Imperial College London

COURSEWORK

IMPERIAL COLLEGE LONDON

DEPARTMENT OF MATHEMATICS

Dynamics of Games Coursework: Colonel Blotto Games and an application to US Elections

Author:
Matteo Mario Di Venti (CID: 01496583)

Date: January 22, 2021

1 Introduction

1.1 The Captain Blotto game.

A 'Colonel Blotto' game is a type of two person game in which the two players (usually called Blotto and Enemy) must distribute a finite predetermined quantity of resources over a finite number of objects denoted as 'battlefields'. Both players allocate the resources simultaneously and they know reciprocally the initial resource endowment. The division of the initial endowment can be continuous or discrete according to the game variant.
When the resources are allocated to each battlefield, they are confronted and the player that allocated most resources 'wins' the battlefield. The payoff for each player is the number of battlefields won. The winner of the game is whoever has won the most battlefields, with a draw if all players have the same number of conquered battlefields.
The game was initially developed by Borel in 1921 and has received extensive research during the last 100 years thanks to its direct applications in election funds allocation, warfare tactics and auctions. In particular, significant improvement to the problem of determining Nash Equilibria of the game was reached thanks to Regret-matching algorithm by Hart and Mas-Colell [2000].

1.2 Actions and Matrix Form

Consider two cases: the discrete and the continuous allocation.
In the discrete case we have an integer numbers of resources B and E and an integer number of battlefields "N" with constraint $B, E > N$.
It is straight forward to see that the number of possible distributions of resources across battlefields is the classic stars and bars combinatorics problem. For this reason we have $A_B = \binom{B+S-1}{B}$ strategies for Colonel Blotto (B) and $A_E = \binom{E+S-1}{E}$ for Enemy (E).
The payoff matrix is therefore $A_B \times A_E$ dimensional with entries (1,-1) for Blotto's victory, (-1,1) for Enemy victory and (0,0) for a draw.

In the continuous case, the possible actions are uncountably infinite. We represent the set of all possible allocations as the two sets $C_B = \left\{ \begin{bmatrix} b_1 \\ ... \\ b_n \end{bmatrix} \in \mathbb{R}^n_{\geq 0} \text{ s.t. } \sum_{i=1}^n b_i = B \right\}$ and $C_E = \left\{ \begin{bmatrix} d_1 \\ ... \\ d_n \end{bmatrix} \in \mathbb{R}^n_{\geq 0} \text{ s.t. } \sum_{i=1}^n d_i = E \right\}$.
The payoff is given as a function on $C_B \times C_E$ that maps to the set $\{1, 0, -1\}$.
We can define the payoff function for B as payoff$(\boldsymbol{b}, \boldsymbol{e}) = sign(\sum_{i=1}^n sign(b_i - e_i))$ While payoff function for E is the negative version of the same function above as we are in a zero sum game setting.

1.3 Existence of Nash Equilibria.

Again let's consider the two cases: the discrete allocation and the continuous one. For the discrete allocation, the existence of at least on Nash Equilibrium derives directly from the proof of existence by John Nash.

Theorem 1
Every discrete Blotto game has at least one Nash Equilibrium.

Proof 1
To prove the statement we will use the simplification introduced by Nash [1950] that makes use of the Kakutani Fixed Point theorem. But first let's define the best response r_i of player i to the strategy of the other player σ_{-i} as $r_i(\sigma_{-i}) = arg_{\sigma_i} maxA_{\sigma_i,\sigma-i}$ where $\sigma_i \in \Delta = \{x \in \mathbb{R}^n_{\geq 0}; \quad \sum_{i=1}^n x_i\}$, the n-dimensional simplex. Now define set valued function $r : \Delta \to 2^\Delta$ such that $r = r(\sigma_{-i}) \times r_{-i}(\sigma_i)$ and the problem of existence of a Nash Equilibrium becomes equivalent to r having a fixed point.
In this case we can apply Kakutani's fixed point theorem if four condition are met:

- *Δ is convex, compact and nonempty. This is satisfied by being the n-dimensional simplex*
- *$r(\sigma)$ is nonempty and upper hemicontinous (i.e. $\forall a$ for any neighbourhood V of $r(a) \quad \exists$ neighbourhood U of x s.t. $\forall x \in U, r(x) \subset V$). This condition is satisfied by Berge's maximum theorem which provides partial continuity for the optimizations of a function.*
- *$r(\sigma)$ is convex. This is guaranteed by the existence of mixed strategies, since the linear combination of any two strategies maximising payoff will yeld the same maximum payoff.*

Therefore r has a fixed point and a Nash Equilibrium

Considering the continuous case we can not directly apply Nash Theorem as the existence of NE for games with infinite choices is not guaranteed. However, if the choices are compact and each player's payoff continuous in the strategy of the players then a Nash Equilibrium is guaranteed to exist (Ozdaglar [2013]). Since the Continuous Colonel Blotto setting satisfies these requirements, a Nash Equilibrium always exists.

2 Nash Equilibria of simple Blotto

Although the existence of at least a Nash Equilibrium is guaranteed, calculating Nash Equilibria for each Colonel Blotto games is not straightforward, as can be seen from the following simple context. Let us define a simple Colonel Blotto game in which the number of battlefields is two (N=2), the two players have unequal resources, the battlefields might have unequal value and the allocations do not need to be integer numbers. Without loss of generality we will consider Blotto to be the player with more resources and in case of draw Blotto will win the game.
Given this specification, the solution to the problem of finding NE is given by Macdonell and Mastronardi [2015] in a recent paper. In the paper they prove the statement rigorously but also present a particularly clever graphical solution, both will be described in the following sections.

2.1 Simple wars

The game is defined as follows:

- $\begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$ and $\begin{bmatrix} e_1 \\ e_2 \end{bmatrix}$ denote the allocations of Blotto and Enemy on the two battlefields 1 and 2.
- The battlefields values are normalized such that the first battlefield has fixed value of 1 and battlefield 2 has the value of w.
- The payoff is defined as the summed value of all battlefields conquered.
- Blotto and Enemy strategy can be either mixed or pure but the general case of a probabilistic strategy is defined by using probability measures μ_B and μ_E.
- Both allocations need to be positive on both battlefields and each is limited by resource constraints which are defined generally as $b_2 \leq f(b_1)$ and $e_2 \leq g(e_1)$.

Given the above specifications and constraints the optimization problem for Blotto is therefore:
$\max_{b_1,b_2}(\sum_{i=1}^{2}(Prob(b_i \geq e_i|\mu_E)\cdot\omega_i))$ and similarly $\max_{e_1,e_2}(\sum_{i=1}^{2}(Prob(e_i \geq b_i|\mu_B)\cdot\omega_i))$
To define the traditional linear setup of the Colonel Blotto game we can take the two constraint functions to be $f(b_1) = B - b_1$ and $g(e_1) = E - e_1$ for $B \geq E$
The analysis also requires the definition of two functions:

- $h(x) = g^{-1}(f(x))$
- $p(y) = g(f^{-1}(y))$

The intuition behind these functions is that $h(x)$ determines the value of the e_1 component for which applying $f(\cdot)$ to both x and $h(x)$ the result is the same and similarly $p(y)$ determines the value of e_2 for which applying $f^{-1}(\cdot)$ to both y and $p(y)$ the first components of Enemy and Blotto vectors correspond.

Determining the NE of games requires partitioning the possible games based on the difference of allocation between Blotto and the Enemy. Each game falls into a single partition called n, based on the greatest integer n that satisfies the the condition $E_2 \in [f(h^{n-2}(E_1)), f(h^{n-1}(E_1))]$
Intuitively, looking at the graphical representation of the game above, n is the number of bounces against the Blotto line when starting from any extreme of the enemy line and moving parallel to the axes until reaching the other axis.
Some particular sets will be fundamental in the following analysis. First of all let's describe some sets of allocations that contain Blotto best possible plays in a two battlefield game.

$$\forall i = 1, ..., n \quad T_i^b = \{(b_1, b_2) : (b_1 \geq h^{n-i}(E_1), b_2 \geq p^{i-1}(E_2), b_2 \leq f(b_1)\}$$

And similarly the best possible plays for the enemy

$$\forall i = 1, ..., n \quad T_i^b = \{(b_1, b_2) : (b_1 \geq h^{n-i}(E_1), b_2 \geq p^{i-1}(E_2), b_2 \leq f(b_1)\}$$

Then given an enemy play of $(x, g(x))$, define also the two following sets:

$$J_b^{x,i} = \{(b_1, b_2) : ((b_1, b_2) \in T_i^b, b_1 < x)\}$$

which denotes the cases in T_i^b where Blotto looses first battlefield

$$K_b^{x,i} = \{(b_1, b_2) : ((b_1, b_2) \in T_{i+1}^b, b_2 \geq g(x))\}$$

which denotes the cases in T_{i+1}^b where Blotto wins the second battlefield. The reasons for specifying these particular sets will be more evident later on. Similarly define for Blotto play $(x, f(x))$, the equivalent sets for Enemy.

$$J_e^{x,i} = \{(e_1, e_2) : ((e_1, e_2) \in T_{i+1}^e, e_1 \leq x)\}$$

$$K_e^{x,i} = \{(e_1, e_2) : ((e_1, e_2) \in T_i^e, e_2 > f(x))\}$$

i.e. the set of cases in T_{i+1}^e in which Enemy loses the first battlefield and the set of cases in T_i^e in which Enemy wins the second battlefield.
Now we can define Ω_B which is the set of probability measures μ_B that satisfy

- Property 1b

$$\forall i = 1, ..., n \quad \mu_B(T_i^b) = \frac{\omega^{n-i}}{\sum_{j=0}^{n-1} \omega^j}$$

which ensures that probability mass is assigned to each T_i^b according to its tactical relevance

- Property 2b

$$\forall i < 1, 2, ..., n-1 \quad \forall x \in [h^{n-1}(E_1), f^{-1}(p^{i-1}(E_2))] \quad \mu_B(J_b^{x,i}) \leq \mu_B(K_b^{x,i}) \cdot \omega$$

which ensures that enemy might not have an advantage playing an action within his expenditure but not belonging to any of the T_i^e. In other words makes playing outside the T_i^e not payoff improving.
Similarly but slightly differently Ω_E is defined as the set of probability measures μ_E that satisfy:

- Property 1e

$$\forall i = 1, ..., n \quad \mu_B(T_i^e) = \frac{\omega^{i-1}}{\sum_{j=0}^{n-1} \omega^j}$$

- Property 2e

$$\forall i < 1, 2, ..., n-1 \quad \forall x \in [h^{n-i-1}(E_1), f^{-1}(p^{i-1}(E_2))] \quad \mu_E(J_e^{x,i}) \leq \mu_E(K_e^{x,i}) \cdot \omega$$

The main result **?** is that

Theorem 2
Any pair of strategies $\{\mu_B, \mu_E\}$ such that $\mu_B \in \Omega_B$ and $\mu_E \in \Omega_E$ constitute a Nash Equilibrium and all these pairs are the only NE of the game

Proof 2
The proof relies principally on properties 2e and 2b. To prove any pair $\{\mu_B, \mu_E\}$ is a NE, it must be proved that any player would not get a better payoff expectation by deviating from its T_is. A better payoff expectation for player p could be achieved if playing $(x, f(x))$ or $(x, g(x))$ at full expenditure the additional probability of winning set$J_p^{x,i}$ is bigger than $K_p^{x,i}$. However properties 2b and 2e ensure that the additional probability of winning thanks to a deviation is weakly less than the probability of additional defeat. Since any deviation does not improve payoff, any pair $\{\mu_B, \mu_E\}$ is a Nash Equilibrium. The proof of equilibrium exhaustion is omitted and can be found explained in Macdonell and Mastronardi [2015].

2.2 The graphical algorithm

As it was mentioned before any Colonel Blotto game defined as above is characterised by the number of partitions n. Therefore it is helpful to consider a few of these possible partitions and see how they are connected with the difference of resources between the two players. Here following the two initial resources are normalized by dividing both by B.

$$B = 1, \quad 0 < E < 1$$

2.2.1 $E \leq 1/2$

Consider the case in which E is less or equal to half B. Blotto is in an excellent position as he can play even just $(\frac{B}{2}, \frac{B}{2})$ and win both battlefields. Therefore, if Blotto plays anything more than the minimum (E, E) he will be guaranteed to win. In this case, Enemy best strategy is very limited and relies upon the possibility that Blotto might not play his best play. The Enemy strategy is to play each of $(E, 0)$ and $(0, E)$ at 50% probability. Following the definition above we can clearly see that this is a $n = 1$ partition game.

2.2.2 $1/2 \leq E < 3/2$

In this case, Blotto resources are not enough to guarantee a win at best play. Enemy's best play is to try to attack heavily on one battlefield hoping to mismatch with Blotto's heavier allocation. Blotto instead hopes to match his heavier allocation with Enemy's heavier allocation to win on both fronts.
Let's consider how the possible best strategies evolve from the assumption that Enemy might play $(0, E)$. In this case, Blotto would want to play at least E on the second battlefield.

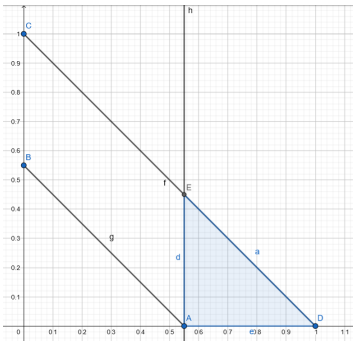

Figure 1: First Consideration

However this strategy exposes him to the threat of Enemy playing on the first battlefield $c \quad s.t. \quad E \geq c > f(E)$ therefore Blotto will want to consider playing in the area $\{(d, e) \quad s.t. \quad d > g^{-1}(f(E)) \quad and \quad e > B\}$ but also in part of the area considered before which is $\{(j, k) \quad s.t. \quad j > B \quad and \quad k > g(f^{-1}(B))\}$

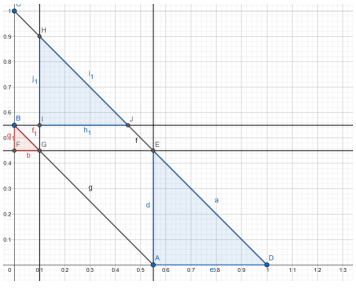

Figure 2: Finding T_i^b

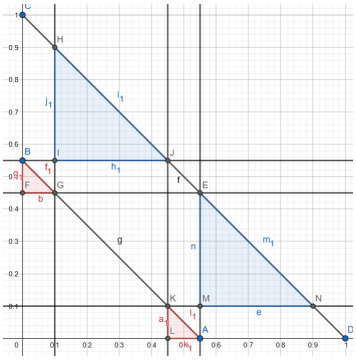

Figure 3: The final result of the graphical algorithm

It can be seen that, through these considerations, the sets $T_1^b, T_0^b, T_1^e, T_0^e$ where determined.
The graphical visualization also helps to highlight the role of the sets $J_b^{x,i}, K_b^{x,i}, J_e^{x,i}, K_e^{x,i}$. Suppose Enemy decided to play outside the prescribed sets by playing $(x, g(x)) \notin T_i^e$ for any $i \in 1, ..., n$

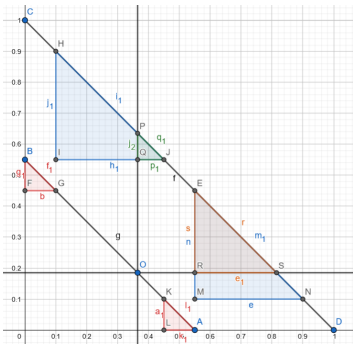

Figure 4: Enemy plays $O = (x, G(x))$ outside the prescribed sets. The blue area (to the left of the green area) above represents the J_i^e while the orange area denotes the K_i^e

As can be clearly seen $J_b^{x,1}$ denotes the area where Blotto would lose the first battlefield while $K_b^{x,1}$, denotes the area in T_{i+1} in which Blotto would win on the second front. Enemy deviating from the prescribed tries to gain an advantage on battlefield 1 by the measure of probability that Blotto assigns to $J_b^{x,1}$ at the cost of reducing his gains on battlefield 2 by the measure of probability that Blotto assigns to $K_b^{x,1}$. If the first probability measure was to be bigger than the second then Enemy would gain a significant advantage. That is why condition 2b is introduced. Similarly happens for $J_e^{x,i}, k_e^{x,i}$ with condition 2e.

2.3 $E \geq \frac{3}{2}$

Therefore we can now precise the steps of the graphical algorithm:

- Step 1
  1. Start from $(E, 0)$.
  2. Draw a vertical line until reaching Blotto constraint line and then rotate by 90 degrees counterclockwise.
  3. Draw an horizontal line until reaching Enemy constraint line and then rotate by 90 degrees clockwise.
  4. Complete the last horizontal line with a dotted line reaching the vertical axis.
  5. Repeat 2,3,4 until touching the vertical axis.
- Step 2
  1. Start from $(0, E)$.
  2. Draw an horizontal line until reaching Blotto constraint line and then rotate by 90 degrees clockwise.
  3. Draw an vertical line until reaching Enemy constraint line and then rotate by 90 degrees counterclockwise.
  4. Complete the last vertical line with a dotted line reaching the horizontal axis.
  5. Repeat 2,3,4 until touching the horizontal axis.
- Step 3
  1. Identify the Enemy T_i^e in the triangles created by the dotted lines.
  2. Identify the Blotto T_i^b in the triangles opposing the T_i^e.

The graphical algorithm provides a quick way to determine the NEs of the game when n is big ($\leftrightarrow$ E is close to B).

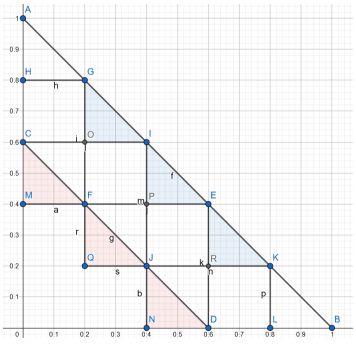

Figure 5: A partition 3 Blotto Game with T_i^e in red and T_i^bin blue

2.4 The Regret Learning Algorithm

Regret learning algorithm was introduced by Hart and Mas-Colell in 2000. The aim of the algorithm is to inform the strategic decision by taking into account the regret of the previous actions. Neller and Lanctot [2013]
Define the environment of the game:

- S_i the finite set of actions for player i
- $D = S_1 \times ... \times S_n$ set of all possible combinations of simultaneous actions of all players or "action profiles".
- u_i function mapping an action profile to utility for player i
- σ_i is the (possibly mixed) strategy of player i while σ_{-i} is the collection of the strategies of player i opponents.

The expected utility of a strategy is given by

$$\sum_{s \in S_i} \sum_{s' \in S_{-i}} \sigma_i \sigma_{-i} u_i(s, s')$$

The regret of not choosing an action is defined by the difference between the utility of that action (its payoff) and the utility of the action actually chosen, with respect to the fixed choices of other players. Then the regret for player i of not having played j against a_{-i} instead of a_i is

$$regret_i^j(A) = u_i(j, a_{-i}) - u_i(a_i, a_{-i})$$

where $j \in S_i, A = (a_i, a_{-i}) \in D, a \in S_i$ and $a_{-i} \in S_1 \times ... \times S_{i-1} \times S_{i+1} \times ... \times S_n$
The regret marching algorithm updates at each turn a vector of cumulative regrets for each player by adding the regrets of the last turn. The next action is then chosen according to the positive regrets with a strategy that is the normalised positive cumulative regrets.
The algorithm as in Neller and Lanctot [2013] is the following:

- For each player, initialize cumulative regrets to 0.
- For some number of iterations:
    1. Compute a regret matching strategy by normalising the positive elements in the cumulative regrets and ignoring the non-positive. (If all regrets are non-positive use uniform random strategy)
    2. Add the strategy profile to the strategy profile sum.
    3. Select each player action according to their strategy profile.
    4. Compute player regrets.
    5. Add player regrets to player cumulative regrets.
- Return the average strategy profile which is the strategy sum divided by the number of iterations.

The algorithm guarantees the convergence of the average strategy to a correlated equilibrium as proved in Hart and Mas-Colell [2000].

3 Regret Learning Algorithm Implementation in Blotto Games

Regret learning algorithm can be successfully applied to Colonel Blotto games, as in the following example.
The game implemented is defined as following:

- N (number of battle fields) = 3
- S (number of resources)= 5
- The allocation is discrete and both players have the same S
- All battlefields have the same value

Blotto Game Algorithm:

- STEP 1: INITIALIZATION
  1. Create a vector ν of possible strategies by solving the $(5,3)$ "stars and bars" problem.
  2. Determines payoff matrix entry $Payof f_{i,j} = sign(\sum_{k=0}^{n}(sign(\nu_i - \nu_j))_k)$
  3. For each player, initialise the vector of the sum of regrets (RegretSum).
  4. For each player, initialise the vector of the sum of strategies (StrategySum).
- STEP 2: FUNCTIONS DEFINITION
  1. Define GetStrategy function
     - Take RegretSum and StrategySum as arguments
     - Define Strategy vector by substituting the negative elements with zero and normalise the vector
     - If all elements are non-positive then Strategy is a uniform vector
     - Update the StrategySum vector of the player
     - Return Strategy
  2. Define GetAction function
     - Take Strategy
     - Produce random number in $(0,1)$
     - Determine the cumulative probability associated with action n
     - Return random action according to the Strategy profile
  3. Define Train
     - for a number of iterations
       * determine the action of each player by applying getAction to getStartegy of the player's RegretSum and StrategySum

        * determine the regret of not having played another action by the regret formula above and update the RegretSum vector.

    4. Define GetAverageStrategy
        - take StrategySum of a player
        - normalise StrategySum by the sum of its elements if different from a zero vector
        - else return a vector with 1/(number of possible actions) in each entry.

- STEP 3: CALCULATION

    1. Train for a fixed number of iterations (suggested 1000000)
    2. Apply getAverageStrategy to the Strategy Sum vectors and return the players average strategies in a table

The algorithm is then coded in python (Appendix A) and run over 1000000 iterations. Running the code, the average strategies for both players converge quickly to a Nash Equilibrium in which both players have the same mixed strategy (up to a small calculation error) which consists or randomising equally over all permutations of $(1,1,3)$ and $(0,2,3)$.

Blotto	Opponent	Battlefield 1	Battlefield 2	Battlefield 3
1.43E-07	4.76E-08	0	0	5
2.67E-06	7.73E-07	0	1	4
0.11113	0.114735	0	2	3
0.102173	0.110703	0	3	2
2.48E-06	1.81E-06	0	4	1
1.43E-07	4.76E-08	0	5	0
1.68E-06	2.61E-07	1	0	4
0.115996	0.106018	1	1	3
9.15E-07	3.58E-06	1	2	2
0.124285	0.110593	1	3	1
3.10E-07	6.75E-07	1	4	0
0.110588	0.114116	2	0	3
3.66E-06	6.96E-07	2	1	2
1.42E-06	2.85E-06	2	2	1
0.101794	0.111553	2	3	0
0.108033	0.111552	3	0	2
0.118604	0.109241	3	1	1
0.107372	0.111476	3	2	0
2.37E-06	7.39E-07	4	0	1
8.80E-06	9.82E-07	4	1	0
1.43E-07	9.36E-08	5	0	0

4 An application of Regret Matching to 2020 U.S. Presidential Elections

Colonel Blotto games have been used extensively in political, economical and strategical environments. In the following section, the Colonel Blotto setting will be used to model strategies for election funds allocation in the key 'Swing states' of 2020 U.S. Presidential Elections. Through the use of regret matching algorithm, the best mixed strategies for both candidates will be highlighted.

U.S. presidential election is a type of indirect election in which US citizens eligible to vote in one of the fifty states or in Washington D.C. cast their vote for their state (or district) members of the Electoral College. The members of the Electoral College then proceed to nominate a president. The Electoral College is composed of 538 members. For a candidate to become U.S. President, requires the vote of at least 270 members of the Electoral College. In case of a Electoral College tie, the President is then elected by the House of Representatives.

Each of the 50 states has a number of members of the Electoral College to be elected which tends to be proportional to the population of the state. Washington D.C. elects the same number of Electors as the least populated state (3). The members of the Electoral College cast their vote according to the candidate that won the popular vote within their state (or district in case of Nebraska and Maine).

A particularity of the U.S. political landscape is the dominance of two major parties: the "Democratic Party" and the "Republican Party". Furthermore, many states represent political strongholds for one of these two parties. The Cook Political Report is an independent institute that assesses the "leaning" of the population of each state towards voting one of the two parties and determines the so-called "swing states". Swing states are defined as the states where there is no dominance of one voting intention over the other. Therefore, swing states represent the battleground for U.S. Electoral campaigns.

Table 1: Number of members of Electoral College for each Swing State

Arizona	15
Georgia	16
Pennsylvania	20
North Carolina	15
Wisconsin	10

An effective Colonel Blotto game can be developed to describe the political situation. The algorithm and the code provided can depict quite well any political situation with different swing states. However, the number of possible strategies increases dramatically with both the number of states considered and the amount of resources, making computational accuracy and timings unfeasible for complex situations. Therefore, the number of states considered should be restricted to the very minimum and similarly the amount of resources for each candidate. Many states have been highlighted this year as swing states, however, looking back at the election results, many of these states actually expressed a strong political preference. In the model, the five most close races where selected as the battlefields for a Colonel Blotto game between the two presidential candidates: Biden and Trump.

The five states Swing States selected are Arizona, Georgia, Pennsylvania, North Carolina and Wisconsin. Cook [2020] The remaining states electors were given to their actual winner as in the election result. The model tries to depict a situation in which both candidates try to allocate electoral funds just before the election day in these five key states. The Democrats are modeled to have already secured 249 electors, needing 21 to elect Biden while the Republicans are modeled to be carrying 217 electors, needing 53 to elect Trump.

The resources are the same for both of them and the allocation is discrete. The number of resources is 6 for each. The number was chosen to give enough resources to be playing on every state if necessary with a surplus to strengthen important battlefields. Each candidate wins a state if he spent more resources there than the opponent. In case of draw, the state is awarded according to the status quo which, in this example, is modeled to be (due to historical voting patterns) Trump favourable in Arizona, Georgia and North Carolina while Biden favourable in Pennsylvania and Wisconsin.

Table 2: Status Quo of Swing States

Arizona	Trump
Georgia	Trump
Pennsylvania	Biden
North Carolina	Trump
Wisconsin	Biden

The payoff matrix can be modeled in two different ways: in one case (appendix B), the winner gains 1 while the loser loses 1, in the other case the winner gets payoff 1 while the loser doesn't gain or lose anything(appendix C). The first model depicts a very "harsh" electoral climate in which both parties have significant stakes to be lost in case the other candidate wins. The two models produce significantly different best strategies when the regret matching algorithm is applied, but the first model seems to be a more accurate depiction of the current political climate.

To implement the model, the payoff matrix of the algorithm described in the previous section for regret matching in Colonel Blotto games must be modified.
In the Harsh political climate game, the payoff for Biden when he plays action v_i against opponent action v_j is

$$payoff_B(i,j) = sign(\delta + sign(q + 2sign(v_i - v_j) \cdot \mu))$$

where δ is the difference between the carried democratic electors and the republican ones, q is the vector of the current status quo (with $+1$ indicating Biden victory in the state and -1 indicating Trump victory), v is the vector of possible actions and μ is the vector of the states values.

Similarly, payoff for Trump when plays action v_i against opponent action v_j is

$$payoff_T(i,j) = sign(-\delta + sign(-q + 2sign(v_i - v_j) \cdot \mu))$$

The two different payoffs are then employed in calculating the action utility vector in the train part of the code.

In the more peaceful political climate, the payoffs are similar but map the two actions to either 0 in case of tie or defeat and 1 in case of victory.

$$payoff_B(i,j) = \begin{cases} 1 & if \quad \delta + sign(q + 2sign(v_i - v_j) \cdot \mu) > 0 \\ 0 & otherwise \end{cases}$$

$$payoff_T(i,j) = \begin{cases} 1 & if \quad -\delta + sign(-q + 2sign(v_i - v_j) \cdot \mu) > 0 \\ 0 & otherwise \end{cases}$$

4.1 Tables and Results

In the case of the harsh political climate, the code (Appendix B) was run over 1000000 iterations and produced the mixed strategies presented in the first following table.
What seems surprising in the mixed strategies of the first "harsh" setting is the prevalence of actions allocating few to no funds in Pennsylvania for both candidates.

In the case of Trump, this seems particularly contradictory as he can not win the game without gaining Pennsylvania and three other states. However, the strategy makes sense in the context of regret matching, due to the fact that a tie in the presidential race gives a better payoff than risking a defeat. Therefore, since Biden needs 21 votes and Pennsylvania yields only 20, giving up the state to block Biden in other states is the best play in order to get a tie .

In the case of Biden, the strategy is explained by Biden's status quo advantage in Pennsylvania and the prevalence of Trump's defecting in the key state while attacking with more strength other Trump holds like Arizona. The mixed strategies however still appear to be quite turbulent.

A completely different case is the one of the more "peaceful" election climate. In this model a tie does not give an advantage over a defeat as they are both valued zero. Therefore, both players try to win but have no advantage in undercutting the other player chances of victory. The strategies are again produced by running over 1000000 training iteration the code (appendix C) and reported in the second table.

The strategy for Biden converges quite quickly to the pure strategy of playing $(1, 1, 3, 1, 0)$. This action is particularly fruitful because tries to cement the victory in Pennsylvania while attacking Trump holds in Arizona, Georgia and North Carolina.

On the other side, Trump has a more mixed strategy in which the most frequent actions suggest to hold strong in Pennsylvania while assigning minor funds to defend two holds and possibly attack with the remaining.

4.2 The ”harsh” competition model strategies

2020 U.S. Presidential Elections - harsh competition model						
Trump	**Biden**	**Arizona**	**Georgia**	**Pennsylvania**	**NorthCarolina**	**Wisconsin**
6.12E-08	2.86E-08	0	0	0	0	6
6.12E-08	2.86E-08	0	0	0	1	5
2.63E-08	2.86E-08	0	0	0	2	4
2.63E-08	0.00873	0	0	0	3	3
2.63E-08	9.51E-08	0	0	0	4	2
4.76E-09	2.86E-08	0	0	0	5	1
4.76E-09	2.86E-08	0	0	0	6	0
6.12E-08	2.86E-08	0	0	1	0	5
0.005357	2.86E-08	0	0	1	1	4
0.00517	4.64E-07	0	0	1	2	3
1.60E-05	0.00637	0	0	1	3	2
4.76E-09	2.86E-08	0	0	1	4	1
4.76E-09	2.86E-08	0	0	1	5	0
0.009751	2.86E-08	0	0	2	0	4
0.0035	2.86E-08	0	0	2	1	3
0.022954	0.007103	0	0	2	2	2
0.010684	0.002395	0	0	2	3	1
4.76E-09	2.86E-08	0	0	2	4	0
0.015466	1.44E-06	0	0	3	0	3
0.009703	0.015046	0	0	3	1	2
0.013863	2.78E-05	0	0	3	2	1
3.86E-07	0.00962	0	0	3	3	0
6.94E-06	5.01E-07	0	0	4	0	2
0.011436	3.26E-08	0	0	4	1	1
7.45E-08	6.44E-07	0	0	4	2	0
7.45E-08	2.86E-08	0	0	5	0	1
7.45E-08	2.86E-08	0	0	5	1	0
4.69E-07	2.86E-08	0	0	6	0	0
6.12E-08	2.86E-08	0	1	0	0	5
2.63E-08	2.86E-08	0	1	0	1	4
1.15E-06	2.53E-07	0	1	0	2	3
2.63E-08	0.005474	0	1	0	3	2
4.76E-09	2.86E-08	0	1	0	4	1
4.76E-09	2.86E-08	0	1	0	5	0
0.004434	2.86E-08	0	1	1	0	4
0.013145	3.49E-08	0	1	1	1	3
0.013883	0.005122	0	1	1	2	2
0.007174	1.23E-05	0	1	1	3	1
3.43E-08	2.86E-08	0	1	1	4	0
0.002768	2.86E-08	0	1	2	0	3
0.005317	0.015543	0	1	2	1	2

2020 U.S. Presidential Elections - harsh competition model						
Trump	**Biden**	**Arizona**	**Georgia**	**Pennsylvania**	**NorthCarolina**	**Wisconsin**
0.017899	0.018388	0	1	2	2	1
9.58E-08	0.001125	0	1	2	3	0
0.009761	0.015372	0	1	3	0	2
0.004108	1.32E-06	0	1	3	1	1
0.007295	1.52E-06	0	1	3	2	0
0.012283	3.26E-08	0	1	4	0	1
0.006326	2.82E-07	0	1	4	1	0
7.45E-08	2.86E-08	0	1	5	0	0
2.63E-08	2.86E-08	0	2	0	0	4
4.69E-06	2.86E-08	0	2	0	1	3
1.75E-05	0.011748	0	2	0	2	2
4.76E-09	1.83E-06	0	2	0	3	1
4.76E-09	5.41E-08	0	2	0	4	0
0.005713	2.86E-08	0	2	1	0	3
0.01443	0.004762	0	2	1	1	2
0.014491	0.022967	0	2	1	2	1
4.85E-07	2.25E-06	0	2	1	3	0
0.022901	0.007353	0	2	2	0	2
0.01848	0.018459	0	2	2	1	1
0.00064	0.018348	0	2	2	2	0
0.014402	0.00012	0	2	3	0	1
0.006985	4.56E-06	0	2	3	1	0
4.26E-07	3.99E-07	0	2	4	0	0
2.63E-08	0.009635	0	3	0	0	3
2.63E-08	0.005448	0	3	0	1	2
4.76E-09	3.47E-07	0	3	0	2	1
4.76E-09	0.013199	0	3	0	3	0
1.62E-05	0.005419	0	3	1	0	2
0.006101	1.59E-04	0	3	1	1	1
1.74E-07	2.44E-06	0	3	1	2	0
0.010789	0.002601	0	3	2	0	1
4.76E-09	0.001205	0	3	2	1	0
4.52E-07	0.008916	0	3	3	0	0
2.63E-08	9.51E-08	0	4	0	0	2
4.76E-09	2.86E-08	0	4	0	1	1
4.76E-09	2.86E-08	0	4	0	2	0
4.76E-09	2.86E-08	0	4	1	0	1
4.76E-09	2.86E-08	0	4	1	1	0
4.76E-09	2.86E-08	0	4	2	0	0
4.76E-09	2.86E-08	0	5	0	0	1
4.76E-09	2.86E-08	0	5	0	1	0
4.76E-09	2.86E-08	0	5	1	0	0
4.76E-09	2.86E-08	0	6	0	0	0
6.12E-08	4.48E-08	1	0	0	0	5

2020 U.S. Presidential Elections - harsh competition model

Trump	Biden	Arizona	Georgia	Pennsylvania	NorthCarolina	Wisconsin
2.63E-08	4.48E-08	1	0	0	1	4
3.18E-06	1.32E-07	1	0	0	2	3
1.55E-07	0.005577	1	0	0	3	2
9.19E-09	1.32E-07	1	0	0	4	1
4.76E-09	4.48E-08	1	0	0	5	0
0.003624	4.48E-08	1	0	1	0	4
0.013137	4.48E-08	1	0	1	1	3
0.013075	0.004639	1	0	1	2	2
0.005405	1.18E-04	1	0	1	3	1
4.76E-09	4.48E-08	1	0	1	4	0
0.003986	3.67E-08	1	0	2	0	3
0.00693	0.015466	1	0	2	1	2
0.017213	0.019202	1	0	2	2	1
4.76E-09	0.001042	1	0	2	3	0
0.011056	0.015738	1	0	3	0	2
0.003901	2.88E-07	1	0	3	1	1
0.006519	5.12E-06	1	0	3	2	0
0.010534	8.02E-08	1	0	4	0	1
0.007237	6.47E-08	1	0	4	1	0
7.45E-08	3.67E-08	1	0	5	0	0
2.63E-08	2.26E-07	1	1	0	0	4
0.002058	1.29E-06	1	1	0	1	3
0.044084	3.10E-05	1	1	0	2	2
9.19E-09	0.002481	1	1	0	3	1
4.76E-09	4.10E-07	1	1	0	4	0
0.014849	3.95E-07	1	1	1	0	3
2.25E-06	2.09E-05	1	1	1	1	2
3.43E-08	0.036682	1	1	1	2	1
3.06E-06	0.006535	1	1	1	3	0
0.005925	0.015205	1	1	2	0	2
4.76E-09	1.17E-05	1	1	2	1	1
0.017717	0.019142	1	1	2	2	0
0.003397	1.53E-06	1	1	3	0	1
0.014651	7.20E-06	1	1	3	1	0
0.006101	6.99E-07	1	1	4	0	0
1.53E-06	3.97E-07	1	2	0	0	3
0.044757	1.47E-05	1	2	0	1	2
0.021662	0.034596	1	2	0	2	1
4.76E-09	5.61E-06	1	2	0	3	0
0.012559	0.004939	1	2	1	0	2
1.27E-06	0.036307	1	2	1	1	1
0.020346	0.034254	1	2	1	2	0
0.018272	0.017553	1	2	2	0	1
0.016395	0.020068	1	2	2	1	0

2020 U.S. Presidential Elections - harsh competition model						
Trump	**Biden**	**Arizona**	**Georgia**	**Pennsylvania**	**NorthCarolina**	**Wisconsin**
0.006462	3.04E-06	1	2	3	0	0
5.86E-08	0.004642	1	3	0	0	2
4.76E-09	0.002964	1	3	0	1	1
4.76E-09	2.12E-05	1	3	0	2	0
0.006382	1.18E-04	1	3	1	0	1
1.31E-08	0.006588	1	3	1	1	0
1.27E-06	0.001076	1	3	2	0	0
4.76E-09	1.32E-07	1	4	0	0	1
4.76E-09	4.48E-08	1	4	0	1	0
4.76E-09	4.48E-08	1	4	1	0	0
4.76E-09	4.48E-08	1	5	0	0	0
5.56E-08	1.46E-07	2	0	0	0	4
3.00E-06	2.62E-07	2	0	0	1	3
2.41E-05	0.011618	2	0	0	2	2
9.19E-09	1.11E-06	2	0	0	3	1
4.76E-09	1.46E-07	2	0	0	4	0
0.005924	8.75E-08	2	0	1	0	3
0.013832	0.004955	2	0	1	1	2
0.015271	0.023072	2	0	1	2	1
4.36E-07	1.01E-06	2	0	1	3	0
0.022426	0.007139	2	0	2	0	2
0.018194	0.018803	2	0	2	1	1
0.000886	0.018687	2	0	2	2	0
0.014452	0.000166	2	0	3	0	1
0.006311	4.18E-06	2	0	3	1	0
5.77E-07	9.20E-08	2	0	4	0	0
6.26E-08	6.83E-07	2	1	0	0	3
0.046759	1.74E-05	2	1	0	1	2
0.019412	0.034594	2	1	0	2	1
4.76E-09	5.46E-06	2	1	0	3	0
0.013497	0.005381	2	1	1	0	2
5.68E-06	0.036156	2	1	1	1	1
0.018719	0.034826	2	1	1	2	0
0.018374	0.018379	2	1	2	0	1
0.017319	0.02008	2	1	2	1	0
0.006436	4.06E-06	2	1	3	0	0
1.80E-05	0.012056	2	2	0	0	2
0.018924	0.03401	2	2	0	1	1
4.76E-09	0.02515	2	2	0	2	0
0.01587	0.023143	2	2	1	0	1
0.017848	0.034252	2	2	1	1	0
0.000902	0.018216	2	2	2	0	0
4.76E-09	2.70E-07	2	3	0	0	1
4.76E-09	2.93E-06	2	3	0	1	0

2020 U.S. Presidential Elections - harsh competition model						
Trump	**Biden**	**Arizona**	**Georgia**	**Pennsylvania**	**NorthCarolina**	**Wisconsin**
4.76E-09	6.06E-07	2	3	1	0	0
4.76E-09	1.46E-07	2	4	0	0	0
5.56E-08	0.009902	3	0	0	0	3
2.27E-07	0.004889	3	0	0	1	2
4.76E-09	5.91E-07	3	0	0	2	1
4.76E-09	0.012739	3	0	0	3	0
2.19E-05	0.005622	3	0	1	0	2
0.006246	1.59E-04	3	0	1	1	1
4.76E-09	3.15E-06	3	0	1	2	0
0.010979	0.002467	3	0	2	0	1
4.76E-09	0.000614	3	0	2	1	0
7.45E-08	0.009629	3	0	3	0	0
5.56E-08	0.004718	3	1	0	0	2
4.76E-09	0.002201	3	1	0	1	1
4.76E-09	5.82E-07	3	1	0	2	0
0.005823	1.01E-05	3	1	1	0	1
4.76E-09	0.006555	3	1	1	1	0
2.60E-07	0.001081	3	1	2	0	0
4.76E-09	1.51E-06	3	2	0	0	1
4.76E-09	7.15E-06	3	2	0	1	0
2.72E-07	1.99E-06	3	2	1	0	0
4.76E-09	0.012932	3	3	0	0	0
5.56E-08	9.51E-08	4	0	0	0	2
4.76E-09	2.86E-08	4	0	0	1	1
4.76E-09	2.86E-08	4	0	0	2	0
4.76E-09	2.86E-08	4	0	1	0	1
2.31E-08	2.86E-08	4	0	1	1	0
4.76E-09	2.86E-08	4	0	2	0	0
4.76E-09	2.86E-08	4	1	0	0	1
4.76E-09	2.86E-08	4	1	0	1	0
4.76E-09	2.86E-08	4	1	1	0	0
4.76E-09	5.41E-08	4	2	0	0	0
4.76E-09	2.86E-08	5	0	0	0	1
4.76E-09	2.86E-08	5	0	0	1	0
4.76E-09	2.86E-08	5	0	1	0	0
4.76E-09	2.86E-08	5	1	0	0	0
4.76E-09	2.86E-08	6	0	0	0	0

4.3 The ”relaxed” competition model strategies

2020 U.S. Presidential Elections - ”relaxed” competition model						
Trump	**Biden**	**Arizona**	**Georgia**	**Pennsylvania**	**NorthCarolina**	**Wisconsin**
4.76E-09	1.43E-08	0	0	0	0	6
4.76E-09	1.43E-08	0	0	0	1	5
4.76E-09	7.26E-08	0	0	0	2	4
4.76E-09	1.88E-05	0	0	0	3	3
4.76E-09	5.14E-08	0	0	0	4	2
4.76E-09	3.87E-08	0	0	0	5	1
4.76E-09	1.43E-08	0	0	0	6	0
4.72E-07	1.43E-08	0	0	1	0	5
2.13E-05	1.43E-08	0	0	1	1	4
1.51E-05	2.32E-07	0	0	1	2	3
8.71E-07	4.76E-06	0	0	1	3	2
1.06E-06	2.50E-07	0	0	1	4	1
4.76E-09	1.43E-08	0	0	1	5	0
1.22E-06	1.76E-08	0	0	2	0	4
2.83E-06	8.65E-08	0	0	2	1	3
3.91E-05	1.41E-05	0	0	2	2	2
6.05E-03	5.28E-07	0	0	2	3	1
2.94E-07	1.76E-08	0	0	2	4	0
0.018189	2.36E-06	0	0	3	0	3
0.018217	1.96E-05	0	0	3	1	2
5.09E-05	2.33E-06	0	0	3	2	1
2.79E-07	3.34E-06	0	0	3	3	0
1.84E-06	5.14E-08	0	0	4	0	2
4.16E-05	2.00E-07	0	0	4	1	1
4.76E-09	7.26E-08	0	0	4	2	0
4.76E-09	3.87E-08	0	0	5	0	1
4.76E-09	1.43E-08	0	0	5	1	0
4.76E-09	1.43E-08	0	0	6	0	0
4.76E-09	1.43E-08	0	1	0	0	5
4.76E-09	1.43E-08	0	1	0	1	4
4.76E-09	3.50E-08	0	1	0	2	3
4.76E-09	1.20E-05	0	1	0	3	2
4.76E-09	3.87E-08	0	1	0	4	1
4.76E-09	1.43E-08	0	1	0	5	0
0.036284	1.43E-08	0	1	1	0	4
0.03629	1.43E-08	0	1	1	1	3
0.012103	7.06E-06	0	1	1	2	2
0.06653	7.35E-07	0	1	1	3	1
4.76E-09	1.43E-08	0	1	1	4	0
4.23E-02	1.43E-08	0	1	2	0	3
0.018176	8.80E-07	0	1	2	1	2

2020 U.S. Presidential Elections - "relaxed" competition model						
Trump	**Biden**	**Arizona**	**Georgia**	**Pennsylvania**	**NorthCarolina**	**Wisconsin**
6.00E-05	5.34E-05	0	1	2	2	1
4.76E-09	1.76E-08	0	1	2	3	0
5.58E-05	1.92E-05	0	1	3	0	2
0.030322	2.33E-06	0	1	3	1	1
2.44E-05	1.15E-05	0	1	3	2	0
1.83E-05	3.87E-08	0	1	4	0	1
8.06E-06	1.43E-08	0	1	4	1	0
4.76E-09	1.43E-08	0	1	5	0	0
4.76E-09	3.14E-08	0	2	0	0	4
4.76E-09	3.54E-08	0	2	0	1	3
4.76E-09	2.51E-05	0	2	0	2	2
4.76E-09	8.31E-07	0	2	0	3	1
4.76E-09	3.14E-08	0	2	0	4	0
0.036275	3.97E-08	0	2	1	0	3
3.69E-05	5.09E-07	0	2	1	1	2
1.64E-05	3.26E-05	0	2	1	2	1
1.46E-07	3.26E-07	0	2	1	3	0
4.75E-05	1.69E-05	0	2	2	0	2
0.018174	3.44E-05	0	2	2	1	1
1.28E-05	3.91E-05	0	2	2	2	0
0.024201	1.39E-06	0	2	3	0	1
2.91E-06	3.54E-08	0	2	3	1	0
4.76E-09	3.14E-08	0	2	4	0	0
4.76E-09	1.44E-05	0	3	0	0	3
4.76E-09	9.32E-06	0	3	0	1	2
4.76E-09	1.59E-07	0	3	0	2	1
4.76E-09	2.70E-05	0	3	0	3	0
1.35E-06	6.36E-06	0	3	1	0	2
0.006075	2.33E-07	0	3	1	1	1
3.30E-06	1.22E-07	0	3	1	2	0
5.63E-07	2.21E-07	0	3	2	0	1
8.47E-07	3.19E-07	0	3	2	1	0
4.48E-08	8.01E-06	0	3	3	0	0
4.76E-09	5.14E-08	0	4	0	0	2
4.76E-09	2.00E-07	0	4	0	1	1
4.76E-09	7.26E-08	0	4	0	2	0
5.63E-07	2.50E-07	0	4	1	0	1
8.47E-07	1.43E-08	0	4	1	1	0
4.48E-08	1.76E-08	0	4	2	0	0
4.76E-09	3.87E-08	0	5	0	0	1
4.76E-09	1.43E-08	0	5	0	1	0
4.48E-08	1.43E-08	0	5	1	0	0
4.76E-09	1.43E-08	0	6	0	0	0
4.76E-09	2.87E-08	1	0	0	0	5

2020 U.S. Presidential Elections - ”relaxed” competition model

Trump	Biden	Arizona	Georgia	Pennsylvania	NorthCarolina	Wisconsin
4.76E-09	2.87E-08	1	0	0	1	4
4.76E-09	8.02E-07	1	0	0	2	3
4.76E-09	1.01E-05	1	0	0	3	2
4.76E-09	2.22E-07	1	0	0	4	1
4.76E-09	2.87E-08	1	0	0	5	0
2.86E-06	5.70E-08	1	0	1	0	4
8.35E-06	1.30E-07	1	0	1	1	3
3.35E-05	1.25E-06	1	0	1	2	2
0.018149	4.41E-07	1	0	1	3	1
2.15E-07	5.70E-08	1	0	1	4	0
2.93E-05	1.19E-07	1	0	2	0	3
0.012144	2.26E-06	1	0	2	1	2
0.042363	3.55E-05	1	0	2	2	1
4.16E-07	1.19E-07	1	0	2	3	0
5.45E-05	7.28E-06	1	0	3	0	2
0.102891	1.20E-05	1	0	3	1	1
3.34E-06	2.57E-06	1	0	3	2	0
1.71E-05	2.22E-07	1	0	4	0	1
4.94E-06	2.87E-08	1	0	4	1	0
4.76E-09	2.87E-08	1	0	5	0	0
4.76E-09	2.87E-08	1	1	0	0	4
4.76E-09	2.87E-08	1	1	0	1	3
4.76E-09	7.31E-07	1	1	0	2	2
4.76E-09	4.03E-06	1	1	0	3	1
4.76E-09	2.87E-08	1	1	0	4	0
2.10E-05	5.70E-08	1	1	1	0	3
4.62E-06	9.11E-07	1	1	1	1	2
2.16E-06	6.62E-05	1	1	1	2	1
1.41E-06	1.05E-07	1	1	1	3	0
3.01E-05	5.55E-07	1	1	2	0	2
8.36E-07	5.82E-05	1	1	2	1	1
7.80E-05	8.49E-05	1	1	2	2	0
0.096829	5.86E-06	1	1	3	0	1
0.018227	0.998063	1	1	3	1	0
2.38E-07	2.87E-08	1	1	4	0	0
4.76E-09	7.05E-08	1	2	0	0	3
4.76E-09	1.90E-07	1	2	0	1	2
4.76E-09	4.20E-05	1	2	0	2	1
4.76E-09	7.05E-08	1	2	0	3	0
1.62E-05	1.88E-06	1	2	1	0	2
2.62E-07	5.92E-05	1	2	1	1	1
4.24E-05	8.10E-05	1	2	1	2	0
0.04236	2.83E-05	1	2	2	0	1
6.17E-05	0.000114	1	2	2	1	0

2020 U.S. Presidential Elections - "relaxed" competition model						
Trump	**Biden**	**Arizona**	**Georgia**	**Pennsylvania**	**NorthCarolina**	**Wisconsin**
1.96E-05	3.64E-07	1	2	3	0	0
4.76E-09	9.54E-06	1	3	0	0	2
4.76E-09	6.04E-06	1	3	0	1	1
4.76E-09	1.49E-06	1	3	0	2	0
2.50E-06	4.41E-07	1	3	1	0	1
3.00E-06	2.35E-07	1	3	1	1	0
4.48E-08	9.87E-08	1	3	2	0	0
4.76E-09	2.22E-07	1	4	0	0	1
4.76E-09	2.87E-08	1	4	0	1	0
4.48E-08	5.70E-08	1	4	1	0	0
4.76E-09	2.87E-08	1	5	0	0	0
4.76E-09	1.09E-07	2	0	0	0	4
4.76E-09	3.04E-07	2	0	0	1	3
4.76E-09	2.83E-05	2	0	0	2	2
4.76E-09	5.38E-06	2	0	0	3	1
4.76E-09	9.83E-08	2	0	0	4	0
1.30E-05	3.02E-07	2	0	1	0	3
3.84E-05	1.13E-06	2	0	1	1	2
0.0182	1.93E-05	2	0	1	2	1
1.07E-07	5.60E-07	2	0	1	3	0
8.35E-05	1.92E-05	2	0	2	0	2
0.018176	5.16E-05	2	0	2	1	1
3.43E-06	2.12E-05	2	0	2	2	0
0.060524	6.58E-06	2	0	3	0	1
9.56E-06	5.57E-06	2	0	3	1	0
4.76E-09	1.09E-07	2	0	4	0	0
4.76E-09	5.14E-08	2	1	0	0	3
4.76E-09	7.91E-06	2	1	0	1	2
4.76E-09	6.39E-05	2	1	0	2	1
4.76E-09	5.54E-06	2	1	0	3	0
0.030261	4.99E-06	2	1	1	0	2
7.53E-07	5.78E-05	2	1	1	1	1
0.078647	6.23E-05	2	1	1	2	0
5.18E-05	2.02E-05	2	1	2	0	1
0.012153	7.55E-05	2	1	2	1	0
1.51E-06	9.69E-07	2	1	3	0	0
4.76E-09	2.80E-05	2	2	0	0	2
4.76E-09	5.19E-05	2	2	0	1	1
4.76E-09	5.32E-05	2	2	0	2	0
0.018178	3.92E-05	2	2	1	0	1
0.048435	0.000103	2	2	1	1	0
4.96E-06	4.07E-05	2	2	2	0	0
4.76E-09	2.73E-06	2	3	0	0	1
4.76E-09	8.22E-06	2	3	0	1	0

2020 U.S. Presidential Elections - ”relaxed” competition model						
Trump	**Biden**	**Arizona**	**Georgia**	**Pennsylvania**	**NorthCarolina**	**Wisconsin**
4.48E-08	1.91E-07	2	3	1	0	0
4.76E-09	1.09E-07	2	4	0	0	0
4.76E-09	4.69E-06	3	0	0	0	3
4.76E-09	1.61E-05	3	0	0	1	2
4.76E-09	2.33E-06	3	0	0	2	1
4.76E-09	1.64E-05	3	0	0	3	0
2.61E-07	7.01E-06	3	0	1	0	2
2.51E-05	3.94E-07	3	0	1	1	1
4.76E-09	4.90E-07	3	0	1	2	0
1.14E-05	5.28E-07	3	0	2	0	1
4.76E-09	7.24E-07	3	0	2	1	0
4.76E-09	8.14E-06	3	0	3	0	0
4.76E-09	2.83E-05	3	1	0	0	2
4.76E-09	9.49E-06	3	1	0	1	1
4.76E-09	2.34E-06	3	1	0	2	0
0.012094	7.35E-07	3	1	1	0	1
1.80E-06	1.43E-08	3	1	1	1	0
4.76E-09	1.76E-08	3	1	2	0	0
4.76E-09	1.39E-06	3	2	0	0	1
4.76E-09	2.25E-06	3	2	0	1	0
4.76E-09	3.26E-07	3	2	1	0	0
4.76E-09	1.67E-05	3	3	0	0	0
4.76E-09	5.14E-08	4	0	0	0	2
4.76E-09	2.00E-07	4	0	0	1	1
4.76E-09	7.26E-08	4	0	0	2	0
2.30E-08	2.50E-07	4	0	1	0	1
4.76E-09	1.43E-08	4	0	1	1	0
4.76E-09	1.76E-08	4	0	2	0	0
4.76E-09	3.87E-08	4	1	0	0	1
4.76E-09	1.43E-08	4	1	0	1	0
4.76E-09	1.43E-08	4	1	1	0	0
4.76E-09	3.14E-08	4	2	0	0	0
4.76E-09	3.87E-08	5	0	0	0	1
4.76E-09	1.43E-08	5	0	0	1	0
4.76E-09	1.43E-08	5	0	1	0	0
4.76E-09	1.43E-08	5	1	0	0	0
4.76E-09	1.43E-08	6	0	0	0	0

5 Bibliography

References

Sergiu Hart and Andreu Mas-Colell. A simple adaptive procedure leading to correlated equilibrium. **Econometrica**, 68(5):1127–1150, 2000. ISSN 0012-9682. pages 2, 10

J. Nash. Equilibrium points in n-person games. **Proceedings of the National Academy of Sciences of the United States of America**, 36 1:48–9, 1950. pages 3

Lecturer Asu Ozdaglar. Mit open courseware 6.254: Game theory with engineering applications february 23, 2010 lecture 6: Continuous and discontinuous games, 2013. pages 3

Scott T. Macdonell and Nick Mastronardi. Waging simple wars: a complete characterization of two-battlefield blotto equilibria. **Economic Theory**, 58:183–216, 2015. pages 4, 6

T. Neller and Marc Lanctot. An introduction to counterfactual regret minimization. 2013. pages 10

Cook. 2020 electoral college ratings, 2020. pages 14

6 Appendix A

```
import random
import numpy as np
from itertools import product

#create payoff and list
dec = np.array([np.array(i) for i in product(range(5+1), repeat=3) if
                                        sum(i)==5]);
num_actions=len(dec);
payoff=np.zeros((num_actions,num_actions));
for i in range (num_actions):
    for j in range(num_actions):
        payoff[i][j]= np.sign(((dec[i]-dec[j])>0).sum()-((dec[i]-dec[
                                                j])<0).sum())
#initialize
regretSum=np.zeros(num_actions);
oppRegretSum=np.zeros(num_actions);

strategySum=np.zeros(num_actions);
oppStrategySum=np.zeros(num_actions);


def getStrategy(regretSum,sSum):
    normalizingSum=0;
    strategy=np.zeros(num_actions)
    Sum=sSum
    for a in range(num_actions):
     if regretSum[a]>0 :
      strategy[a]=regretSum[a]
     else:
         strategy[a]=0
     normalizingSum += strategy[a]

    for a in range(num_actions):
        if normalizingSum>0 :
            strategy[a] /= normalizingSum
        else:
            strategy[a]= 1/num_actions
        Sum[a]+=strategy[a]
    sSum=Sum
    return(strategy)


def getAction(strategy)  :
    r=random.random()
    a=0
    cumulativeProbability=0
    while a < num_actions-1:
        cumulativeProbability+=strategy[a]
        if r < cumulativeProbability:
            break
        a+=1
```

```
    return(a)

def Train(iterations):
    for i in range(iterations+1):
         newStrategy=getStrategy(regretSum, strategySum)
         oppNewStrategy=getStrategy(oppRegretSum, oppStrategySum)
         myAction=getAction(newStrategy)
         oppAction=getAction(oppNewStrategy)
    #get utility
         actionUtility=payoff[:,oppAction]
         oppActionUtility=payoff[:,myAction]
    #get regret
         for a in range(num_actions):
          regretSum[a]+=actionUtility[a]-actionUtility[myAction]
          oppRegretSum[a]+=oppActionUtility[a]-oppActionUtility[
                                              oppAction]



def getAverageStrategy(straSum):
    avgStrategy=np.zeros(num_actions)
    normalizingSum=0
    for a in range(num_actions):
        normalizingSum+=straSum[a]
    for a in range(num_actions):
        if normalizingSum >0:
            avgStrategy[a]=straSum[a]/normalizingSum
        else:
            avgStrategy[a]=1.0/num_actions
    return(avgStrategy)

Train(1000000);
Strategies=np.zeros((21,2))
Strategies[:,0]=getAverageStrategy(strategySum)
Strategies[:,1]=getAverageStrategy(oppStrategySum)
print(Strategies)
```

7 Appendix B

```
import random
import numpy as np
from itertools import product
#
dem_states=249
rep_states=217

#32
startdifference=dem_states-rep_states

Arizona=11
Georgia=16
Pennsylvania=20
NorthCarolina=15
Wisconsin=10

swing_states=np.array([Arizona, Georgia, Pennsylvania, NorthCarolina,
                                    Wisconsin])
states=len(swing_states)
resources= 6
status_quo=np.array([-1,-1,1,-1,1])
#create payoff and list
dec = np.array([np.array(i) for i in product(range(resources+1),
                                    repeat=states) if sum(i)==
                                    resources]);
num_actions=len(dec);
payoffT=np.zeros((num_actions,num_actions));
payoffB=np.zeros((num_actions,num_actions));


for i in range (num_actions):
    for j in range(num_actions):
        payoffB[i][j]= np.sign(startdifference+(np.dot(np.sign(
                                        status_quo+ 2*np.sign(dec[
                                        i]-dec[j])),swing_states))
                                        )

        payoffT[i][j]= np.sign(-startdifference+(np.dot(np.sign(2*np.
                                        sign(dec[i]-dec[j])-
                                        status_quo),swing_states))
                                        )

#initialize
regretSum=np.zeros(num_actions);
oppRegretSum=np.zeros(num_actions);

strategySum=np.zeros(num_actions);
oppStrategySum=np.zeros(num_actions);


def getStrategy(regretSum,sSum):
```

```
    normalizingSum=0;
    strategy=np.zeros(num_actions)
    Sum=sSum
    for a in range(num_actions):
     if regretSum[a]>0 :
      strategy[a]=regretSum[a]
     else:
         strategy[a]=0
     normalizingSum += strategy[a]

    for a in range(num_actions):
        if normalizingSum>0 :
            strategy[a] /= normalizingSum
        else:
            strategy[a]= 1/num_actions
        Sum[a]+=strategy[a]
    sSum=Sum
    return(strategy)


def getAction(strategy)  :
    r=random.random()
    a=0
    cumulativeProbability=0
    while a < num_actions-1:
        cumulativeProbability+=strategy[a]
        if r < cumulativeProbability:
            break
        a+=1
    return(a)

def Train(iterations):
    for i in range(iterations+1):
         newStrategy=getStrategy(regretSum, strategySum)
         oppNewStrategy=getStrategy(oppRegretSum, oppStrategySum)
         myAction=getAction(newStrategy)
         oppAction=getAction(oppNewStrategy)
    #get utility
         actionUtility=payoffB[:,oppAction]
         oppActionUtility=payoffT[:,myAction]
    #get regret
         for a in range(num_actions):
          regretSum[a]+=actionUtility[a]-actionUtility[myAction]
          oppRegretSum[a]+=oppActionUtility[a]-oppActionUtility[
                                              oppAction]



def getAverageStrategy(straSum):
    avgStrategy=np.zeros(num_actions)
    normalizingSum=0
    for a in range(num_actions):
        normalizingSum+=straSum[a]
    for a in range(num_actions):
        if normalizingSum >0:
```

```
            avgStrategy[a]=straSum[a]/normalizingSum
        else:
            avgStrategy[a]=1.0/num_actions
    return(avgStrategy)

Train(1000000);
Strategies_table=np.zeros((num_actions,7))
Strategies_table[:,0]=getAverageStrategy(oppStrategySum)
Strategies_table[:,1]=getAverageStrategy(strategySum)
Strategies_table[:,2:7]=dec

print(Strategies_table)
```

8 Appendix C

```
import random
import numpy as np
from itertools import product
#
dem_states=249
rep_states=217

#32
startdifference=dem_states-rep_states

Arizona=11
Georgia=16
Pennsylvania=20
NorthCarolina=15
Wisconsin=10

swing_states=np.array([Arizona, Georgia, Pennsylvania, NorthCarolina,
                                    Wisconsin])
states=len(swing_states)
resources= 6
status_quo=np.array([-1,-1,1,-1,1])
#create payoff and list
dec = np.array([np.array(i) for i in product(range(resources+1),
                                    repeat=states) if sum(i)==
                                    resources]);
num_actions=len(dec);
payoffT=np.zeros((num_actions,num_actions));
payoffB=np.zeros((num_actions,num_actions));


for i in range (num_actions):
    for j in range(num_actions):
        payoffB[i][j]= (startdifference+(np.dot(np.sign(status_quo+ 2
                                    *np.sign(dec[i]-dec[j])),
                                    swing_states)))>0

        payoffT[i][j]= (-startdifference+(np.dot(np.sign(2*np.sign(
                                    dec[i]-dec[j])-status_quo)
                                    ,swing_states)))>0

def getStrategyTable(n):
 #initialize
 regretSum=np.zeros(num_actions);
 oppRegretSum=np.zeros(num_actions);

 strategySum=np.zeros(num_actions);
 oppStrategySum=np.zeros(num_actions);


 def getStrategy(regretSum,sSum):
    normalizingSum=0;
```

```
    strategy=np.zeros(num_actions)
    Sum=sSum
    for a in range(num_actions):
     if regretSum[a]>0 :
      strategy[a]=regretSum[a]
     else:
         strategy[a]=0
     normalizingSum += strategy[a]

    for a in range(num_actions):
        if normalizingSum>0 :
            strategy[a] /= normalizingSum
        else:
            strategy[a]= 1/num_actions
        Sum[a]+=strategy[a]
    sSum=Sum
    return(strategy)


def getAction(strategy)  :
    r=random.random()
    a=0
    cumulativeProbability=0
    while a < num_actions-1:
        cumulativeProbability+=strategy[a]
        if r < cumulativeProbability:
            break
        a+=1
    return(a)

def Train(iterations):
    for i in range(iterations+1):
         newStrategy=getStrategy(regretSum, strategySum)
         oppNewStrategy=getStrategy(oppRegretSum, oppStrategySum)
         myAction=getAction(newStrategy)
         oppAction=getAction(oppNewStrategy)
    #get utility
         actionUtility=payoffB[:,oppAction]
         oppActionUtility=payoffT[:,myAction]
    #get regret
         for a in range(num_actions):
          regretSum[a]+=actionUtility[a]-actionUtility[myAction]
          oppRegretSum[a]+=oppActionUtility[a]-oppActionUtility[
                                                 oppAction]



def getAverageStrategy(straSum):
    avgStrategy=np.zeros(num_actions)
    normalizingSum=0
    for a in range(num_actions):
        normalizingSum+=straSum[a]
    for a in range(num_actions):
        if normalizingSum >0:
            avgStrategy[a]=straSum[a]/normalizingSum
```

```
        else:
            avgStrategy[a]=1.0/num_actions
    return(avgStrategy)

Train(n);
Strategies_table=np.zeros((num_actions,7))
Strategies_table[:,0]=getAverageStrategy(oppStrategySum)
Strategies_table[:,1]=getAverageStrategy(strategySum)
Strategies_table[:,2:7]=dec

return(Strategies_table)
```