Evolution of US Presidential Election Spending Strategies

Kade Heckel
kade.heckel@gmail.com
University of Sussex
Brighton, East Sussex, United Kingdom

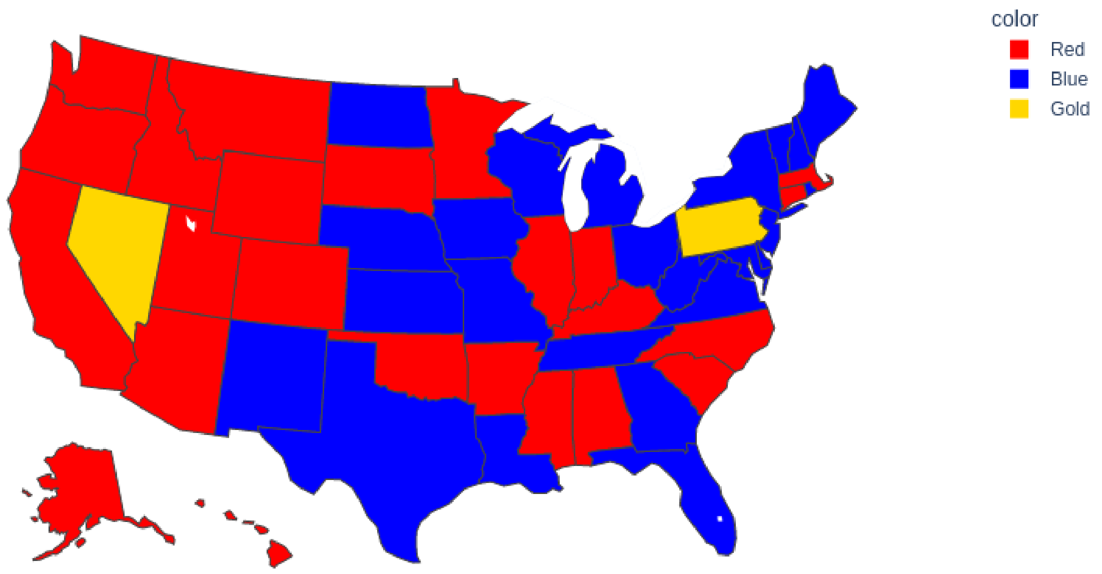

Figure 1. Example electoral results for three candidates to maximize votes received

Abstract

This report investigates campaign finance spending strategies in US presidential elections using an evolutionary game theory approach. Framing an election as a Colonel Blotto game, two and three party campaign strategies for allocating resources from a constrained budget are evolved in a coevolutionary fashion. Results of the simulations show that strategies for complex and continuous games with adapting reward landscapes can be computed using evolutionary game theory approaches. Finally, improvements to the fidelity and realism of the model strategies are suggested in order to increase applicability to real-world scenarios.

Keywords: evolutionary game theory, US presidential elections

1 Introduction

This project studies the dynamics of evolutionary algorithms when acting as adaptive systems within an evolutionary game theory scenario. This section will define and describe the Colonel Blotto game and its implementation in an evolutionary form as well as the evolutionary algorithms which act as adaptive systems to produce high performing strategies.

Introduced by Borel in 1921, the Colonel Blotto game is a century-old classic in game theory and strategic decision making. In a Blotto game, two or more players of equal or differing budgets allocate resources to "battlefields", with the player with the largest commitment winning the value of the battlefield in a winner take all fashion. While framed as a military scenario, the Blotto game has wide applicability to situations such as political elections or roster management in professional sports. This project explores US presidential elections as Blotto games, where political parties allocate resources to campaigning in certain states in the hopes of winning electoral votes. In US presidential elections, candidates compete to win states which each have a number of electoral votes proportional to their population; the candidate to win a simple majority of these electoral votes (270) wins the presidency; otherwise the election is sent to congress to decide the next president.[1]

Since analytical solutions for a multiplayer Blotto game with many battlefields and continuous resource allocation values are difficult to find, an evolutionary game theory approach is used where allocation strategies are evolved.[2] In such a framework each member of a population describes a fraction of the overall budget to spend on each state. This population is then evolved by calculating the number of votes accrued by a strategy when tested against all combinations of other party strategies. Strategies that accrue the highest average number of votes are assigned the highest fitness and become more prolific through selection and reproduction. This paper employs the PGPE algorithm which is population based but acts more like a traditional gradient descent algorithm while not requiring a differentiable loss function; additional details can be found in the appendix under methodology. [3] In summary, the advantage of evolutionary game theory lies in its ease of numerical simulation for complex problems such as the Blotto game for many battlefields and multiple players. [4]

Finally, an adaptive system is one which modifies itself in response to external stimuli and internal measures in order to achieve specific objectives. In the case of the Blotto election game implemented in this work there are three adaptive systems, one for each of the political parties. Each system is composed the respective strategy population and evolutionary algorithm for that political party. Within this paradigm, the specific valuations of states can be thought of as adaptations made to accrue votes, while the evolutionary selection of robust strategies is the adaptive process which causes the adaptations to change over time. These adaptations such as switching from investing in one state to another are driven by external stimulus in the form of competition when parties attempt to push each other out of certain states to gain votes.

2 Results

The presented data tracks direct competition between the mean strategies and the "best" strategies based on average votes accrued. The "best" strategies are the allocations which obtained the highest number of electoral votes when competing against all combinations of opposing strategies; these strategies do not update continuously and when they do may change in a deleterious fashion since there is a disconnect between the adaptation rates of the best strategies. Whereas comparison of the mean strategies results in smooth and gradual fluctuations in spending, the best/high fitness strategy comparison change abruptly. This is caused by the fact that the mean strategy changes every epoch of the evolutionary process while the best strategy only updates when a member beats the highest average votes accrued by the previous best strategy. The figures in this report illustrate the mean strategy behavior, while additional figures for the best strategy dynamics can be found in the associated Python Jupyter notebook.

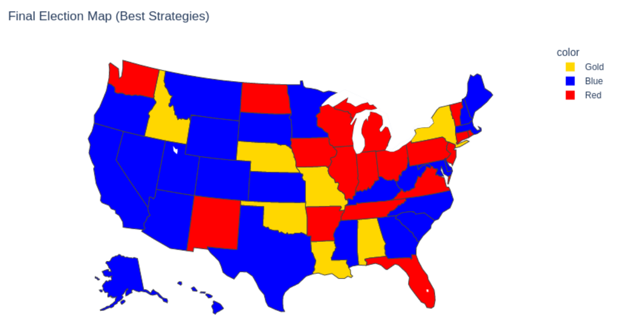

Figure 2. Final electoral results for best strategies by average votes accrued.

The gold strategy acted as a third party candidate, with their allocations initially suppressed to allow the red and blue strategies to stabilize. At epoch 500 the gold budget was modified from 0 to 0.75 in an attempt to disturb the existing red and blue strategies, however the minimal adjustment in the blue and red strategies suggests that a larger disturbance over time is needed.

2.1 Final Election Maps

Figures 2 and 3 depict the results of a simulated election between the final best and mean strategies evolved over 1500 epochs. Note that for the highest performing strategies for each party blue barely wins with the high value states being split between major blue and red, while red dominates the mean strategies with wins in all of the major states.

2.2 Evolution of Vote Counts over Time

Figure 4 shows how the votes accrued by the mean and best strategies change over time and in relation to each other. Gains by one strategy are correlated with losses by one or both of the other parties as the election is zero sum. Note that the mean strategy fluctuates with every epoch whereas the best strategies are stagnant for long periods before shifting dramatically. Figure 5 charts the electoral loss over time that each adaptive system seeks to minimize, with the "best" strategy loss remaining fairly stagnant over time while the mean loss is more dynamic and changes smoothly over time.

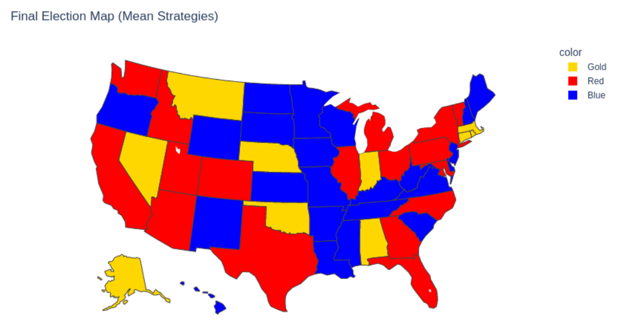

Figure 3. Electoral results between mean strategies for each political party. Note red dominates with wins in California, Texas, Florida, New York, and Pennsylvania.

Figure 9 highlights the adaptive properties of each evolving strategy population as states flip back and forth from red to blue as each party adjusts strategies. Specifically the red allocation over time displays adaptation to opposing strategies as red initially over invests in California and Texas before redistributing funding to other states.

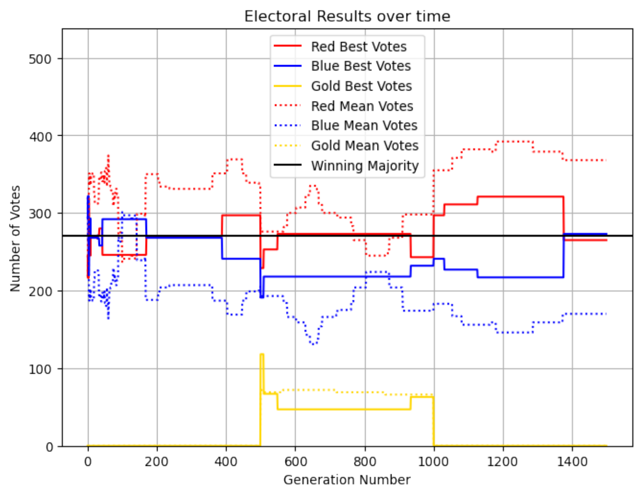

Figure 4. The evolution of electoral strategies over time and their relative performance. Note that the mean strategies for each algorithm have a wider performance gap than the "best" strategies.

The adaptive nature of these coupled systems is demonstrated in figures 6, 7, and 8. While the mean blue strategy remained more stable over time with smaller fluctuations, the mean red strategy adjusts significantly to improve its electoral returns, prioritizing high value states such as California, Texas, Florida, New York, and Pennsylvania, which all possess significantly more votes than other states due to having larger population sizes.

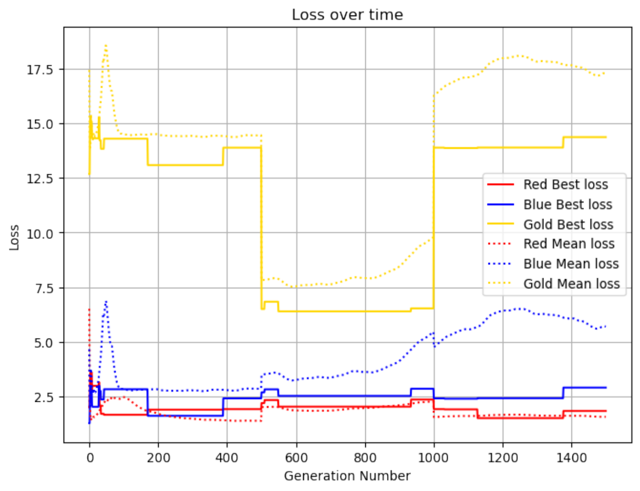

Figure 5. The task-specific loss values for each party and strategy type over time. The gradual rise in loss for the blue mean strategy correlates with the drop in accrued votes from epoch 800 onward.

2.3 Winning Spending

Figures 10 and 11 illustrate the winning spending amounts per state. Note that a strategy that evenly allocates resources across all 50 states and Washington, D.C. would assign 1.96% of resources to each state. Thus, states that were won with a value less than 0.0196 can be thought of as a bargain while values approaching 0.04 can be seen as valued the same as two other states.

3 Discussion and Future Work

3.1 Analysis

Based on the winning expenditures plots, tracking the mean strategy for each party over time provides a more realistic and adaptive model that targets states with large populations compared to tracking the best strategy which transitions less often and can overvalue less relevant states such as the red "best" strategy wasting resources on North Dakota for only a 3 electoral vote return. The determination fitness by testing each strategy against every combination of opposing strategies is computationally costly at $O(N^3)$ where N is the population size; simply by randomly matching a set of strategies from each party and performing one evaluation each, the computational complexity would only be $O(N)$ and would be much more efficient and allow for larger population sizes. The asymmetric advantage gained by red is interesting and perhaps a symptom of formulating the game as winning as many votes as possible rather than trying to consistently achieve the winning threshold of 270.

One behavior that was not demonstrated to the anticipated degree was oscillatory behavior between which party accrued the most votes. Over many different random seeds it

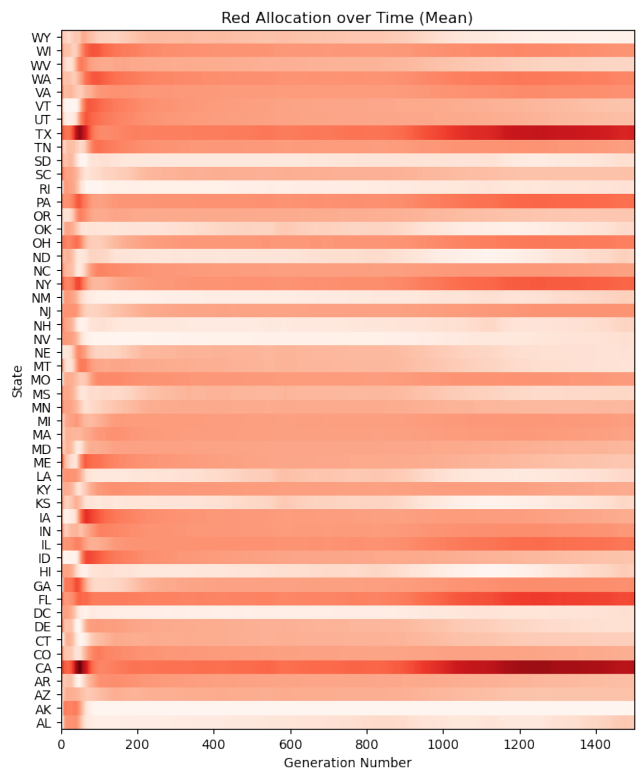

Figure 6. Resource allocations for the mean red strategy over time. Note the dark bands in CA and TX as well as darker shades for other high value states. The early shift in allocations in the first 50 epochs correlates with an over investment with large states resulting in blue gaining many votes before red returns to a more balanced distribution. This is evidence that red in adapted itself to regain performance, a trait of an adaptive system.

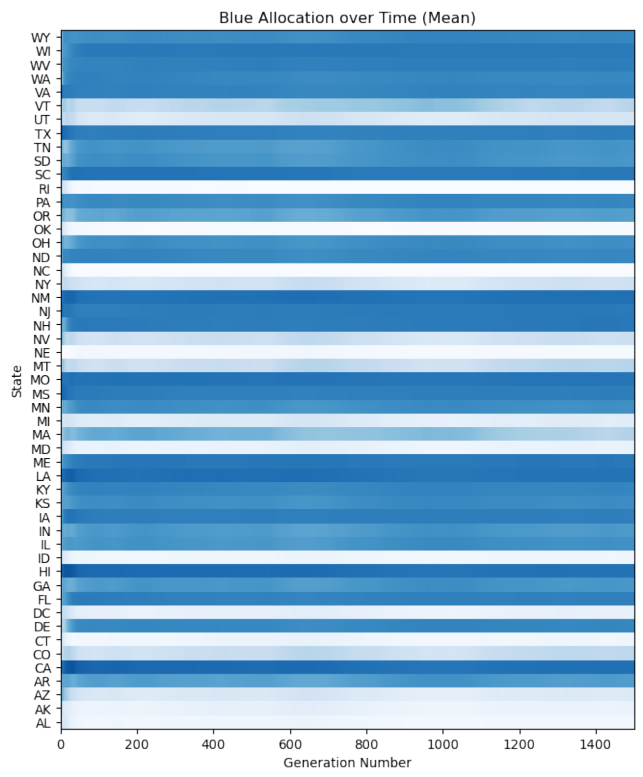

Figure 7. Blue mean strategy resource allocations over time. Note the lack of major shifts and gradual lightening between epochs 600 and 800, indicating that introduction of the gold party caused blue adapt and divert resources to other states to gain more votes; after the gold party is removed at epoch 1,000 the blue mean strategy returns to its previous distribution.

seemed that one party would stabilize as the winner, contrary to the thought that an evolutionary arms race would emerge. Additionally, it can be concluded that the emergent strategies are resilient in that they are not easily perturbed by newly introduced players such as the gold party. One reason the strategies may be so stable is that the current formulation of the loss function places a heavier penalty on abandoning a small value state than barely losing it and therefore strategies are encouraged to distribute their resources around all states. The adaptation of budget levels was attempted but found to be unstable as the values would trend to infinity due to a lack of constraints on budget growth.

In conclusion, tracking the mean strategies for each party as they adapt over time rather than an elitist approach allows for more realism in modeling elections. Observing the mean strategy demonstrates the dynamics of the political party strategies as coupled adaptive systems and the changes in spending on high value states over time demonstrates the adaptive nature of the systems. The use of the gold party as a disturbance resulted in the red and blue parties reallocating resources for the disturbance duration, and future work can further explore the impact of a surprise party for varying duration and budget levels.

3.2 Next Steps

This work formulates elections as a competition to receive the most votes while in reality the objective is to accrue a simple majority of 270. Modifying the structure of the reward to this paradigm could improve the model by promoting more sparsity and concentration of investment. For example, winning a state such as California with it's 55 electoral votes is far more important than winning Montana, Hawaii, Alaska, and North Dakota which combine for a mere 12 electoral votes.

Aside from computational considerations for optimization, future work can seek to increase the realism of the model through two methods: accounting each state's population, voter registration, and their party affiliation and allowing for

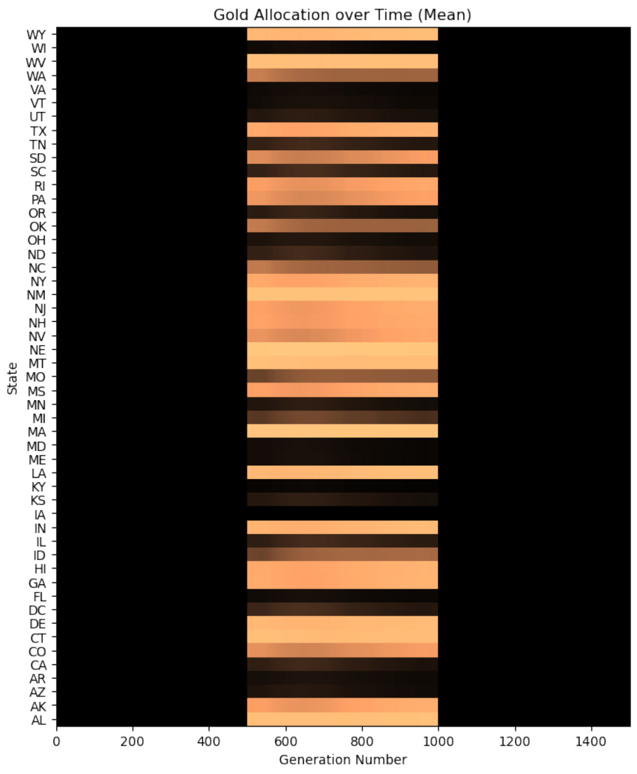

Figure 8. Gold mean strategy resource allocation. Note that while running for 500 epochs there are only small variations as the red and blue strategies have fixed on highly robust configurations.

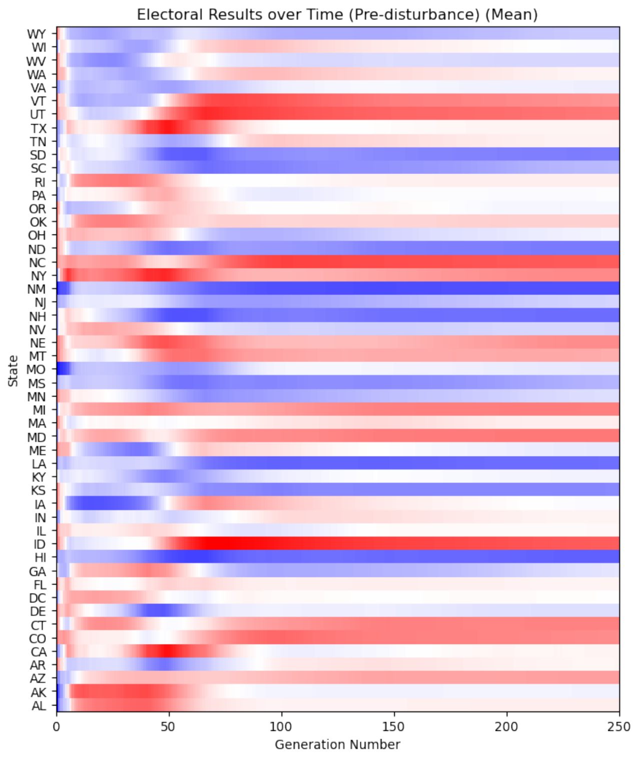

Figure 9. The first 250 epochs of red and blue strategy competition. Observe the dynamics as red and blue battle for certain states before settling into more evenly distributed allocation strategies.

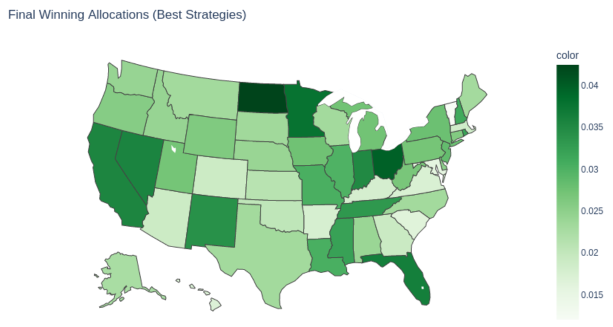

Figure 10. Winning resource allocations after 1,500 epochs for strategies which average the highest vote count. North Dakota receives an abnormally large investment despite only possessing 3 electoral votes, indicating inefficiency.

different types of resource allocations and modelling their effectiveness across different demographics.

Currently the model assumes that all voters in a state can be persuaded to vote for one candidate or the other, and that spending money on advertising/campaigning is directly tied to this persuasion. This is not the case in reality, where American political parties see significant party loyalty and campaign spending is mainly targeted at either persuading independents and undecided voters as well as energizing the candidate's political base to increase voter turnout. A more realistic model would account for the demographics of each state and their political affiliations when simulating elections. For example, responses to a survey of 12,251 Californians yielded 46.8% Democratic/blue responses, 23.9% Republican/red responses, and 22.7% independent responses; for Republicans to have a chance at winning the state they would need to either win over almost all independent voters and/or peel off votes from moderate/centrist Democratic voters.[5] Since the current political climate in California as an example would make it very difficult for a Republican to win, an evolved Republican campaign strategy should account for this and allocate resources elsewhere where they could provide better benefits. This would also increase the fidelity of the model as it would drive behavior associated with so-called "battleground" states where campaigns spend disproportionate amounts of their funding trying to win several key states where margins between the major parties are narrow.

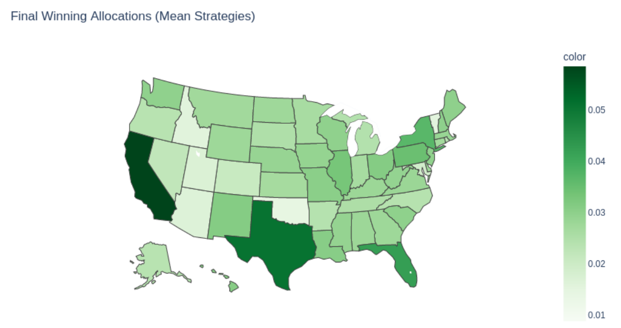

Figure 11. Winning resource allocations after 1,500 epochs for the mean evolved strategies of each party. In better relation to reality, the amount of resources required to win a state appears correlated with the number of electoral votes the state has.

An additional aspect which could be incorporated into the model would be allowing for different types of spending/resource allocations, such as TV advertising, internet advertising, canvassing, voter registration drives, and visits/rallies. By matching these events with weights for how much they appeal to different demographics, the complexity of the model would increase as the evolved strategies would describe not just what states to campaign in but also what techniques to use in the pursuit of maximizing electoral returns.

References

[1] Soheil Behnezhad, Avrim Blum, Mahsa Derakhshan, Mohammadtaghi Hajiaghayi, Christos H. Papadimitriou, and Saeed Seddighin. Optimal strategies of blotto games. *Proceedings of the 2019 ACM Conference on Economics and Computation*, Jun 2019.

[2] J. McKenzie Alexander. Evolutionary game theory, Apr 2021.

[3] Frank Sehnke, Christian Osendorfer, Thomas Rückstieß, Alex Graves, Jan Peters, and Jürgen Schmidhuber. Parameter-exploring policy gradients. *Neural Networks*, 23(4):551–559, 2010. The 18th International Conference on Artificial Neural Networks, ICANN 2008.

[4] Aymeric Vié. A genetic algorithm approach to asymmetrical blotto games with heterogeneous valuations. *CoRR*, abs/2103.14372, 2021.

[5] Mark Baldassare, Dean Bonner, Rachel Lawler, and Deja Thomas. California voter and party profiles, Jan 2023.

[6] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[7] Robert Tjarko Lange. evosax: Jax-based evolution strategies. *arXiv preprint arXiv:2212.04180*, 2022.

[8] Plotly Technologies Inc. Collaborative data science, 2015.

[9] United states electoral college votes by state.

A Methodology

To implement the evolutionary system, the JAX [6] and EvoSax [7] python libraries were used to leverage GPU acceleration for highly efficient evolution runs. JAX is a high-performance array computing package that provides automatic differentiation capabilities and a domain-specific compiler to optimized accelerator kernels through a familiar NumPy-style API. EvoSax[7] is an evolution strategies library built on top of JAX, implementing a wide variety of evolution strategies algorithms for use in numerical computing and neuroevolution. The Plotly library [8] was used to facilitate easy plotting of choropleth maps depicting the resource allocation of spending strategies as well as visualizing electoral maps showing which states each candidate won in the simulated election. The electoral vote counts per state were based on the 2020 US presidential election values. [9]

Following the design of a Colonel Blotto game, an election is implemented by determining the candidate who allocated the most resources to a state and assigning them the number of votes corresponding to that state while the others receive no votes. This can be efficiently implemented over vector representations by computing the argument max over the stacked allocation strategy vectors and then taking the dot product between the vector of each candidate's victories and a vector encoding the electoral votes of each state.

The Parameter Exploring Policy Gradients algorithm from [3] was employed to evolve allocation strategies for each party. PGPE maintains a mean strategy and a standard deviation parameter, from which a population of sample strategies are generated. Each generated strategy then has a mirror strategy generated of itself by reflecting across the mean strategy. These pairs are then evaluated against all combinations of the other parties' populations to calculate a loss value based on how close different states were won or lost. These loss values are used as a surrogate for the actual gradient of the loss function with which the mean for the next generation is calculated by using stochastic gradient descent. This method allows for easy optimization of non-differentiable loss functions while leveraging high-performing second order gradient descent algorithms such as the Adam optimizer. For the task at hand, the potential strategies for each of the three candidates are encoded as 100-member populations of 51 dimensional vectors, corresponding to the 50 US states and the District of Columbia. The simulation initializes the populations for each candidate and then begins the iterative ask-evaluate-tell evolution process. The PGPE algorithm is explained through Python psuedocode in listing 1.

The fitness/performance of a strategy is formulated as a loss function intended to provide a smoother loss landscape for PGPE to explore. Instead of all-or-nothing returns corresponding to the electoral vote count of a state, the winner of the state is penalized proportional to the cost per electoral vote while the loser of a state is penalized proportional to the number of electoral votes times the difference between their allocation and the winning allocation. The minimization of this loss function attempts to drive the evolution of strategies which win the most states by the narrowest of margins. Mathematically the electoral loss is roughly defined as follows:

$$allocations = [R, B, G]$$

$$winners = argmax_i(allocations)$$

$$\mathcal{L}_election = sum((max_{state} - allocations) * electoral_votes + (allocai$$

The ask-evaluate-tell loop is implemented as a function that can then be directly compiled to a GPU kernel using the jax.lax.scan() method, which in tandem with the just-in-time compilation by JAX of the overall evolution experiment function enables simulations to be run in seconds.[6] The overall structure of the simulation is detailed in listing 2.

The evaluation of each strategy is carried out by vectorized mapping of a single comparison function across all three populations. Through this approach each strategy in each population is evaluated against all possible combinations of opposing strategies with the average number of votes accrued determining the fitness. Concretely, if the strategy populations are represented as $r \in R, b \in B$, and $g \in G$ and the votes accrued by each strategy triplet (r, g, b) is calculated by the *election* function, then the vectorized mapping process applies $election(r, b, g)$ such that $election(r_i, b_i, g_i)$ for all strategies in R, B, G. This results in promoting strategies which are robust to a variety of opposition strategies.

B Additional Figures

B.1 Final Spending Strategies per Color

These figures provide a breakdown of party spending levels per state, illustrating how the mean strategies adapted to target high value states whereas best performing member strategies remained fixated on low value states worth few electoral votes.

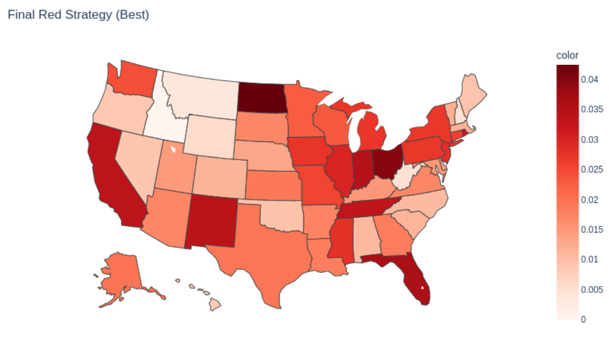

Figure 12. Final "best" spending distribution for red. Note the extremely high allocation for North Dakota, a low value state.

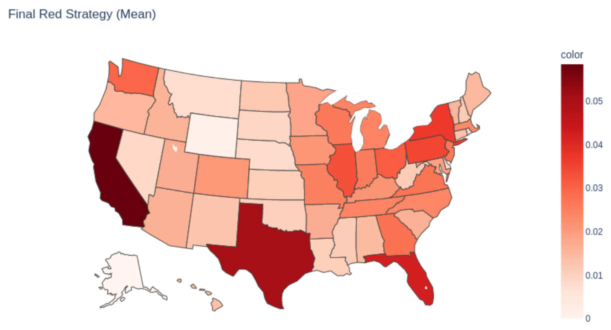

Figure 13. Final mean spending distribution for red. Observe priortization of high value states like California and Texas.

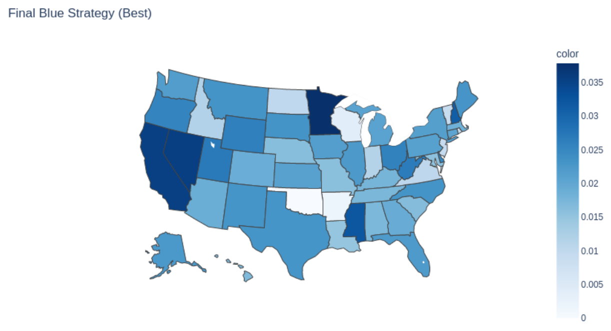

Figure 14. Final "best" spending distribution for blue. Note the emphasis on Minnesota which only has 10 electoral votes compared to other states such as Pennsylvania which has 20.

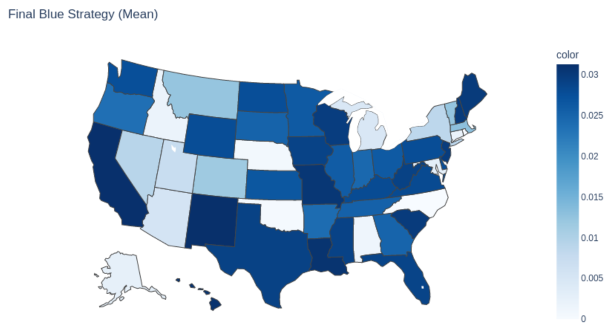

Figure 15. Final mean spending distribution for blue. The relatively even distribution of resources is interesting.

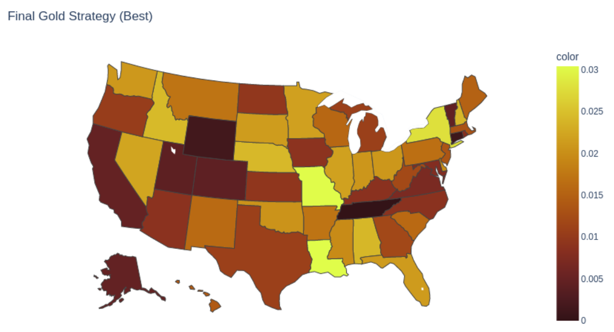

Figure 16. Final "best" spending strategy for gold. Note the slightly lower peak allocations due to the gold being constrained to 75% of the red and blue budgets.

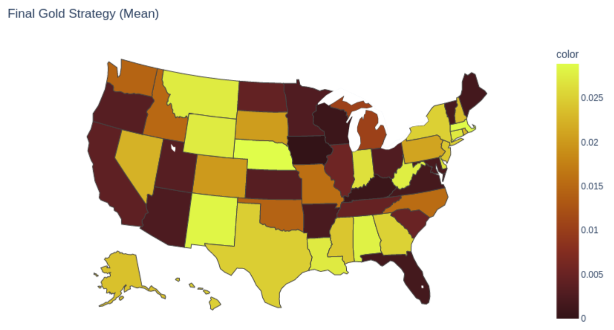

Figure 17. Final mean spending distribution for gold. Comparing with figure 2, it can be seen that gold successfully steals New York from the other two states.

B.2 Code Listings

Note that these psuedocode listings are not executable despite being in a python like syntax. Please use the Jupyter notebook to reproduce experiments. These listings are written in an object-oriented style while JAX is functional and requires explicitly passing state between function calls and not having code that produces side effects.

B.3 Listing 1

```
# This psuedocode describes the high level logic of the EvoSax PGPE implementation

class PGPE:
    def __init__(self, popsize, num_dims):
        self.population_size = popsize
        self.num_dimensions = num_dims
        self.mean = random.uniform(num_dims)
        self.sigma = 1
        self.optimizer = StochasticGradientDescentFunction

    def ask():
        positive_samples = random.normal(population_size / 2,
                                         num_dimensions)
        mirrors = -positive_samples
        samples = concatenate(positive_samples, mirrors)
        strategies = self.mean + self.sigma * samples
        return strategies

    # Treats fitness as a loss function to minimize
    def tell(strategies, fitness):
        initial_samples = (strategies - self.mean) / self.sigma
        direction_vectors = initial_samples[:self.population_size/2]
        direction_vectors -= initial_samples[self.population_size/2:]

        initial_samples_fitness = strategies[:self.population_size/2]
        mirror_fitness = strategies[self.population_size/2:]

        fitness_deltas = initial_sample_fitness - mirror_fitness
        fitness_deltas = direction_vectors * fitness_deltas
        gradient = 1.0 / self.population_size * fitness_deltas
        self.mean = self.optimizer.step(mean, gradient)
```

B.4 Listing 2

```
# Python psuedocode of ask-eval-tell evolution loop
# full version in accompanying code is implemented in JAX
# and follows a functional programming paradigm.

RED_BUDGET  = 1
BLUE_BUDGET = 1
GOLD_BUDGET = .75
NOISE = 0.03

def election(R, B, G):
    allocations = vstack([R, B, G])
    winners = jnp.argmax(allocations, axis=0)
    loss = (max(allocations, axis=0) - allocations) * electoral_votes
    loss = where(loss == 0,
                 (allocations-mean(allocations, axis=0))/electoral_votes, loss)
    loss = sum(loss, axis=-1)
    red_loss, blue_loss, gold_loss = loss[0], loss[1], loss[2]
    red_votes = dot(where(winners==0, 1, 0), electoral_votes)
    blue_votes = dot(where(winners==1, 1, 0), electoral_votes)
    gold_votes = dot(where(winners==2, 1, 0), electoral_votes)
    return red_votes, blue_votes, gold_votes, winners, rloss, bloss, gloss

sim_election = vmap(vmap(vmap(election, (None, None, 0)),
                                        (None, 0, None)),
                                        (0, None, None))

def blotto(red_pop, blue_pop, gold_pop):
    red_results, blue_results, gold_results, _, \
    rloss, bloss, gloss = sim_election(red_pop, blue_pop, gold_pop)
    red_fitness = mean(rloss, axis=(-1, -2))
    blue_fitness = mean(bloss, axis=(-1, -2))
    gold_fitness = mean(gloss, axis=(-1, -2))
    return red_fitness, blue_fitness, gold_fitness

def allocate_funds(strats, budeget):
    min_spend = min(strats, axis=-1)
    max_spend = max(strats, axis=-1)
    norm_strats = (strats - min_spend) / (max_spend-min_spend)
    return (norm_strats - sum(norm_strats, axis=-1)) * budget

def gold_step(epoch_num):
    return epoch_num // 500 % 2

def evolve(num_epochs):
    red, blue, gold = PGPE(popsize=100, num_dims=51), \
                      PGPE(popsize=100, num_dims=51), \
                      PGPE(popsize=100, num_dims=51)

    def epoch():
        red_pop, blue_pop, gold_pop = red.ask(red_state), \
                                      blue.ask(blue_state), \
                                      gold.ask(gold_state)

        red_alloc, \
        blue_alloc, \
        gold_alloc = allocate_funds(red_pop, RED_BUDGET)
                     allocate_funds(blue_pop, BLUE_BUDGET)
                     allocate_funds(gold_pop, gold_step(epoch_num)*GOLD_BUDGET)
        red_loss, blue_loss, gold_loss = blotto(red_alloc, blue_alloc, gold_alloc)

        red.tell(red_alloc, red_loss)
        blue.tell(blue_alloc, blue_loss)
        gold.tell(gold_alloc, gold_loss)

    final_strategies, scan_steps = scan(epoch)
    return final_strategies, scan_steps
```