The Thirty-First International Florida
Artificial Intelligence Research Society Conference (FLAIRS-31)

Improved Manipulation Algorithms for District-Based Elections

Ramoni O. Lasisi
Department of Computer and Information Sciences
Virginia Military Institute
LasisiRO@vmi.edu

Abstract

District-based elections where voters vote for a district representative and those representatives in turn vote to determine the overall winner are vulnerable to a manipulation called *gerrymandering*. Gerrymandering occurs when the outcome of a district-based election is manipulated by changing the locations and/or borders of districts in the election. A recent work shows that the problem of gerrymandering in district-based election is NP-complete. This previous work also proposed a manipulation algorithm that is polynomial in the parameters (number of voters, candidates, and districts) of the election. However, the algorithm suffers from a high running time. We propose in this work, three improved manipulation algorithms for this problem. We then show that the three algorithms are also polynomial in these parameters, albeit, with lower running times compared to the previous work.

1 Introduction

Voting protocols are commonly used for preferences aggregation. Bartholdi, Tovey, and Trick (1989) define a voting protocol as an algorithm that takes as input a set C of candidates and a set P of preferences that are *strict* (irreflexive and antisymmetric), *transitive*, and *complete* on C. The algorithm outputs a subset of C (allowing for ties), who are the winners. Voting protocols, including *plurality, Borda, Copeland, veto,* and *sequential runoff*, are some widely studied voting schemes. See for example (Faliszewski, Hemaspaandra, and Schnoor 2010; Lasisi 2016; Lasisi and Lasisi 2017).

Unlike the above voting schemes where elections are completed in a single stage, *district-based elections* that we study in this work are conducted in two stages. In a district-based election, *voters vote for a district representative and those representatives (from each of the districts) further vote to determine the overall winner of the election*. District-based elections have real-life applications ranging from human societies, artificial intelligence, to multi-agent systems. For example, the Electoral College of the United States for electing the US president uses the district-based election scheme where voters in the general elections elect members of the Electoral College who afterwards vote to determine the next president. Also in many companies, shareholders vote at annual general meetings to elect board of directors, the board then vote amongst her members to elect a chair for the board. Furthermore, consider a multi-sensor network environment where several sensor agents jointly decide on strategies to track targets in their fields of view. Depending on the complexity of the networks (e.g, several sub networks) and target types (e.g., multiple or moving targets), decisions taken by the independent sub networks may then be aggregated to decide the overall best strategy for the network. The sensor agents model the voters while the sub networks model districts in a district-based election.

The ideal of a society is that a candidate emerging as a winner in an election be as widely and socially acceptable as possible. As useful and widely applicable as the district-based election scheme is, it is not immune from the vulnerability of manipulation in the election process. In particular, this scheme is vulnerable to a type of manipulation referred to as *gerrymandering*. Gerrymandering *occurs when the outcome of a district-based election is manipulated by changing the locations and/or borders of districts in the election*. Gerrymandering has been widely studied from different areas of research, including political science (Feix et al. 2004; Miller 2014), history (Butler 1992; Engstrom 2006), and the social choice community (Guillermo and Bernard 1988; Bachrach et al. 2016).

A recent work (Lewenberg, Lev, and Rosenschein 2017) considers the computational complexity of gerrymandering while using the plurality protocol in district-based elections. The authors show that the problem of gerrymandering in district-based elections is NP-complete in the worst case. However, they propose a *greedy manipulation algorithm*, referred to as *Greedy Gerrymandering*$_{plurality}$, for the problem. The algorithm was used to uncover collections of districts in simulated and real-world elections data (from the 2015 Israeli and UK elections) to demonstrate how gerrymandering could affect elections outcomes. The work thus shows that there are instances of the elections that may be manipulated using the algorithm. Although the proposed algorithm is polynomial in the parameters (number of voters, candidates, and districts) of the election, it suffers from its high running time. This running time may become a source of concern, especially when the parameters are large.

We propose in this work three improved manipulation algorithms for district-based elections. These algorithms together advance the state of the art by extending a recent work of Lewenberg, Lev, and Rosenschein (2017). Our first algorithm corrects an inefficiency in the Greedy Gerrymandering$_{plurality}$ algorithm that is due to an expensive computation which is not necessary. The second and third algorithms are based on *dynamic programming* and *randomization* techniques, respectively.

We show that the three algorithms are polynomial in the parameters of the district-based elections, albeit, with lower running times compared to the Greedy Gerrymandering$_{plurality}$ algorithm. For the case of the randomized algorithm, we also account for the number of samples required for a given accuracy and the probability of missing the accurate value of the number of district ballot boxes for a target candidate to achieve plurality of the ballot boxes. The implication of these new results is that although gerrymandering in district-based elections using the plurality voting protocol is NP-complete in the worst case, nonetheless, it may be achieved with some instances of the district-based elections with little computational efforts.

2 Preliminaries

Definitions and Notation

Let $k, l, m, n, z \in \mathbb{N}$. Let $C = \{c_1, \ldots, c_z\}$ be a set of z candidates in an election. Let $V = \{v_1, \ldots, v_n\}$ be a set of n voters. Let $\pi(C)$ be the set of preference orders over C. Thus, $\pi(C)^n = \{\pi_1, \ldots, \pi_n\}$, defines the preference orders of V over C. We define a relation, $\succ$, for each π_i. We say that a voter $v_m \in V$ ranks candidate c_i over candidate c_j denoted, $c_i \succ c_j$, if v_m prefers c_i to c_j in her preference order π_m.

Definition 1. *Voting Rule*

A voting rule $f : \pi(C)^n \rightarrow C$ is a function that maps the voters' preferences to candidates in an election.

Definition 2. *Plurality Voting*

Plurality is a voting rule in which each voter casts one vote for her most preferred candidate. It is only the first choice candidate of the voters' preferences that are considered in determining the winner. This scheme requires a voter to indicate only her first choice candidate and not the entire order.

Definition 3. *District-based Election*

A district-based election involving voters V and candidates C consists of a partition of V into m disjoint districts $V_1, \ldots, V_m$, such that $V = \bigcup_{i=1}^{m} V_i$, with each district V_i having a ballot box b_i. Let $B = \{b_1, \ldots, b_m\}$ be the set of the ballot boxes in all districts. An election is conducted by applying a voting rule to each of the districts. A candidate who wins the highest number of districts is the winner. We assume the existence of some tie-breaking rules. We employ the plurality voting rule in this paper since the work we extend and compare our results to also used plurality.

Definition 4. *(Lewenberg, Lev, and Rosenschein 2017) The Gerrymandering$_{plurality}$ Problem*

The Gerrymandering$_{plurality}$ is a district-based election with additional parameters l and k, such that $l \leq k \leq m$ and a target candidate $p \in C$. We are asked whether there is a subset of k ballot boxes $B' \subset B$, such that they define a district-based election, in which every voter votes at their closest ballot box in B', the winner at every ballot box is determined by plurality, and p wins in at least l ballot boxes.

As noted in (Lewenberg, Lev, and Rosenschein 2017) and upon which the authors' greedy algorithm is based on, Definition 4 is equivalent to "…find[ing] a partition to k districts such that p wins plurality of districts …" We adopt this equivalency too in designing our proposed algorithms.

Definition 5. *Problem Inputs and the $plurality_b^c$ Procedure*

We restate the inputs to the problem we attempt to address in this paper. We are given sets V of n voters, C of z candidates, B of m ballot boxes in m districts V_i, a target candidate $p \in C$, and a constant k.

Also, we define a procedure called $plurality_b^c$ that will be used in the three proposed gerrymandering manipulation algorithms for the district-based elections. $plurality_b^c$ is used to check whether a candidate $c \in C$ wins a plurality election in a district with ballot box $b \in B$:

$$plurality_b^c = \begin{cases} 1 & \text{if } c \text{ wins in a district with a ballot box } b \\ 0 & \text{otherwise} \end{cases}$$

Finally, it is widely known that determining a winner in a plurality election with n voters and z candidates takes $O(z+n)$ time. In our case, plurality elections are conducted in each district (with voters V_i). Without loss of generality, we assume that there are equal number of voters in each district. This assumption has also been used elsewhere in the literature. See for example (Bachrach et al. 2016). Let the number of voters in each district i.e, $|V_i|$ be a. Since the same set C of z candidates are featured in each of the districts, then, $plurality_b^c$ for any candidate c in a district with a ballot box b takes $O(z + a)$ time to compute.

3 Greedy Gerrymandering$_{plurality}$ Algorithm

The Greedy Gerrymandering$_{plurality}$ algorithm of Lewenberg, Lev, and Rosenschein forms the benchmark for our work, so we reproduce it here. The pseudocode is shown in Algorithm 1. We also provide an analysis of the algorithm[1].

Greedy Gerrymandering$_{plurality}$ starts by setting B' to set B of the original ballot boxes in the district-based election. The algorithm then continuously removes ballot boxes b from B' one at a time until $|B'| = k$. The objective of the algorithm at every elimination step is to maximize the ratio between the number of ballot boxes won by the target candidate p to that of the number of ballot boxes won by any other candidate $c \in C$.

Analysis of Greedy Gerrymandering$_{plurality}$

Theorem 1. *Greedy Gerrymandering$_{plurality}$ algorithm runs in $O(zm \cdot (z + a) \cdot (m - k + 1)^2)$ time.*

Proof. We consider the $FindRatio$ procedure first. Clearly from the while loop, $|B'|$ is at most m since $|B| = m$ originally. Finding a plurality winner in the numerator or denominator of the return statement in line 18 takes $O(z + a)$ time. However, the denominator of the statement computes

[1] This analysis was not given in Lewenberg et al.

Algorithm 1 : $GreedyGerrymandering_{plurality}$

1: **procedure** GREEDYGERRYMANDERING(V, B, k, p)
2: $B' \leftarrow B$
3: **while** $|B'| > k$ **do**
4: **for all** $b \in B'$ **do**
5: $f_b \leftarrow FINDRATIO(B', b, V, p)$
6: **end for**
7: $b \leftarrow \arg\max_{b \in B'} \{f_b\}$
8: $B' \leftarrow B' \setminus \{b\}$
9: **end while**
10: **if** p wins a plurality of ballot boxes **then**
11: **return** True
12: **else**
13: **return** False
14: **end if**
15: **end procedure**
16: **procedure** FINDRATIO(B, b, V, p)
17: $B' = B \setminus \{b\}$
18: **return** $\frac{|\{\tilde{b} \in B' : \text{p wins in } \tilde{b}\}|}{\max_{c \in C, c \neq p} |\{\tilde{b} \in B' : \text{c wins in } \tilde{b}\}|}$
19: **end procedure**

the maximum number of wins among all the z candidates (except p) in a plurality election for all ballot boxes $\tilde{b} \in B'$. Thus, $FindRatio$ takes a total of $O(zm \cdot (z + a))$ time.

Now for the Greedy Gerrymandering$_{plurality}$ procedure, it is clear that each of the loops starting from lines $3-4$ takes $O(m - k + 1)$ time since the while loop in line 3 terminates when $|B'| = k$. Also, $FindRatio$ is called each time in the nested loops in lines $3 - 4$. Thus, the overall running time of the Greedy Gerrymandering$_{plurality}$ algorithm is $O(zm \cdot (z + a) \cdot (m - k + 1)^2)$. $\square$

4 The GreedyGerrymandering$+_{plurality}$

We present $GreedyGerrymandering+_{plurality}$ algorithm that removes an expensive computation that is not necessary in the $FindRatio$ procedure of Algorithm 1. $FindRatio$ is called for every ballot box $b \in B'$, and for each of these calls, $FindRatio$ recomputes the number of plurality wins for each candidates $c \in C$ in the set B' of ballot boxes under consideration. We make the following observations:

- The set B' of ballot boxes examined in the $FindRatio$ procedure at some step s differs from that at the next step $s + 1$ by a single ballot box, where $1 \leq s \leq |B'| - 1$.
- Since each ballot box induces a district, the plurality winner, say, $c \in C$ in a particular district V_i with a ballot box, say, b is the same across all steps $1 \leq s \leq |B'|$ of the $FindRatio$ procedure for district V_i.

These two observations lead us to avoid recomputation of the plurality winners in $FindRatio$ for every call from the Greedy Gerrymandering$_{plurality}$ procedure. Rather, we compute the plurality winners in the original ballot boxes B once and then update as appropriate. Using $plurality_b^c$ with $c \in C$, we partition B into disjoint sets of ballot boxes won by each candidate. This partition can be maintained in a HashMap data structure with the candidates and their corresponding sets of ballot boxes won as a key/value pair. Each disjoint set can be maintained by a HashSet data structure. E.g., we can use the following structures in Java:

```
Map<Candidate, Set<BallotBoxes>> partition
Set<BallotBoxes> ballotBoxes
```

The key/value pairs of information stored by these structures can be represented in table form using some arbitrary candidates and ballot boxes as shown in Table 1.

Table 1: Arbitrary key/value pairs of candidates and their sets of plurality ballot boxes won represented in a table form

Candidates	Ballot Boxes Won
c_1	$\{b_2, b_3, b_6\}$
c_2	$\{b_4, b_5, b_7, b_8, b_{11}\}$
p	$\{b_1, b_9, b_{10}\}$

Let procedure $CreatePartition(V, C, B)$ be an algorithm that creates this partition as shown in Algorithm 2.

Algorithm 2 : $CreatePartition$

1: **procedure** $CreatePartition(V, C, B)$
2: $Map < C, Set < B >> partition$
3: $Set < B > ballotBoxes$
4: **for all** $c \in C$ **do**
5: $ballotBoxes \leftarrow \emptyset$
6: **for all** $b \in B$ **do**
7: **if** $plurality_b^c = 1$ **then**
8: $ballotBoxes.add(b)$
9: **end if**
10: **end for**
11: $partition.put(c, ballotBoxes)$
12: **end for**
13: **return** $partition$
14: **end procedure**

Lines 2 and 3 of Algorithm 2 create two data structures, a HashMap, $partition$, and a HashSet, $ballotBoxes$. In lines 4 to 12, for each candidate $c \in C$, we determine the plurality ballot boxes that c wins and stores c and her corresponding set of plurality ballot boxes ($ballotBoxes$) won as a key/value pair in $partition$ at line 11.

We now define a different method for computing the ratio between the number of ballot boxes won by the target candidate p to that of the maximum number of ballot boxes won by any other candidate $c \in C$. The method is illustrated in procedure $ComputeRatio(partition, C, b, p)$ as shown in Algorithm 3 and a description provided in Lemma 1.

Algorithm 3 : $ComputeRatio$

```
procedure ComputeRatio(partition, C, b, p)
        pBallotBoxes ← partition.getValue(p)
        pBallotSize ← pBallotBoxes.size()
        if pBallotBoxes.contains(b) then
            pBallotSize ← pBallotSize − 1
        end if
        maxPluralityWins ← 0
        for all c ∈ C and c ≠ p do
            ballotBoxes ← partition.getValue(c)
            ballotSize ← ballotBoxes.size()
            if ballotBoxes.contains(b) then
                ballotSize ← ballotSize − 1
            end if
            if ballotSize > maxPluralityWins then
                maxPluralityWins ← ballotSize
            end if
        end for
        return pBallotSize / maxPluralityWins
end procedure
```

We modify Algorithm 1. The pseudocode of the new algorithm, $GreedyGerrymandering{+}_{plurality}$ is shown in Algorithm 4. It is similar to Algorithm 1 except it avoids repeated computations of the number of times each candidate wins plurality elections in the set of ballot boxes to find maximal ratios between a target candidate p and any other candidate c, using $CreatePartition$ and $ComputeRatio$. Then, we update $partition$ in lines $9-16$: if the reference ballot box b is won by a certain candidate c, we remove b from its set of ballot boxes. This has the same effect as the statement $B' \leftarrow B' \setminus \{b\}$ in line 8 of Algorithm 1.

Algorithm 4 : $GreedyGerrymandering{+}_{plurality}$

```
1:  procedure GreedyGerrymandering+(V, C, B, k, p)
2:      partition ← CreatePartition(V, C, B)
3:      B' ← B
4:      while |B'| > k do
5:          for all b ∈ B' do
6:              f_b ← ComputeRatio(partition, C, b, p)
7:          end for
8:          b ← arg max_{b∈B'} {f_b}
9:          for all c ∈ C do  //partition update: B' ← B'\{b}
10:             ballotBoxes ← partition.getValue(c)
11:             if ballotBoxes.contains(b) then
12:                 ballotBoxes ← ballotBoxes.remove(b)
13:                 partition.remove(c)
14:                 partition.put(c, ballotBoxes)
15:             end if
16:         end for
17:     end while
18:     if p wins a plurality of ballot boxes then
19:         return true
20:     else
21:         return false
22:     end if
23: end procedure
```

Correctness of $GreedyGerrymandering{+}_{plurality}$

Lemma 1. *For any set B of ballot boxes, $ComputeRatio$ correctly computes the maximal ratios between a target candidate p and any other candidate $c \in C$.*

Proof. $FindRatio$ (in Algorithm 1) is passed the original ballot box B in the very first call to it. Then, it returns the ratio of the number of ballot boxes won by p in B to the highest number of ballot boxes won by any other candidate $c \in C, c \neq p$, while excluding a reference ballot box b. The set B is updated for subsequent calls. We provide a matching description for $FindRatio$ as implemented in the $ComputeRatio$ procedure. First, we create a partition structure, $partition$, that keeps track of all the disjoint sets of the ballot boxes won by each $c \in C$ in the original set B of ballot boxes. Using $partition$, we obtain $pBallotSize$, the size of the set of ballot boxes won by p, then run through for each $c \neq p$ to obtain the largest size, $maxPluralityWins$, of the set of ballot boxes won by one of the candidates. Also, for both cases we exclude the reference ballot box b, in conformity with a similar step (line 17) in $FindRatio$. Then, we return the ratio, $\frac{pBallotSize}{maxPluralityWins}$. There is clearly a correspondency between the number of ballot boxes won by p and $c \neq p$ in $FindRatio$ and the sets sizes $pBallotSize$ and $maxPluralityWins$ of $ComputeRatio$. □

Theorem 2. *$GreedyGerrymandering{+}_{plurality}$ returns exactly the same result as Greedy Gerrymandering$_{plurality}$ when both algorithms are given the same set of inputs.*

Proof. This is obvious since the two algorithms use exactly the same set of instructions except for how the maximal ratios are computed, which Lemma 1 clarifies. □

Analysis of $GreedyGerrymandering{+}_{plurality}$

Theorem 3. *$GreedyGerrymandering{+}_{plurality}$ algorithm runs in $O(\max(zm(z+a), z(m-k+1)^2)$ time.*

Proof. We first note that all of the operations in a HashSet or HashMap data structure can each be completed in constant time. Consider the $CreatePartition$ procedure: there is a loop of size $|B| = m$ nested in another loop of size $|C| = z$. This nested loop calls the $plurality_b^c$ procedure each time. Thus, the running time of $CreatePartition$ is $O(zm(z+a))$. $CreatePartition$ is completed once in $GreedyGerrymandering{+}_{plurality}$. On the other hand, it takes $O(z)$ time to complete $ComputeRatio$ procedure. Now, in Algorithm 4, each of the loops starting from lines $4-5$ of the $GreedyGerrymandering{+}_{plurality}$ algorithm takes $O(m-k+1)$ time since the while loop in line 4 terminates when $|B'| = k$. This nested loop calls the $ComputeRatio$ procedure each time, for a running time of $O(z(m-k+1)^2)$. Since any of the district-based election parameters may be arbitrarily large, we conclude that the total running time of the $GreedyGerrymandering{+}_{plurality}$ algorithm is $O(\max(zm(z+a), z(m-k+1)^2)$. □

5 The Dynamic Programming Algorithm

We now describe the dynamic programming (DP) algorithm. Let b_i with $1 \leq i \leq m$, be the ith ballot box in the set of ballot boxes B. Let $1 \leq j \leq l \leq k$. Denote by $W(i,j)$ the number of plurality ballot boxes won by candidate p when asked: *how many ballot boxes does p wins given a subset of k ballot boxes $B' \subset B$?* The base case and the recurrence for the DP table are shown in Algorithm 5.

Algorithm 5 : $GerrymanderingDP(V, B, k, l, p)$

Base case: $W(i, 0) = 0 \quad 0 \leq i \leq m$

$W(0, j) = 0 \quad 0 \leq j \leq k$

Recurrence: for all (i, j) such that $i \geq j$,

$$W(i,j) = \begin{cases} x & \text{if } j > i \\ W(i-1,j) & \text{if } W(i-1,j) \geq j \\ w + plurality^p_{b_i} & \text{if } W(i-1,j) < j \text{ and } flag = 0 \\ w & \text{if } W(i-1,j) < j \text{ and } flag = 1 \end{cases}$$

where $x = \max\{W(i, j-1), W(i-1, j-1)\}$, $w = \max\{x, W(i-1, j)\}$, $flag = 0$ means $plurality^p_{b_i}$ is not yet called, and $flag = 1$ means it is already called for b_i.

The correctness of the algorithm is obvious as we compute the number of plurality wins for cell (i, j) by checking if p wins a plurality election for the current ballot box b_i and then update the score based on the maximum score from previously computed subproblems in cells $(i, j-1), (i-1, j-1), (i-1, j)$. We continue until we compute the score for cell (m, k), where we report true if $W(m, k) \geq l$, i.e., p wins a plurality of at least l ballot boxes, or false otherwise. Note that we call $plurality^p_{b_i}$ once per row of the DP table.

Theorem 4. *The $GerrymanderingDP$ algorithm runs in $O(mk(z + a))$ time.*

Proof. The size of the ballot boxes is m and the number of ballot boxes of interest for p is $l \leq k$. We call $plurality^p_{b_i}$ once per row to compute the value of each cell. Thus, the running time for this algorithm is $O(mk(z + a))$. □

6 The Randomized Algorithm

Our randomized method is based on an approach in (Bachrach et al. 2008) for approximating power indices in weighted voting games. The method in our case involves random selection of ballot boxes and checking whether candidate $p \in C$ wins in at least l districts. A natural question to ask then is: *what is the amount of random selections needed to achieve at least l wins assuming such enough winning ballot boxes are present in the set B of all ballot boxes?*

We approximate this number by doing large enough random selections of the ballot boxes. Similar to Bachrach et al., our proposed method determines the number η of random selections required for a given approximation accuracy $\epsilon > 0$ and probability δ of missing the accurate value of the number of wins for candidate p in the elections.

We define a random selection procedure. The procedure first generates a random number i corresponding to a ballot box b_i. Then calls $plurality^p_{b_i}$ to check if p wins plurality with ballot box b_i. We abuse notation and let the procedure return $\{1, b_i\}$ if p wins and $\{0, b_i\}$ otherwise. Let procedure $RandomSelect(V, C, B, p)$ randomly selects a ballot box as shown in Algorithm 5. This procedure models the *Bernoulli* distribution. Let X_i be Bernoulli random variables associated with different trials of the $RandomSelect$ procedure in which X_i is 1 if p wins plurality election and 0 otherwise. The Bernoulli random variable is defined with parameter ρ, where $P(X_i = 1) = \rho$ and $P(X_i = 0) = 1 - \rho$.

Algorithm 6 : $RandomSelect$

```
procedure RandomSelect(V, C, B, p)
    i ← generate a random number between 1 and |B|
    select ballot box b_i for district V_i ⊂ V from B
    if plurality^p_{b_i} = 1 then
        return {1, b_i}
    else
        return {0, b_i}
    end if
end procedure
```

Consider η independent repetitions of such trials. Let X be the number of successes. $X = \sum_{i=1}^{\eta} X_i$ is said to have a *Binomial distribution* with parameters η and ρ, denoted $X \sim B(\eta, \rho)$. Observe that ρ is unknown. Since $X \sim B(\eta, \rho)$ then the estimate for ρ, is $\hat{\rho} = \frac{X}{\eta}$, and is known to be unbiased. We now estimate the amount of district elections that candidate p wins. We employ a specialized version of the well-known Hoeffding's inequality (Hoeffding 1963), referred to as the Chernoff's bound to obtain a relationship among η, ϵ, and δ.

Theorem 5. *(Hoeffding's inequality). Let $X_1, \ldots, X_\eta$ be independent random variables on $\mathbb{R}$ such that $a_i \leq X_i \leq c_i$ with probability one. If $X = \sum_{i=1}^{\eta} X_i$ then for all $\epsilon > 0$*

$$Pr(|X - E[X]| \geq \epsilon) \leq 2e^{-\frac{2\epsilon^2}{\sum(c_i - a_i)^2}} \quad (1)$$

Hoeffding's inequality specializes to Chernoff's bound as follows. If X_i are independent and identically distributed Bernoulli random variables, then $a_i = 0, c_i = 1$, and $X \sim B(\eta, \rho)$. Since $E[X] = \eta\rho$, Chernoff's bound is:

$$Pr\left(\left|\frac{1}{\eta}\sum_{i=1}^{\eta} X_i - \rho\right| \geq \epsilon\right) \leq 2e^{-2\eta\epsilon^2} \quad (2)$$

which simplifies to the following

$$Pr\left(\left|\frac{X}{\eta} - \rho\right| \geq \epsilon\right) \leq 2e^{-2\eta\epsilon^2} \quad (3)$$

$$Pr(|\hat{\rho} - \rho| \geq \epsilon) \leq 2e^{-2\eta\epsilon^2}. \quad (4)$$

We ensure that the Chernoff's bound given in Equation 4 does not exceed the probability δ of missing the accurate value of the number of wins for p in all district-based elections, and simplify the expression as follows:

$$\begin{aligned} Pr(|\hat{\rho} - \rho| \geq \epsilon) \leq 2e^{-2\eta\epsilon^2} &\leq \delta \\ -2\eta\epsilon^2 &\leq \ln\frac{\delta}{2}. \end{aligned}$$

We obtain $\eta \geq \frac{1}{2\epsilon^2} \ln \frac{2}{\delta}$. Thus, the number of random selections for a given accuracy $\epsilon > 0$ and probability δ of missing the accurate value of the number of wins is at least $\frac{\ln \frac{2}{\delta}}{2\epsilon^2}$.

Let $RandomGerrymandering(V, C, B, l, p, \epsilon, \delta)$ be our randomized manipulation algorithm for district-based elections. $RandomGerrymandering$ randomly selects ballot boxes using $RandomSelect$. The algorithm starts with an empty value for a HashSet data structure, namely, *found*. *found* is continuously updated with ballot boxes corresponding to new districts that p wins plurality elections in. $RandomGerrymandering$ continues to select ballot boxes until we have enough number of selections, i.e., when $\eta \geq \frac{\ln \frac{2}{\delta}}{2\epsilon^2}$. The algorithm returns true if the number of districts won by p is l, i.e., $|found| = l$, otherwise, false. The pseudocode of the algorithm is shown in Algorithm 7.

Algorithm 7 : $RandomGerrymandering$

procedure $RandomGerrymandering(V, C, B, l, p, \epsilon, \delta)$
  $counter \leftarrow 0$
  $found \longleftarrow \emptyset$
  repeat
    $counter \leftarrow counter + 1$
    if $RandomSelect(V, C, B, p) = \{1, b_i\}$ **then**
      $found \longleftarrow found \cup \{b_i\}$
      if $|found| = l$ **then**
        return $true$
      end if
    end if
    $B \leftarrow B \setminus \{b_i\}$
  unitl $counter \leq \frac{1}{2\epsilon^2} \ln \frac{2}{\delta}$
  return $false$
end procedure

The correctness of the $RandomGerrymandering$ algorithm follows from the determination of sufficient number η of random selections for a given approximation accuracy ϵ and probability δ of missing the accurate value of the number of wins for candidate p as demonstrated using the Chernoff's bound. Having obtained this bound we biased our selection process by excluding any already selected ballot box in subsequent selections, thus increasing the probability of plurality ballot boxes win for p.

Analysis of $RandomGerrymandering$ Algorithm

Theorem 6. *The $RandomGerrymandering$ algorithm runs in $O((z + a) \cdot \frac{\ln \frac{2}{\delta}}{2\epsilon^2})$ time.*

Proof. We have a single loop that runs for $\frac{\ln \frac{2}{\delta}}{2\epsilon^2}$ times and calls $RandomSelect$ (which costs $O(z+a)$) each time. □

7 Conclusions

District-based elections where voters vote for a district representative and those representatives in turn vote to determine the overall winner have real-life applications ranging from human societies to artificial intelligence. We consider district-based elections and a form of manipulation, referred to as gerrymandering that is associated with the scheme. We extend a recent work of (Lewenberg, Lev, and Rosenschein 2017) which shows that the problem of gerrymandering in district-based elections is NP-complete, but also proposed a manipulation algorithm for the problem.

We propose three improved algorithms for this problem, and show that the proposed algorithms are polynomial in the parameters of the district-based elections with lower running times compared to the algorithm of Lewenberg, Lev, and Rosenschein. The implication of these results is that although gerrymandering in district-based elections is NP-complete in the worst case, it may be achieved with some instances of the elections with little computational efforts.

8 Acknowledgment

This research work is partially supported by the Virginia Military Institute's Professional Travel Funds.

References

Bachrach, Y.; Markakis, E.; Procaccia, A. D.; Rosenschein, J. S.; and Saberi, A. 2008. Approximating power indices. In *7th AAMAS Conference*, 943–950.

Bachrach, Y.; Lev, O.; Lewenberg, Y.; and Zick, Y. 2016. Misrepresentation in district voting. In *25th International Joint Conference on Artificial Intelligence*, 81–87.

Bartholdi, J. J.; Tovey, C. A.; and Trick, M. A. 1989. The computational difficulty of manipulating an election. *Social Choice and Welfare* 6(3):227–241.

Butler, D. 1992. The redrawing of parliamentary boundaries in Britain. *British Elections and Parties Yearbook* 2(1).

Engstrom, E. J. 2006. Stacking the states, stacking the house: The partisan consequences of congressional redistricting in 19th century. *APSR* 100(1):419–427.

Faliszewski, P.; Hemaspaandra, E.; and Schnoor, H. 2010. Manipulation of Copeland elections. In *9th AAMAS Conference*, 367–374.

Feix, M. R.; Lepelley, D.; Merlin, V. R.; and Rouet, J.-L. 2004. The probability of conflicts in a U.S. presidential type election. *Economic Theory* 23(2):227–257.

Guillermo, O., and Bernard, G. 1988. Optimal partisan gerrymandering. *Political Geography Quarterly* 7(1):5–22.

Hoeffding, W. 1963. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association* 58(301):13–30.

Lasisi, R. O., and Lasisi, A. A. 2017. Improved heuristic for manipulation of second-order Copeland elections. In *3rd Global Conference on Artificial Intelligence*, 162–174.

Lasisi, R. O. 2016. Manipulation of second-order Copeland elections: Heuristic and experiment. In *29th FLAIRS Conference*, 68–73.

Lewenberg, Y.; Lev, O.; and Rosenschein, J. S. 2017. Divide and conquer: using geographic manipulation to win district-based elections. In *16th AAMAS Conference*, 624–632.

Miller, N. R. 2014. The house size effect and the referendum paradox in u.s. presidential elections. *Electoral Studies* 35:265–271.