1. Then we create a frequency matrix with Algorithm 2. If we have an ESP Type I game we activate the FD-function and compute the rewards corresponding to the frequency matrix. If not, we just compute the rewards. These rewards are then sorted for determination of the threat point, which in the end is done by first choosing a maximum reward for the player under threat given each mixed stationary threatening strategy. Then, the minimum of all generated

threatening strategies is chosen. If these steps are followed, the algorithm should result in the threat point of the two-player (non)-FD Type I game.

Algorithm 4 (Non)-FD Type I threat point algorithm.

Input: Type I Game, total number of points, activate FD function
Output: Threat point

If v^1 is calculated, then $A = 1$, $B = 2$,
If v^2 is calculated, then $A = 2$, $B = 1$

Turn reward matrix Player A into flattened reward vector
Threatening strategy matrix Player B = Draw Random Strategies
Best response Player A frequency matrix = Create Frequency Matrix
if Activate FD function = True **then**
    Activate and Calculate FD reward function result
end if
Payoffs Player A = Sum over all columns of: (Frequency matrix per row flattened reward vector Player A)
if FD Function is active **then**
    Element wise multiplication of FD reward function result with rewards Player A
end if
Sorted reward matrix = Reward sort
Pick the maximum value of each row of the sorted reward matrix as best response of Player X
Pick the minimum value over all rows as the result of v^A

Return threat point $v = (v^1, v^2)$

3.5. Results on the example game and testing

In order to validate the algorithm and to test the workings in terms of accuracy and speed we have run a few tests. We can safely say that we found a good approximation of, as an example the result of v^1, as a part of the threat point v if and only if:

$$\max_{\pi} \min_{\sigma} \qquad \min_{\sigma} \max_{\pi}$$

The $\min_\sigma \max_\pi$ result should be the approximation of v^1. If we have found $\max_\pi \min_\sigma$ to be equal to $\min_\sigma \max_\pi$, we have an exact result of v^1. If there is a difference but the inequality is adhered, we say that we have found an approximation.

Tests show that the accuracy of our algorithm corresponds to the number of mixed strategies that are generated for the threatening player. The higher the amount of mixed strategies, the more accurate the results seem to get. With our example game we are able to produce, by generating $100,000$ mixed threatening strategies an accuracy of 9 decimals for the non-FD version. On the FD version we find an accuracy of 7 decimals [18].

Visually we receive the following figures in our example game. We use an adjusted version of the algorithm of Joosten and Samuel (2017) for the computation of the total set of rewards.

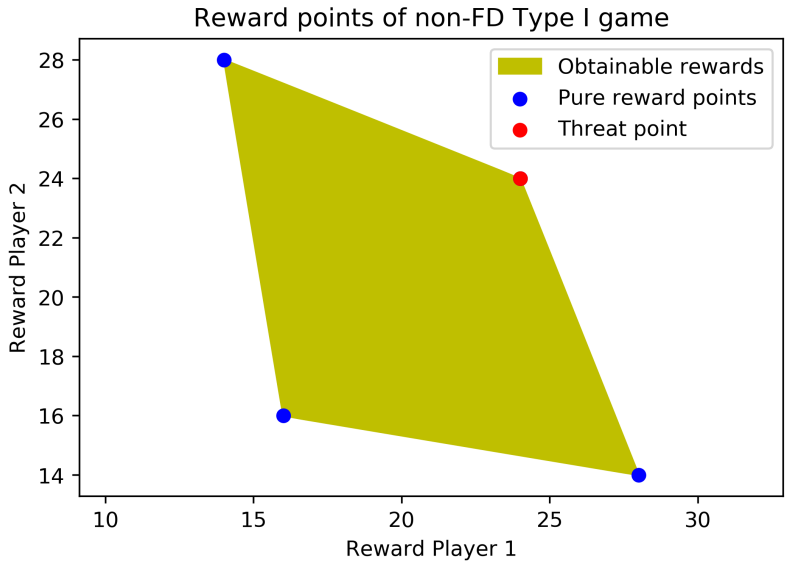


Figure 1: Rewards including threat point of the Type I non-FD example game [18].

As is visible in Figure 1, we see that the threat point is the only Nash equilibrium possible. Therefore, if players decide to play individually rational, they can only end up at the threat point. However, if we add the ESP to the game, we end up with a FD Type I game.

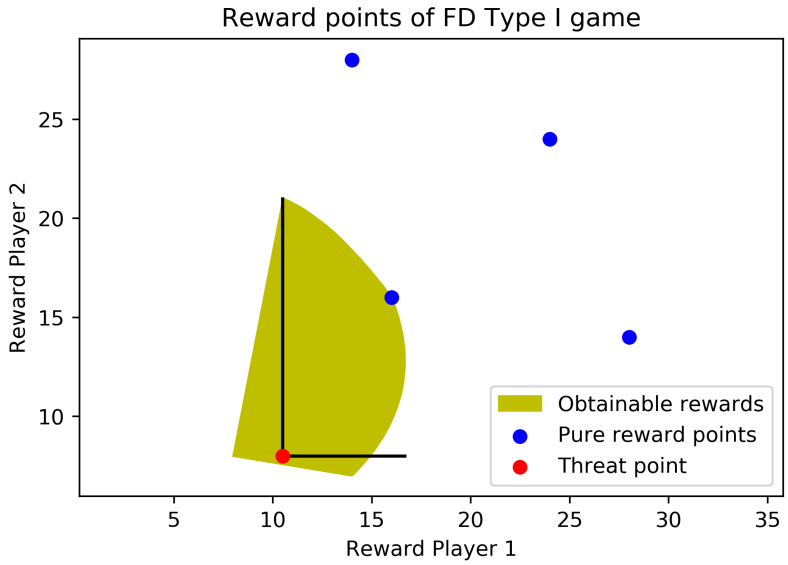


Figure 2: Rewards including threat point of the Type I FD example game [18].

Figure 2 clearly shows that the set of obtainable rewards shifted to the lower left part of the pure reward points. Also, in this case the threat point denotes a bigger set of Nash equilibria that are attainable as described by Folk Theorem. The FD function therefore has a noticeable impact on the set of obtainable individual-rational rewards and the threat point.

In terms of speed and accuracy, we have compared our algorithm in case of non-FD Type I games with a linear programming approach [18]. We tested both our algorithm and the linear programming approach on accuracy. For larger bi-matrix games, our algorithm was diverging in terms of accuracy more than the linear programming approach. The linear programming approach however, was also not able to find an exact threat point for really large (10 x 10) bi-matrix games. In computational speed, the linear programming algorithm finds an ϵ-accurate solution within $\frac{1}{100}$th of the time that our algorithm takes [18]. However, our algorithm is able to cope with FD functions. Currently, we are also looking into the linear programming approach on FD games in order to improve the speed and accuracy, but have not been able to get this working on the broad range of FD-games.

4. Extending the algorithm to Type II games

Our algorithm works on Type I games, but Type II games are a different ball game. Type II games are stochastic games in which players have an impact on the transition probabilities. A stochastic game could be seen as a generalization of a Markov Decision Process in which one player has a fixed strategy while the other player controls the transition probabilities [24].

The hardest part in computing rewards in stochastic games is the frequency of ending up in a certain state. However, in 2.4. we stated specifications to which the Markov chain guarding the transition probabilities should adhere. If the transition probabilities are ergodic, then there always exists a stationary distribution. Based on this stationary distribution, we could compute the frequency of which play ends up in a certain state.

4.1. Balance equation

This is where the balance equation comes into play. The balance equation computes the stationary distribution to which the frequency vector x will converge based on the transition probabilities from vector p. The balance equation for a two-player two state stochastic game with two actions per state is:

$$\sum_{i=1}^{4} x_i(1 \quad p_i) = \sum_{i=5}^{8} x_i p_i \tag{6}$$

If and only if:

$$\sum_{i=1}^{4} x_i(1 \quad p_i) \neq 0 \tag{7}$$

$$\sum_{i=5}^{8} x_i p_i \neq 0 \tag{8}$$

Equations 7 and 8 state that the balance equation can not end up with absorbing states. We calculate the stationary distribution of the Markov chain in a similar way done by Samuel (2017) with the introduction of intermediate vector y and variable Q. In which Q represents the frequency of play in State 1, and therefore $(1 \quad Q)$ being the frequency of play in State 2.

$$y_i^{S_1} = \frac{x_i^{S_1}}{\sum_{j=1}^{4} x_j^{S_1}} \tag{9}$$

$$y_i^{S_2} = \frac{x_i^{S_2}}{\sum_{j=5}^{8} x_j^{S_2}} \tag{10}$$

Which provide a basis for calculating Q.

$$Q = \frac{\sum_{i=5}^{8} y_i^{S_2} p_i}{\sum_{i=1}^{4} y_i^{S_1}(1 \quad p_i) + \sum_{i=5}^{8} y_i^{S_2} p_i} \tag{11}$$

We incorporate the calculations of y and Q into a function described in Algorithm 5.

Algorithm 5 Balance Equation Function.

Input: Frequency matrix X, transition probabilities p
Output: Frequency matrix X adjusted to stationary distribution

1: Initialize Q and y
2: Calculate y
3: Calculate Q with y and p
4: Calculate new frequency matrix X based on Q and y
5: Return new frequency matrix X adjusted to the stationary distribution

In case of Type II games it is only necessary to compute the balance equation just once, because after just one iteration of the balance equation the frequency matrix already converges to a stationary distribution.

4.2. Combined into Type II algorithm

Computing the threat point in Type II games is now possible. To do this, we need a slight alteration of the Type I algorithm. We add in the balance equation function after we have sorted the frequency matrix based on the best response from the player under threat. After this, computation of the threat point is possible, for both the ESP and CSP version of the Type II game. We describe this in Algorithm 6.

Algorithm 6 (Non)-FD Type II threat point algorithm.

Input: Type II Game, total number of points, activate FD reward function
Output: Threat point

1:
2: If v^1 is calculated, then $A = 1$, $B = 2$,
3: If v^2 is calculated, then $A = 2$, $B = 1$
4:
5: Turn reward matrix Player A into flattened reward vector
6: Threatening strategies Player B = Draw Random Strategies
7: Best response Player A frequency matrix = Create Frequency Matrix
8: Calculate adjusted frequency matrix by computing the balance equation
9: **if** Activate FD reward function = True **then**
10: Activate and Calculate FD Function result
11: **end if**
12: Payoffs Player A = Sum over all columns of: (Frequency Matrix per row flattened reward vector Player A)
13: **if** FD Function is active **then**
14: Element wise multiplication of FD function result with reward Player A
15: **end if**
16: Sorted reward matrix = Reward sort
17: Pick the maximum value of each row of the sorted reward matrix as best response of Player A
18: Pick the minimum value over all rows as the result of v^A
19:
20: Return threat point $v = (v^1, v^2)$

4.3. Results on the example game

Again, we run our algorithm on the example game described in 2.7. Also, we check our algorithm with a Relative Value Iteration from algorithm from [18] and also determine the lower boundary with a maximin version of the algorithm. First we look at the non-FD Type II example game with our algorithm.

Figure 3 shows that the set of obtainable rewards is in the middle of the pure reward points of both states. In this case, the threat point is in the higher regions of the obtainable rewards. In comparison to Figure 1 we already see that there are more Nash equilibria obtainable in this case, this

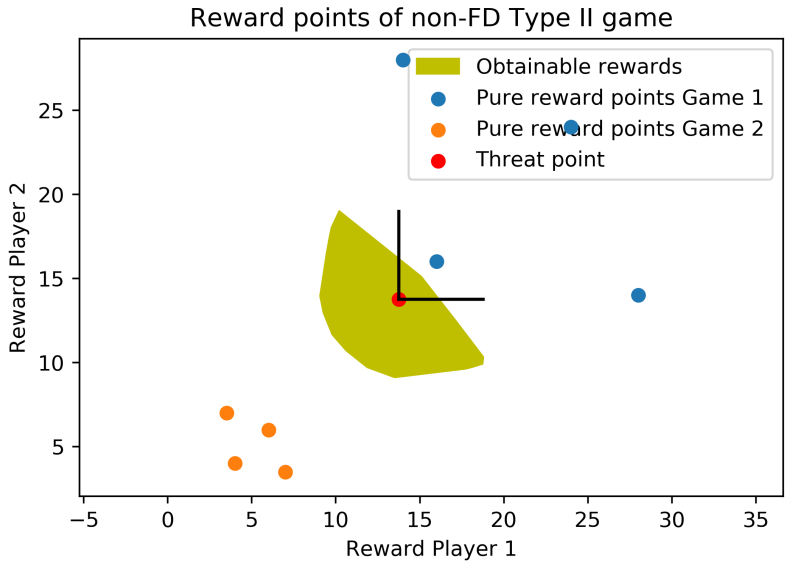


Figure 3: Rewards including threat point of the Type II non-FD example game [18].

is largely due to the game now being of a stochastic nature.

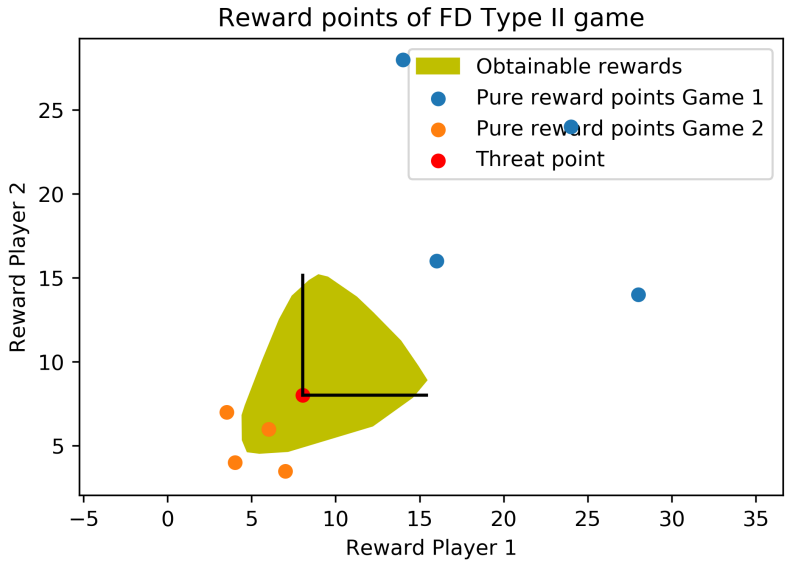


Figure 4: Rewards including threat point of the Type II FD example game [18].

In Figure 4 we see that the impact of the ESP function is larger. The set of obtainable rewards shifts to the pure reward points from the second state, in which the common-pool resource is more depleted than in case of the first state. The threat point also shifts to the bottom part of the set of obtainable rewards, increasing the absolute set of Nash equilibria as a whole.

We again tested the algorithm in terms of speed and accuracy. The Relative Value Algorithm from [18] works on non-FD Type II games but not on FD Type II games. We test this on the example game, but also on a 10 x 10 bi-matrix stochastic game. The nature of the Relative Value Algorithm lies in the Bellman equation and dynamic programming. Therefore, the algorithm is able to find good approximations of the threat point. However, our results on the example game show that when generating $10,000$ points, the Relative Value Iteration algorithm is able to generate a 4 decimal accurate result but taking 110 times longer to generate the result [18]. Our algorithm computes a 2 decimal accurate result but in a much faster time, but on the other side uses more memory than the Relative Value Iteration algorithm. On larger stochastic games we see that for a large amount of points our algorithm can run into memory errors. The Relative Value Iteration algorithm is able to generate a threat point result, but was not close to a best approximation of the maximin result [18].

5. Incorporating the ETP from Type III games

The last, but also the most complex hurdle to take is incorporating Endogenous Transition Probabilities into the algorithm. For Type II games we only had to compute the balance equation once, but for Type III games "determining a stationary distribution is like shooting at a moving target [11]".

5.1. Balance equations for an ETP function

In order to try to 'hit' the stationary distribution we adjust the balance equation function created in Section 4 in order to cope with an ETP function. We do this in two ways, with and without Aitken's Δ^2 accelerator. Both have their positive and negative aspects, which we will address. Starting with the function without Aitken's Δ^2.

5.1.1. Without Aitken's

In this version we compute the stationary distribution of the Type III game without accelerator. The main purpose is to look at convergence of

a row in the frequency matrix towards a stationary distribution. If the row in the matrix settles to a stationary distribution, it is removed from the computations. Rows that have not converged will be computed again.

Algorithm 7 Balance Equation ETP function.

Input: Frequency matrix X, transition probabilities p, ETP matrix E
Output: Frequency matrix X adjusted to stationary distribution

Initialize Q and y
Calculate y

while Not all rows in frequency matrix X have converged **do**
    Calculate new $p(x)$ with $X \quad E$
    Calculate Q with y and $p(x)$
    Calculate new frequency matrix X based on Q and y
    Check whether Q has converged based on earlier result

    if Q has converged **then** Remove row from frequency matrix X from calculating a new Q
    end if
end while
Return new frequency matrix X adjusted to the stationary distribution

However, our calculations show that some of the targets keep moving, even after shooting for a large amount of times by computing the balance equation. Figure 5 shows us that after roughly 50 60 iterations most Q's have entered a stationary state. Which is also what Joosten and Samuel (2018) have concluded. Unfortunately, still a large proportion of the Q's are unable to reach a stationary state. Even for larger amount of iterations (> 100) convergence of Q is never guaranteed. The underlying reasons are opaque to us, it could be due to the nature of the ETP function that the frequency vector is not able to reach a stationary distribution. Another possibility is that the frequency vector has practically converged to a stationary distribution, but due to floating-point arithmetic the computer is stating that convergence is not exact.

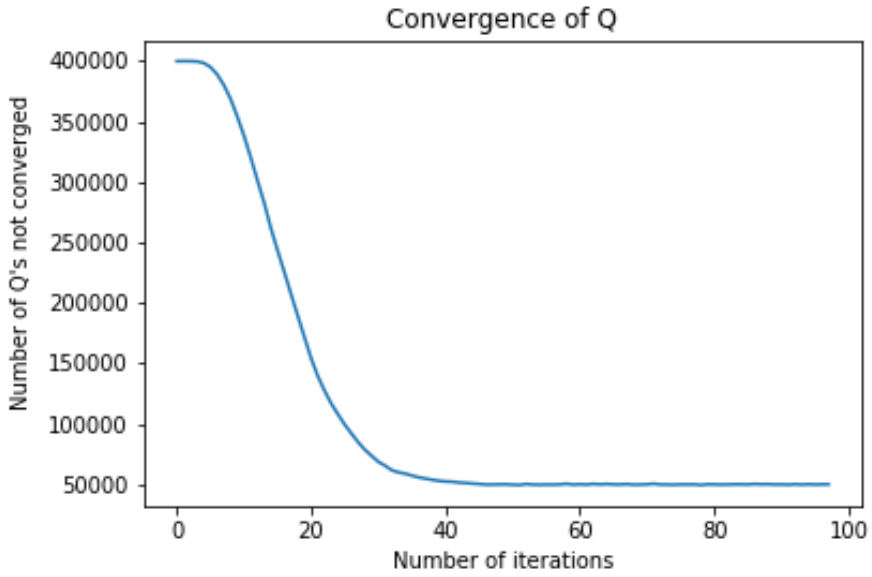


Figure 5: Convergence of Q [18].

5.1.2. With Aitken's and the potential risks

The other version of computing the balance equation with an ETP function uses Aitken's Δ^2. Joosten and Samuel (2018) introduced this accelerator in order to further speed up computations regarding the balance equation. As long as the sequence in it self is linearly convergent, the sequence will converge to a certain value with Aitken's Δ^2 [25]. If the sequence is specified with $\mathbf{f}p_n\mathbf{g}_{n=0}^{1}$, then we define Aitken's Δ^2 as:

$$\hat{p}_n = p_n \quad \frac{(p_{n+1} \quad p_n)^2}{p_{n+2} \quad 2p_{n+1} + p_n} \tag{12}$$

Equation 12 is not always numerically stable. Numerical stability can however be improved by using another version of Aitken's Δ^2:

$$\hat{p}_n = p_{n+2} \quad \frac{(p_{n+2} \quad p_{n+1})^2}{(p_{n+2} \quad p_{n+1}) \quad (p_{n+1} + p_n)} \tag{13}$$

Joosten and Samuel (2018) have successfully implemented this method, which will decrease the number of iterations necessary for convergences to 25 30. We also adopt this method into our balance equation function, which is described in Algorithm 8.

However, incorporating Aitken's Δ^2 in the computation of the balance equation is not without risk. When applying the accelerator, we encountered two issues with the method. First of all, when applying the accelerator after three iterations, we saw a slight numerical instability. Therefore the threat point result could be sometimes slightly lower than the maximin result. Trial and error has resulted in using Aitken's Δ^2 after ten iterations of the balance equation. We can see in Figure 6 that convergence after ten iterations rapidly increases.

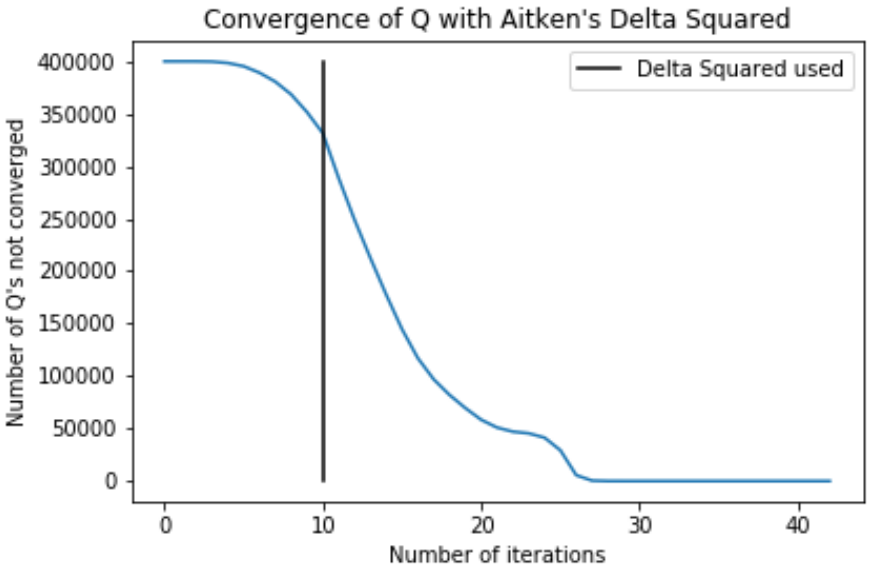
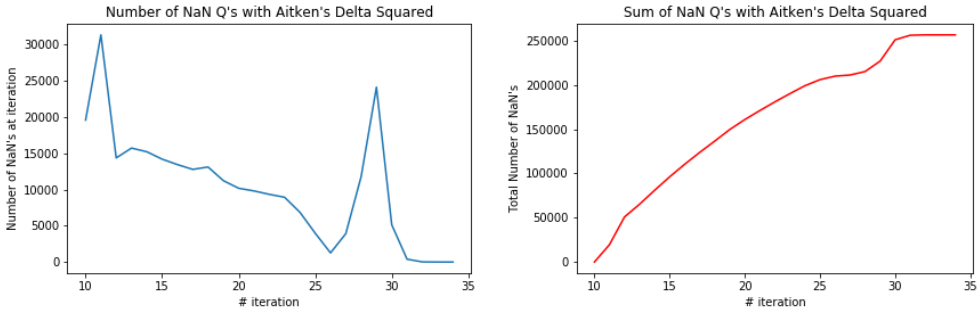


Figure 6: Convergence of Q with Aitken's ^{2} [18].

The second risk surrounding applying Aitken's Δ^2 is due to the rapid convergence of the frequency vector to a stationary distribution. A computer represents their numbers with floating-points, but the risk with Aitken's Δ^2 is directly related to these floating-points. When applying the accelerator, the computer is sometimes unable to compute a more accurate version of the sequence and therefore returns a Not a Number (NaN).



(a) Iterations of Q with NaN (b) Sum of Q with NaN

Figure 7: Occurence of NaN's during iterations of Q with Aitken's ^{2} [18].

Figure 7 shows the number of NaN's occuring when applying the accelerator after 10 iterations. We have run this for $400,000$ Q's to be generated. 62.5% of the Q's have been reported to converge to a NaN by the computer. We expect that the value probably has converged due to the rapid nature of Aitken's Δ^2. Therefore we have constructed a safeguard which restates the last known value for the NaN result and assumes that the frequency vector has converged. These are then combined in to Algorithm 8.

Algorithm 8 Balance Equation ETP Aitken's function.

Input: Frequency matrix X, transition probabilities p, ETP matrix E
Output: Frequency matrix X adjusted to stationary distribution

Initialize Q and y
Calculate y

for ten iterations **do**
    Calculate new $p(x)$ with $X \quad E$
    Calculate Q with y and $p(x)$
    Calculate new frequency matrix X based on Q and y

    Return Q
end for

while Not all rows in frequency matrix X have converged **do**
    Calculate new Q with Aitken Δ^2
    Compare new Q with old Q for convergence
    if Q has converged **then** Remove frequency matrix X from calculating a new Q
    end if
end while
Return new frequency matrix X adjusted to the stationary distribution

5.2. Combined into Type III algorithm

We opted to use the version of the balance equation which uses Aitken's Δ^2 with the safeguarding mechanism. However, this has not fully solved the problem. When using the results of Q for computation of the rewards, the computer still reported some NaN's after multiplication with Q. Therefore we remove all NaN results from the row of rewards containing the NaN reward. In case of our example game containing two states in which each state is a 2x2 bi-matrix game we have to remove 4 rewards from the row containing the NaN reward.

Algorithm 9 (Non)-FD Type III threat point algorithm.

Input: Type III Game, total number of points, activate FD reward function
Output: Threat point

1:
2: If v^1 is calculated, then $A = 1$, $B = 2$,
3: If v^2 is calculated, then $A = 2$, $B = 1$
4:
5: Turn reward matrix Player A into flattened reward vector
6: Threatening strategies Player B = Draw Random Strategies
7: Best response Player A frequency matrix = Create Frequency Matrix
8: Calculate Q based on Algorithm 8 with Aitken's Δ^2
9: Calculate adjusted frequency matrix with help of Q
10: **if** Activate FD reward function = True **then**
11: Activate and Calculate FD Function result
12: **end if**
13: Rewards Player A = Sum over all columns of: (Frequency Matrix per row flattened vector Player A)
14: **if** FD reward function is active **then**
15: Element wise multiplication of FD reward function result with Rewards Player A
16: **end if**
17: Sorted reward matrix = Payoff sort
18: Pick the maximum value of each row of the sorted reward matrix as best response of Player A
19: Pick the minimum value over all rows as the result of v^A
20:
21: Return threat point $v = (v^1, v^2)$

As can been seen in Algorithm 9 the basis for the Type III game is largely Algorithm 6 from the Type II game, but now adjusted with the balance equation for Type III games. All algorithms described in this paper can be found in an online Github environment[1]. We encourage readers to check the workings of our algorithm and to contribute to further research.

[1]Environment link: https://github.com/Rogierr/GTToolbox. We further develop the toolbox as a part of more game theory research.

5.3. Results on the example game

At last we check the workings of our algorithm on the example game. For the ETP game we have been unable to create a second algorithm in order to compare our algorithm with. Therefore the only thing we can check is the accuracy of the algorithm based on the example game.

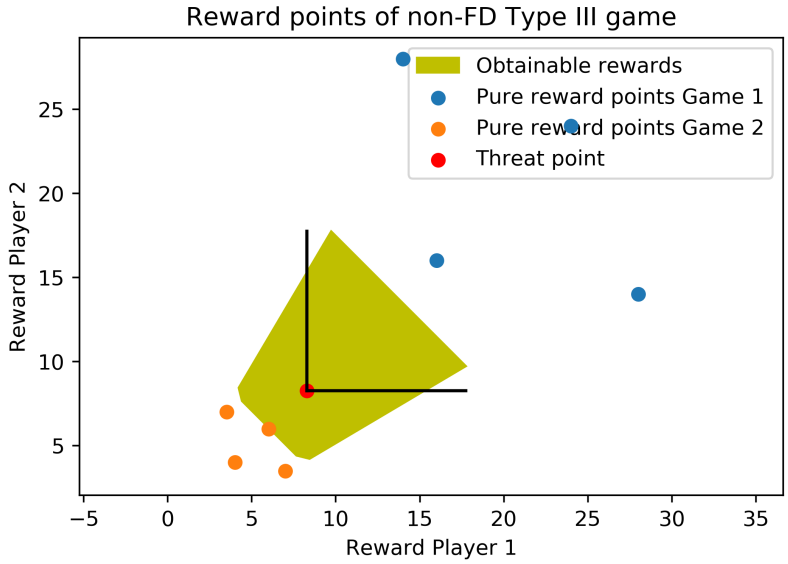


Figure 8: Rewards including threat point of the Type III non-FD example game [18].

Figure 8 shows the rewards including threat point on the Type III game without ESP function. We see that the set of rewards is more shifted towards the pure rewards of the second state, probably due to the ETP function having an impact on ending up more regularly in state two.

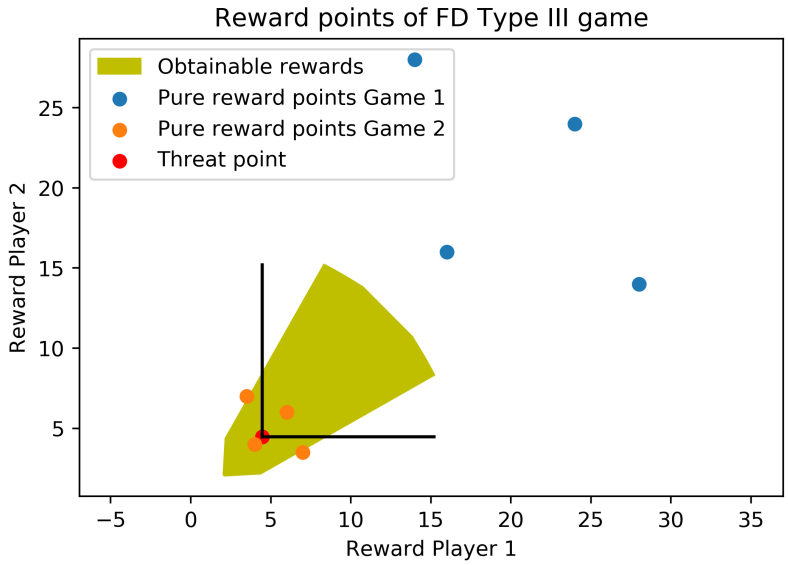


Figure 9: Rewards including threat point of the Type III FD example game [18].

Figure 9 shows the effect of the ESP function on the set of rewards but also on the threat point. Again, the threat point shifts to the lower end of the rewards, as also was the case at the other types of games with ESP function. In terms of speed and accuracy we have cannot compare our algorithm with another algorithm, but in case of the non-FD Type III game we find a three decimal accurate solution within 39.03 seconds. For the FD Type III game we find a two decimal accurate solution in 35.51 seconds[2] [18]. Therefore we find it plausible that we have constructed a brute-force algorithm which finds a reasonably accurate solution in a reasonable amount of computational time.

[2]Computational time is just an indication of the speed of the algorithm. The computer of choice has a large impact on the real-life speed of the algorithm

6. Conclusion and discussion

In this paper we developed an algorithm which can cope with the computation of the threat point in a two-player two-state irreducible ETP-ESP game. We gradually built the algorithm by starting with a Type I game, defining the three basic functions which are then used for all types of games. The balance equation is used when the transition probabilities come into play. Largely, our algorithm is an adaptation of the work of Joosten and Samuel (2018). However, our algorithm approximates the threat point while also having significant gains in reduction of computation time due to vectorization.

We compared the algorithm to other algorithms built for Type I and Type II games without ESP. Our algorithm is in comparison a good trade-off in terms of speed and accuracy when focusing on a two-player two-state game in which each state has two actions. We also have shown that the use of Aitken's Δ^2 accelerator can result in large improvements in terms of speed but is not completely without risk. However, there are still multiple enhancements or alternatives to our algorithm which could result in an even larger improvement in either speed or accuracy[3].

A possible route for a future enhancement could be to adapt a global search local search strategy. First the algorithm should find a global approximation and later look for a more accurate solution on a local level. Another possibility is to further dive into solving the problem as a linear programming problem. We have made a small step in completing this approach with the Sequentially Least Squares Programming algorithm, but have been unable to get it working on Type II and Type III games. Results on Type I games are however very promising. The most out-of-the-box option is to look at computation of the threat point from a reinforcement learning perspective.

Theoretically our biggest pitfall currently is the algorithm not working on FD-games in which the utility functions are non-monotonic. That does not only require an algorithmic perspective, but a more rigorous mathematical approach to solve.

[3]We invite interested researchers to join us in further developing our algorithms in the GTToolbox: https://github.com/Rogierr/GTToolbox

References

[1] J. Nash, Non-cooperative games, Annals of mathematics 54 (1951) 286–295.

[2] W. Poundstone, Prisoner's Dilemma/John von Neumann, Game Theory and the Puzzle of the Bomb, Anchor, 1993.

[3] R. Nozick, Newcombs problem and two principles of choice, in: Essays in honor of Carl G. Hempel, Springer, Dordrecht, 1969, pp. 114–146.

[4] T. A. Weber, A robust resolution of Newcombs paradox, Theory and Decision 81 (2016) 339–356.

[5] L. S. Shapley, Stochastic games, Proceedings of the national academy of sciences 39 (1953) 1095–1100.

[6] T. Brenner, U. Witt, Melioration learning in games with constant and frequency-dependent pay-offs, Journal of Economic Behavior & Organization 50 (2003) 429–448.

[7] R. Joosten, T. Brenner, U. Witt, Games with frequency-dependent stage payoffs, International journal of game theory 31 (2003) 609–620.

[8] R. Joosten, R. Meijboom, Stochastic games with endogenous transitions, in: S. Neogy, R. B. Bapat, D. Dubey (Eds.), Mathematical Programming and Game Theory, Springer, 2018, pp. 205–226.

[9] R. Joosten, Small Fish Wars: a new class of dynamic fishery-management games, The IUP Journal of Managerial Economics (2007) 17–30.

[10] R. Joosten, R. Harmelink, Strong rarity value in view of hysteresis in a stochastic fishery game, 2019.

[11] R. Joosten, L. Samuel, On finding large sets of rewards in two-player ETP-ESP-games, 2018.

[12] W. Mahohoma, Stochastic games with frequency dependent stage payoffs (MSc thesis), 2014.

[13] N. Nisan, A. Ronen, Algorithmic mechanism design, Games and Economic behavior 35 (2001) 166–196.

[14] T. E. S. Raghavan, J. A. Filar, Algorithms for stochastic games - A survey, Zeitschrift für Operations Research 35 (1991) 437–472.

[15] O. J. Vrieze, Linear programming and undiscounted stochastic games in which one player controls transitions, Operations-Research-Spektrum 3 (1981) 29–35.

[16] J. A. Filar, T. E. S. Raghavan, An algorithm for solving an undiscounted stochastic game in which one player controls transitions, Research Memorandum, University of Illinois, Chicago (1979).

[17] R. Joosten, L. Samuel, On the computation of large sets of rewards in ETP-ESP-games with communicating states, 2017.

[18] R. Harmelink, Computing threat points in two-player ETP-ESP games (MSc Thesis), 2019.

[19] S. Sorin, Classification and basic tools, in: Stochastic Games and Applications, Springer, 2003, pp. 27–36.

[20] P. Billingsley, Probability and measure, John Wiley & Sons, 2008.

[21] O. Häggström, Finite Markov chains and algorithmic applications, volume 52, Cambridge University Press, Cambridge, 2002.

[22] L. Samuel, Computations in Stochastic Game Theory (MSc Thesis), 2017.

[23] A. Hordijk, O. J. Vrieze, G. L. Wanrooij, Semi-Markov strategies in stochastic games, International Journal of Game Theory 12 (1983) 81–89.

[24] J. Filar, K. Vrieze, Competitive Markov decision processes, Springer Science & Business Media, 2012.

[25] R. L. Burden, J. D. Faires, Numerical Analysis, Brooks/Cole, ninth edition, 2011.