Algorithms for the Coalitional Manipulation Problem*

Michael Zuckerman† Ariel D. Procaccia‡ Jeffrey S. Rosenschein§

Abstract

We investigate the problem of coalitional manipulation in elections, which is known to be hard in a variety of voting rules. We put forward efficient algorithms for the problem in Borda, Maximin and Plurality with Runoff, and analyze their windows of error. Specifically, given an instance on which an algorithm fails, we bound the additional power the manipulators need in order to succeed. We finally discuss the implications of our results with respect to the popular approach of employing computational hardness to preclude manipulation.

Keywords: Computational social choice, Voting, Manipulation, Computational complexity

1 Introduction

Social choice theory is an extremely well-studied subfield of economics. In recent years, interest in the computational aspects of social choice, and in particular in the computational aspects of voting, has sharply increased.

In an election, a set of voters submit their (linear) preferences (i.e., rankings) over a set of candidates. The winner of the election is designated by a *voting rule*, which is basically a mapping from the space of possible *preference profiles* into candidates. A thorn in the side of social choice theory is formulated in the famous Gibbard-Satterthwaite Theorem [15, 26]. This theorem essentially states that for any voting rule that is not a *dictatorship*, there are elections in which at least one of the voters would benefit by lying. A dictatorship is a voting rule where one of the voters—the dictator—single-handedly decides the outcome of the election.

Since the 1970s, when this impossibility result was established, an enormous amount of effort has been invested in discovering ways to circumvent it. Two prominent and well-established ways are allowing payments [29, 4, 16], or restricting the voters' preferences [20].

In this paper, we wish to discuss a third path—the "path less taken", if you will—which has been explored by computer scientists. The Gibbard-Satterthwaite Theorem implies that in theory, voters are able to *manipulate* elections, i.e., bend them to their advantage by lying. But in practice, deciding which lie to employ may prove to be a hard computational problem; after all, there are a superpolynomial number of possibilities of ranking the candidates.

*A significantly shorter version of this paper (with most of the proofs omitted) appeared in the Proceedings of the Nineteenth ACM-SIAM Symposium on Discrete Algorithms (SODA-08). This work was also presented at the Dagstuhl Workshop on Computational Issues in Social Choice, October 2007.

†School of Engineering and Computer Science, The Hebrew University of Jerusalem, Jerusalem 91904, Israel, email: michez@cs.huji.ac.il. The author thanks Noam Nisan for a generous grant which supported this work.

‡Microsoft Israel R&D Center, 13 Shenka Street, Herzliya 46725, Israel, email: arielpro@gmail.com. The author was supported in this work by the Adams Fellowship Program of the Israel Academy of Sciences and Humanities.

§School of Engineering and Computer Science, The Hebrew University of Jerusalem, Jerusalem 91904, Israel, email: jeff@cs.huji.ac.il

Indeed, Bartholdi *et al.* [3] put forward a voting rule where manipulation is $\mathcal{NP}$-hard. In another important paper, Bartholdi and Orlin [2] greatly strengthened the approach by proving that the important Single Transferable Vote (STV) rule is hard to manipulate.

This line of research has enjoyed new life in recent years thanks to the influential work of Conitzer, Sandholm, and Lang [7].[1] The foregoing paper studied the complexity of coalitional manipulation. In this setting, there is a coalition of potentially untruthful voters, attempting to coordinate their ballots so as to get their favorite candidate elected. The authors further assume that the votes are weighted: some voters have more power than others. Conitzer et al. show that in a variety of prominent voting rules, coalitional manipulation is $\mathcal{NP}$-hard, *even if there are only a constant number of candidates* (for more details, see Section 2). This work has been extended in numerous directions, by different authors [12, 18, 25, 5, 8]; Elkind and Lipmaa [9], for example, strengthened the abovementioned results about coalitional manipulation by employing cryptographic techniques.

In short, computational complexity is by now a well-established method of circumventing the Gibbard-Satterthwaite Theorem. Unfortunately, a shortcoming of the results we mentioned above is that they are worst-case hardness results, and thus provide a poor obstacle against potential manipulators. Recent work regarding the frequency of manipulation has argued that with many worst-case hard-to-manipulate voting rules, a potential manipulator may be able to compute a manipulation in typical settings [6, 13]. In particular, Procaccia and Rosenschein [24, 23] have established some theoretical results regarding the frequency of success of an algorithm for the coalitional manipulation problem. The matter was further discussed by Erdélyi *et al.* [11]. In spite of this, the question of the tractability of the manipulation problem, and in particular of the coalitional manipulation problem, in typical settings is still wide-open.

Our Approach and Results. We wish to convince the reader that, indeed, the coalitional manipulation problem can be efficiently solved in typical settings under some prominent voting rules, but our approach differs from all previous work. We present efficient heuristic algorithms for the problem that provide theoretical guarantees. Indeed, we characterize small windows of instances on which our algorithms may fail; the algorithms are proven to succeed on all other instances.

Specifically, we prove the following results regarding three of the most prominent voting rules (in which coalitional manipulation is known to be $\mathcal{NP}$-hard even for a constant number of candidates):

Theorem.

1. In the Borda rule, if there exists a manipulation for an instance with certain weights, Algorithm 2 will succeed when given an extra manipulator with maximal weight.

2. In the Plurality with Runoff rule, if there exists a manipulation for an instance with certain weights, Algorithm 3 will succeed when given an extra manipulator with maximal weight.

3. In the Maximin rule, if there exists a manipulation for an instance with certain weights, Algorithm 1 will succeed when given two copies of the set of manipulators.

Significance in Artificial Intelligence. The sharply increased interest in computational aspects of voting is motivated by numerous applications of voting techniques and paradigms to problems in Artificial Intelligence (AI). These applications include work in AI subfields as diverse as Planning [10], Automated Scheduling [17], Recommender Systems [14], Collaborative Filtering [22], Information Extraction [27], and Computational Linguistics [21].

[1] Historical note: although we cite the JACM 2007 paper, this work originated in a AAAI 2002 paper.

Unfortunately, in the application of voting to AI, some of the problems investigated in Social Choice Theory, and in particular the issue of manipulation, become especially acute. Indeed, multiagent systems are often inhabited by heterogeneous, self-interested agents. Such agents, unlike human beings, can be designed to be *rational*, and constantly engaged in computations meant to increase their utility. In particular, a self-interested agent could seize the opportunity to manipulate an election to its benefit if such an opportunity were computationally easy to recognize (unless specifically programmed not to).

The agenda of circumventing the Gibbard-Satterthwaite Theorem via computational complexity is, once again, most relevant and compelling when the voters are software agents that populate a multiagent system, since the effective, bounded rationality of such agents is practically governed by the laws of computational complexity. This is why the agenda has become a prominent one in AI, with numerous papers on the subject published in the major AI conferences over the last five years. As of yet, there are few papers on frequency of manipulation, rather than on its worst-case complexity. We feel that this line of work on frequency of manipulation may influence the entire direction of the computational social choice research agenda (see Section 5 for more details regarding work on frequency of manipulation).

Structure of the Article. In Section 2 we describe the major voting rules and formulate the coalitional manipulation problem. In Section 3 we present and analyze our algorithms in three subsections: Borda, Plurality with Runoff, and Maximin. We provide some results regarding an unweighted setting in Section 4. In Section 5 we describe related work at length. Finally, we discuss our approach in Section 6.

2 Voting Rules and Manipulation Problems

An election consists of a set $C = \{c_1, \ldots, c_m\}$ of candidates and a set $S = \{v_1, \ldots, v_{|S|}\}$ of voters. Each voter provides a total order on the candidates. To put it differently, each voter submits a ranking of the candidates. The voting setting also includes a *voting rule*, which is a function from the set of all possible combinations of votes to C.

We shall discuss the following voting rules (whenever the voting rule is based on scores, the candidate with the highest score wins):

- *Scoring rules.* Let $\vec{\alpha} = \langle \alpha_1, \ldots, \alpha_m \rangle$ be a vector of non-negative integers such that $\alpha_1 \geq \alpha_2 \geq \ldots \geq \alpha_m$. For each voter, a candidate receives α_1 points if it is ranked first by the voter, α_2 if it is ranked second, etc. The *score* of a candidate is the total number of points the candidate receives. The scoring rules that we will consider are: *Borda*, where $\vec{\alpha} = \langle m-1, m-2, \ldots, 0 \rangle$; *Veto*, where $\vec{\alpha} = \langle 1, 1, \ldots, 1, 0 \rangle$; and *Plurality*, where $\vec{\alpha} = \langle 1, 0, \ldots, 0 \rangle$.

- *Maximin.* For any two distinct candidates x and y, let $N(x, y)$ be the number of voters who prefer x to y. The *maximin score* of x is $\sigma(x) = \min_{y \neq x} N(x, y)$.

- *Copeland.* For any two distinct candidates x and y, let $C(x, y) = +1$ if $N(x, y) > N(y, x)$ (in this case we say that x *beats* y *in their pairwise election*), $C(x, y) = 0$ if $N(x, y) = N(y, x)$, and $C(x, y) = -1$ if $N(x, y) < N(y, x)$. The *Copeland score* of candidate x is $\sigma(x) = \sum_{y \neq x} C(x, y)$.

- *Plurality with Runoff.* In this rule, a first round eliminates all candidates except the two with the highest plurality scores. The second round determines the winner between these two by their pairwise election.

In some settings the voters are *weighted*. A *weight function* is a mapping $w : S \to \mathbb{N}$. When voters are weighted, the above rules are applied by considering a voter of weight l to be l different voters.

Definition 2.1.

1. In the Constructive Coalitional Weighted Manipulation (CCWM) problem in a voting rule F, we are given a set C of candidates, with a distinguished candidate $p \in C$, a set of weighted voters S that already cast their votes (these are the truthful voters), and a list of weights W for a set of voters T that still have not cast their votes (the manipulators). We are asked whether there is a way to cast the votes in T such that p wins the election under the voting rule F.

2. Constructive Coalitional Unweighted Manipulation (CCUM) problem is a special case of CCWM problem where all the weights equal 1.

Remark 2.2. We implicitly assume in both questions that the manipulators have full knowledge about the other votes. Unless explicitly stated otherwise, we also assume that ties are broken adversarially to the manipulators, so if p ties with another candidate, p loses. The latter assumption is equivalent to formulating the manipulation problems in their *unique winner* version, when one assumes that all candidates with maximal score win, but asks that p be the only winner.

Theorem 2.3 ([7])**.** *The CCWM problem in Borda, Veto, Maximin, Copeland, and Plurality with Runoff is $\mathcal{NP}$-complete, even when the number of candidates is constant.*

Throughout this paper we will use the convention that $|C| = m$, $|S| = N$ and $|T| = n$. Whenever the voting rule is based on scores, we will denote by $\sigma_{S,j}(c)$ the accumulated score of candidate c from the voters in S and the first j voters of T (fixing some order on the voters of T). Whenever it is clear from the context that S is fixed, we will use simply $\sigma_j(c)$ for the same. Also, for $G \subseteq C$, $0 \leq j \leq n$ we will write $\sigma_j(G) = \{\sigma_j(g) \mid g \in G\}$. For two lists A, B (ordered multisets), we denote by $A + B$ the list that is obtained after B is appended to A.

3 Weighted Coalitional Manipulation

We begin our contribution by presenting a general greedy algorithm for the coalitional manipulation problem. Some of our main results concern this algorithm or its restriction to scoring rules.

The greedy algorithm is given as Algorithm 1. It works as follows: the manipulators, according to descending weights, each rank p first and rank the other candidates in a way that minimizes their maximum score. This algorithm is a generalization of the one that appeared in Bartholdi et al. [3].

Definition 3.1. We refer to an iteration of the main for loop in lines 4–12 of the algorithm as a *stage* of the algorithm.

We will use the fact that for many voting rules, if there exists a manipulation for a coalition of manipulators with weight list W, then there exists a manipulation for a coalition of manipulators with weight list W' where $W' \supseteq W$. Normally, if the coalition is too small then there is no manipulation, and this is indeed what the algorithm will report. On the other hand, if the coalition is large enough, then the greedy algorithm will find the manipulation. So there remains a window of error, where for some coalitions there could exist a manipulation, but the algorithm may not find it. We are interested in bounding the size of this window. We first formulate the monotonicity property described above.

Algorithm 1 Decides CCWM

procedure GREEDY(C, p, X_S, W) ▷ X_S is the set of preferences of voters in S, W is the list of weights for voters in T, $|W| = |T| = n$
  sort(W) ▷ Sort the weights in descending order
  $X \leftarrow \emptyset$ ▷ Will contain the preferences of T
  for $j = 1, \ldots, n$ **do** ▷ Iterate over voters by descending weights
    $P_j \leftarrow (p)$ ▷ Put p at the first place of the j-th preference list
    for $t = 2, \ldots, m$ **do** ▷ Iterate over places of j-th preference list
      ▷ Evaluate the score of each candidate if j would put it at the next available place
      Pick $c \in \text{argmin}_{c \in C \setminus P_j}\{\text{Score of } c \text{ from } X_S \cup X \cup \{P_j + \{c\}\}\}$
      $P_j = P_j + \{c\}$ ▷ Add c to j's preference list
    end for
    $X \leftarrow X \cup \{P_j\}$
  end for
  $X_T \leftarrow X$
  if $\text{argmax}_{c \in C}\{\text{Score of } c \text{ based on } X_S \cup X_T\} = \{p\}$ **then**
    return true ▷ p wins
  else
    return false
  end if
end procedure

Definition 3.2. In the context of the CCWM problem, a voting rule is said to be *monotone in weights* if it satisfies the following property: whenever there is a manipulation making p win for manipulator set T with weight list W, there is also a manipulation making p win for manipulator set T' with weight list W', where $T' \supseteq T$, $W' \supseteq W$.

Monotonicity in weights is a prerequisite for the type of analysis we wish to present. However, surprisingly, not all the basic voting rules have this property; in particular, the prominent Copeland rule does not possess it. We show this by example in Appendix A.

3.1 Borda

In this subsection, we analyze the performance of Algorithm 1 with respect to the Borda voting rule. Note that, in the context of scoring rules, Algorithm 1 reduces to Algorithm 2. This algorithm first appeared in Procaccia and Rosenschein [24]. In this specific instantiation of Algorithm 1, we do not require sorting of the manipulator weights, as this does not play a part in our analysis.

Lemma 3.3. *Scoring rules are monotone in weights.*

Proof. Let C be the candidate set; $p \in C$ is the preferred candidate, S is the set of truthful voters, and W are the weights for the manipulators T. Denote $|C| = m$, $|S| = N$, $|W| = |T| = n$. It is enough to show that if there is a manipulation for the set T, then for the same instance with manipulators $T' = T + \{v\}$ with weight list $W' = W + \{w\}$, where $w \geq 1$ is an integer, there is also a manipulation, and the rest will follow by induction.

Let $\vec{\alpha} = \langle \alpha_1, \ldots, \alpha_m \rangle$ be the score vector of the rule. Let X_S be the preference orders of the voters in S, and X_T be the preference orders of voters in T that make p win. Fix some order on the voters in T. By definition, for all $c \in C \setminus \{p\}$, $\sigma_n(c) < \sigma_n(p)$. Let the additional voter of T'

Algorithm 2 Decides CCWM in Scoring rules

procedure SCORING-RULES-GREEDY($C, p, \sigma_0(C), W$) ▷ $\sigma_0(C)$ is the list of scores of candidates distributed by voters in S, W is the list of weights for voters in T, $|W| = |T| = n$
  for $j = 1, \ldots, n$ **do** ▷ Go over voters in T
    $\sigma_j(p) = \sigma_{j-1}(p) + w_j\alpha_1$ ▷ Put p at the first place of the j-th preference list
    Let $t_1, t_2, \ldots, t_{m-1}$ s.t. $\forall l,\ \sigma_{j-1}(c_{t_{l-1}}) \leq \sigma_{j-1}(c_{t_l})$
    j votes $p \succ c_{t_1} \succ \ldots \succ c_{t_{m-1}}$
    for $l = 1, \ldots, m-1$ **do** ▷ Update the scores
      $\sigma_j(c_{t_l}) = \sigma_{j-1}(c_{t_l}) + w_j\alpha_{l+1}$
    end for
  end for
  if $\text{argmax}_{c \in C}\{\sigma_n(c)\} = \{p\}$ **then** ▷ p wins
    return true
  else
    return false
  end if
end procedure

rank p at the first place, and report some arbitrary order on the other candidates. Then for all $c \in C \setminus \{p\}$, $\sigma_{n+1}(p) = \sigma_n(p) + w\alpha_1 > \sigma_n(c) + w\alpha_1 \geq \sigma_{n+1}(c)$. Hence, p wins. □

We are now ready to present our theorem regarding the Borda rule.

Theorem 3.4. *In the CCWM problem under Borda, let C be a set of candidates with $p \in C$ a preferred candidate, S a set of voters who already cast their votes. Let W be the weight list for the set T. Then:*

1. *If there is no ballot making p win the election, then Algorithm 2 will return* false.
2. *If there exists a ballot making p win the election, then for the same instance with weight list $W + \{w'_1, \ldots, w'_k\}$, where $k \geq 1, \sum_{i=1}^{k} w'_i \geq \max(W)$, Algorithm 2 will return* true.

Before we proceed to the theorem's proof, a short discussion is in order. Despite its mathematical formulation, one should not think of Item 2 of the theorem as saying that if the algorithm fails on one instance, it would succeed on another. Rather, the theorem implies that the algorithm succeeds on any given instance such that there is a "smaller" instance (where the manipulators have less weight) on which success is possible. Here the monotonicity in weights property comes into play. Also note that Item 1 of the theorem is true for any constructive algorithm; this item (which also appears in our subsequent theorems) is trivially satisfied.

Another interesting point is that this theorem can be viewed as implying that Algorithm 2 gives some sort of additive approximation ratio. Formally, it seems unnatural to adopt the notion of approximation algorithms in the context of the CCWM problem. However, the exact way in which the theorem yields approximation guarantees will become apparent when we discuss the unweighted setting, in Section 4.

A key notion for the proof of the theorem is the definition of the set G_W. Let W be list of weights; we define G_W as follows. Run the algorithm $n+1$ stages with the weights $W + \{w\}$, where w is an arbitrary weight. Let $G^0_W = \text{argmax}_{g \in C \setminus \{p\}}\{\sigma_0(g)\}$, and, by induction, for $s = 1, 2, \ldots$: $G^s_W = G^{s-1}_W \cup \{g \mid g \text{ was ranked below some } g' \in G^{s-1}_W \text{ in some stage } l,\ 1 \leq l \leq n+1\}$. Finally, let $G_W = \cup_{0 \leq s} G^s_W$.

Informally, G_W is constructed by taking candidates that initially have maximum score, and then inductively adding candidates that are ranked by the algorithm below candidates that were already added to the set. Since the algorithm ranks stronger candidates below weaker candidates, only strong candidates are ultimately members of G_W. The additional arbitrary weight w, and the existence of stage $n+1$ (when there are in fact only n manipulators with weights W) are just a formality: we are also interested in the way the algorithm would rank the candidates after all the manipulators have cast their ballots, but we do not care about their scores after this final "virtual" ranking.

Observe that the indices $s = 1, \ldots$ are not directly related to stages $l = 1, \ldots, n$: a candidate c is added to G_W^s if he is ranked below a candidate $c' \in G_W^{s-1}$ in *some* stage $l = 1, \ldots, n$ (e.g., not necessarily in stage s).

Notice that the above definition is independent of the weight w, as this weight is used only in stage $n+1$, so it does not impact the preferences of the voters, and thus it does not impact G_W. From the definition, $G_W^0 \subseteq G_W^1 \subseteq \ldots \subseteq C \setminus \{p\}$. Furthermore, as $|C \setminus \{p\}| = m-1$, it follows that there exists $0 \leq s' \leq m-2$ s.t. $G_W^{s'} = G_W^{s'+1}$, and thus $G_W = G_W^{s'} = G_W^{m-2}$.

We are now ready to unfold the proof of Theorem 3.4. The proof relies on Lemmata 3.5–3.13. The general intuition of the proof is as follows. Consider the candidates in G_W; we show that if there exists a manipulation, it must be possible to get the score of p to be higher than their average score. The difficult part is to show that the average score of the candidates in G_W is relatively close to the maximal final score. As a result, a few additional manipulators are sufficient to push p above the maximal score as well.

In the first three lemmata, Lemmata 3.5–3.7, we show that the candidates in G_W are the ones with highest scores and we give a connection between their average score and the success of the algorithm in finding a manipulation. The next straightforward lemma formalizes the intuition that the strong candidates in G_W are always ranked last by the algorithm.

Lemma 3.5. *Given W, the candidates in G_W were ranked at each stage l, $1 \leq l \leq n+1$ at the $|G_W|$ last places, i.e., they were always granted the points $|G_W|-1, \ldots, 0$.*

Proof. If, by way of contradiction, there exists $c \in C \setminus G_W$ that was ranked in some stage in one of the last $|G_W|$ places, then there is $g \in G_W$ that was ranked above c at this stage. Let $s \geq 0$ such that $g \in G_W^s$. By definition, $c \in G_W^{s+1} \Rightarrow c \in G_W$, a contradiction. □

Lemma 3.6, directly building on Lemma 3.5, states that when the algorithm terminates, the candidates in G_W have scores that are higher than any candidate outside the set, perhaps except p.

Lemma 3.6. *For all $c \in C \setminus (G_W \cup \{p\})$, it holds that $\sigma_n(c) \leq \min_{g \in G_W}\{\sigma_n(g)\}$.*

Proof. Suppose, for contradiction, that there are $c \in C \setminus (G_W \cup \{p\})$ and $g \in G_W$, s.t. $\sigma_n(c) > \sigma_n(g)$. Then in stage $n+1$, c would have been ranked below g. Let $s \geq 0$ s.t. $g \in G_W^s$. Then $c \in G_W^{s+1} \Rightarrow c \in G_W$, a contradiction. □

The next lemma clarifies the connection between the definition of G_W and Theorem 3.4. Indeed, it links the average score of the candidates in G_W (when the algorithm terminates) and the answer returned by the algorithm.

Lemma 3.7. *Given W, $|W| = n$, let G_W be as before. Denote by $q(W)$ the average score of candidates in G_W after n stages: $q(W) = \frac{1}{|G_W|}\sum_{g \in G_W} \sigma_n(g)$. Then:*

1. *If $\sigma_n(p) \leq q(W)$ then there is no manipulation that makes p win the election, and the algorithm will return* false*.*

2. *If $\sigma_n(p) > \max_{g \in G_W}\{\sigma_n(g)\}$, then there is a manipulation that makes p win, and the algorithm will find it.*

Proof. We first prove part 1. Denote $W = \{w_1, \ldots, w_n\}$. We have the set G_W, and we suppose that $\sigma_n(p) \leq q(W)$. Let us consider a ballot X_T of votes in T, and let $\sigma'_n(c)$ be the scores of the candidates $c \in C$ implied by this ballot (including all the votes in S). Since in Algorithm 2 p was placed at the top of the preference of each voter in T, we have that:

$$\sigma_n(p) = \sigma_0(p) + \sum_{j=1}^{n} w_j(m-1) \geq \sigma'_n(p) \tag{1}$$

On the other hand, since by Lemma 3.5, in Algorithm 2 the candidates of G_W were ranked by all the voters in T in the last $|G_W|$ places, it follows that

$$q(W) = \frac{1}{|G_W|}\left(\sum_{g \in G_W} \sigma_0(g) + \sum_{j=1}^{n} w_j \sum_{i=0}^{|G_W|-1} i\right) \leq \frac{1}{|G_W|}\sum_{g \in G_W} \sigma'_n(g) =: q'(X_T) \tag{2}$$

Combining together (1) and (2) we get that $\sigma'_n(p) \leq q'(X_T)$. There is at least one $g \in G_W$ such that $\sigma'_n(g) \geq q'(X_T)$ (since $q'(X_T)$ is the average of the scores), hence $\sigma'_n(p) \leq \sigma'_n(g)$, and so p will not win when X_T is applied.

Also note that Algorithm 2 returns *true* only if it constructs a (valid) ballot that makes p win, and so for the case $\sigma_n(p) \leq q(W)$ the algorithm will return *false*.

We now prove part 2 of the lemma. If $\sigma_n(p) > \max_{g \in G_W}\{\sigma_n(g)\}$, then by Lemma 3.6 for all $c \in C \setminus \{p\}$, $\sigma_n(p) > \sigma_n(c)$, and so the algorithm will find the manipulation. □

Lemma 3.8 is independent of the lemmata before and after it, but is used directly in the proof of Theorem 3.4. It gives a connection between the average score of the candidates in $G_{W+\{w\}}$ and G_W, where w is the weight of some additional manipulator. In other words, it bounds the effect that adding a manipulator has on the average score of the strong candidates.

Lemma 3.8. *Let $G_W, q(W)$ be as before. Then for $w \geq 1$, $q(W + \{w\}) - q(W) \leq w\frac{m-2}{2}$.*

Proof. First, $G_W \subseteq G_{W+\{w\}}$, because for all $s \geq 0, G_W^s \subseteq G_{W+\{w\}}^s$. Now, for all $g \in G_{W+\{w\}} \setminus G_W$, g was not ranked in the first $n+1$ stages after any candidate in G_W, and so for all $g' \in G_W$, $\sigma_n(g) \leq \sigma_n(g')$, and hence

$$\frac{1}{|G_{W+\{w\}}|}\sum_{g \in G_{W+\{w\}}} \sigma_n(g) \leq \frac{1}{|G_W|}\sum_{g' \in G_W} \sigma_n(g') = q(W)$$

Now we can proceed:

$$\begin{aligned} q(W+\{w\}) &= \frac{1}{|G_{W+\{w\}}|}\sum_{g\in G_{W+\{w\}}}\sigma_{n+1}(g) \\ &= \frac{1}{|G_{W+\{w\}}|}\sum_{g\in G_{W+\{w\}}}\sigma_n(g) + \frac{w}{|G_{W+\{w\}}|}\sum_{i=0}^{|G_{W+\{w\}}|-1} i \\ &\leq q(W) + \frac{w}{m-1}\sum_{i=0}^{m-2} i \\ &= q(W) + w\frac{m-2}{2} \end{aligned}$$

And so, $q(W+\{w\}) - q(W) \leq w\frac{m-2}{2}$. □

The purpose of Lemmata 3.10–3.13 is to show that for any weight list W, $|W| = n$ it holds that $\max_{g\in G_W}\{\sigma_n(g)\} - q(W) \leq \max(W)\frac{m-2}{2}$. This fact is stated in Lemma 3.13, which is the only one directly used in the proof of Theorem 3.4.

First we need to show that the scores of candidates in G_W are concentrated, in a sense. This is intuitive, since the algorithm doesn't allow the score of any candidate in G_W to "escape" by ranking it close to the bottom if its score becomes too high in some stage. We will require the following definition:

Definition 3.9. For an integer $w \geq 0$, a finite non-empty set of integers A is called *w-dense* if when we sort the set in nonincreasing order $b_1 \geq b_2 \geq \ldots \geq b_k$ (such that $\{b_1, \ldots, b_k\} = A$), it holds that for all $1 \leq j \leq k-1$, $b_{j+1} \geq b_j - w$.

So, formally, we want to show (Lemma 3.12) that $\sigma_n(G_W)$ is $w_{\max}$-dense, where $w_{\max} = \max W$. This will be accomplished via a number of technical steps.

Lemma 3.10. *Let W be a list of weights, $|W| = n$. Let $G_W = \bigcup_{0\leq s} G_W^s$, as before. Then for all $s \geq 1$ and $g \in G_W^s \setminus G_W^{s-1}$ there exist $g' \in G_W^{s-1}$, $X \subseteq C \setminus \{p\}$ (perhaps $X = \emptyset$) and j, $0 \leq j \leq n$, s.t. $\{\sigma_j(g), \sigma_j(g')\} \cup \sigma_j(X)$ is $w_{\max}$-dense, where $w_{\max} = \max(W)$.*

Proof. Let $s \geq 1$ and $g \in G_W^s \setminus G_W^{s-1}$. By definition, there exist $g' \in G_W^{s-1}$ and a minimal j, $1 \leq j \leq n+1$, such that g was ranked below g' in stage j. We distinguish between two cases:

Case 1: $j > 1$. In this case g was ranked above g' in stage $j-1$. So we have:

$$\sigma_{j-1}(g) \geq \sigma_{j-1}(g') \tag{3}$$

$$\sigma_{j-2}(g) \leq \sigma_{j-2}(g') \tag{4}$$

Denote $\alpha_d(h) := m - (\text{place of } h \in C \text{ at the preference list of voter } d)$. Further, denote by w_d the weight of voter d (so in stage d, h gets $w_d\alpha_d(h)$ points). g was ranked above g' in stage $j-1$, and hence $\alpha_{j-1}(g) > \alpha_{j-1}(g')$. Denote $l = \alpha_{j-1}(g) - \alpha_{j-1}(g')$, and $w := w_{j-1}$. Let $g' = g_0, g_1, \ldots, g_l = g$ be the candidates that got in stage $j-1$ the points $w\alpha_{j-1}(g'), w(\alpha_{j-1}(g')+1), \ldots, w(\alpha_{j-1}(g')+l)$, respectively. Our purpose is to show that $\{\sigma_{j-1}(g_0), \ldots, \sigma_{j-1}(g_l)\}$ is w-dense, and therefore $w_{\max}$-dense. By definition of the algorithm,

$$\sigma_{j-2}(g_0) \geq \sigma_{j-2}(g_1) \geq \ldots \geq \sigma_{j-2}(g_l) \tag{5}$$

Denote $u_t = \sigma_{j-2}(g_t) + w\alpha_{j-1}(g')$ for $0 \le t \le l$. Then

$$\forall t,\ 0 \le t \le l,\ \sigma_{j-1}(g_t) = u_t + wt \tag{6}$$

So we need to show that $\{u_t + wt \mid 0 \le t \le l\}$ is w-dense. It is enough to show that:

(a) For all t, $0 \le t \le l$, if $u_t + wt < u_0$, then there exists t', $t < t' \le l$, s.t. $u_t + wt < u_{t'} + wt' \le u_t + w(t+1)$, and

(b) For all t, $0 \le t \le l$, if $u_t + wt > u_0$, then there exists t', $0 \le t' < t$, s.t. $u_t + w(t-1) \le u_{t'} + wt' < u_t + wt$.

Proof of (a): From (5) we get

$$u_0 \ge \cdots \ge u_l \tag{7}$$

Also from (3) and (6) we have $u_0 \le u_l + wl$. Let $0 \le t \le l-1$ s.t. $u_t + wt < u_0$. Let us consider the sequence $u_t + wt, u_{t+1} + w(t+1), \ldots, u_l + wl$. Since $u_t + wt < u_0 \le u_l + wl$, it follows that there is a minimal index t', $t < t' \le l$ s.t. $u_t + wt < u_{t'} + wt'$. Then $u_{t'-1} + w(t'-1) \le u_t + wt$, and thus

$$u_{t'-1} + wt' \le u_t + w(t+1) \tag{8}$$

From (7) $u_{t'} \le u_{t'-1}$, and then

$$u_{t'} + wt' \le u_{t'-1} + wt' \tag{9}$$

Combining (8) and (9) together, we get $u_{t'} + wt' \le u_t + w(t+1)$. This concludes the proof of (a). The proof of (b) is analogous, by choosing t' to be the maximal index such that $u_{t'} + wt' < u_t + wt$.

Case 2: $j = 1$. We proceed by essentially reducing this case to Case 1. In Case 2 we have that $s \ge 2$, because otherwise, if $s = 1$, then $g' \in G_W^0$; therefore $\sigma_0(g) \ge \sigma_0(g') = \max_{h \in C \setminus \{p\}}\{\sigma_0(h)\} \Rightarrow g \in G_W^0$, a contradiction. $g' \notin G_W^{s-2}$, because otherwise, by definition, $g \in G_W^{s-1}$. Therefore there exists $g'' \in G_W^{s-2}$ s.t. g' was ranked below g'' in some stage j', i.e., $\sigma_{j'-1}(g') \ge \sigma_{j'-1}(g'')$. g has never been ranked below g'' (because otherwise $g \in G_W^{s-1}$), and it follows that $\sigma_{j'-1}(g) \le \sigma_{j'-1}(g'')$. By combining the last arguments, we get that $\sigma_{j'-1}(g) \le \sigma_{j'-1}(g')$.

Let j_0 be minimal s.t. $\sigma_{j_0}(g) \le \sigma_{j_0}(g')$. As in stage 1 g was ranked below g', it holds that $\sigma_0(g) \ge \sigma_0(g')$. If $j_0 = 0$ then $\sigma_0(g) = \sigma_0(g')$, hence $\{\sigma_0(g), \sigma_0(g')\}$ is 0-dense, and in particular $w_{\max}$-dense.

Otherwise ($j_0 \ne 0$) it holds that $\sigma_{j_0-1}(g) > \sigma_{j_0-1}(g')$ by the minimality of j_0. So, we have that

$$\sigma_{j_0}(g') \ge \sigma_{j_0}(g),$$

and

$$\sigma_{j_0-1}(g') \le \sigma_{j_0-1}(g).$$

These two inequalities are analogous to (3) and (4), with $j-1$ replaced by j_0, and the roles of g and g' exchanged. From this point we can proceed exactly as in Case 1, keeping in mind these cosmetic changes. □

The following lemma asserts that a dense set of the scores of candidates in stage j can be replaced by a dense set of *final* scores by considering a possibly larger set of candidates.

Lemma 3.11. *Let W be a list of weights, $|W| = n$, $w_{\max} = \max(W)$. Let $H \subseteq C \setminus \{p\}$ s.t. $\sigma_j(H)$ is $w_{\max}$-dense for some $0 \le j \le n$. Then there exists H', $H \subseteq H' \subseteq C \setminus \{p\}$ s.t. $\sigma_n(H')$ is $w_{\max}$-dense.*

Proof. We have $H \subseteq C \setminus \{p\}$ and $0 \leq j \leq n$, s.t. $\sigma_j(H)$ is $w_{\max}$-dense. Denote $H_j := H$. Define inductively for $t = j, j+1, \ldots, n-1$: $H_{t+1} = \{g \in C \setminus \{p\} \mid \min_{h \in H_t}\{\sigma_t(h)\} \leq \sigma_t(g) \leq \max_{h \in H_t}\{\sigma_t(h)\}\}$. Of course, for all t, $H_t \subseteq H_{t+1}$. It is easy to see that if for some $j \leq t \leq n-1$, $\sigma_t(H_t)$ is $w_{\max}$-dense, then $\sigma_{t+1}(H_{t+1})$ is also $w_{\max}$-dense. So, we get by induction that $\sigma_n(H_n)$ is $w_{\max}$-dense, and $H \subseteq H_n \subseteq C \setminus \{p\}$. □

Lemma 3.12. *Let W be a list of weights, $|W| = n$, $w_{\max} = \max(W)$. Let G_W be as before. Then the set $\sigma_n(G_W)$ is $w_{\max}$-dense.*

Proof. Let $g = g_0 \in G_W$. If $g \notin G_W^0$, then $g \in G_W^s \setminus G_W^{s-1}$ for some $s \geq 1$. By Lemma 3.10 there exist $g_1 \in G_W^{s-1}$ and $X_1 \subseteq C \setminus \{p\}$ s.t. $\{\sigma_j(g_0), \sigma_j(g_1)\} \cup \sigma_j(X_1)$ is $w_{\max}$-dense for some $0 \leq j \leq n$. By Lemma 3.11 there exists X_1', $X_1 \subseteq X_1' \subseteq C \setminus \{p\}$, s.t. $\{\sigma_n(g_0), \sigma_n(g_1)\} \cup \sigma_n(X_1')$ is $w_{\max}$-dense. Denote $Z_1 := \{g_0, g_1\} \cup X_1'$. Similarly, if $g_1 \notin G_W^0$, then there exist $g_2 \in G_W^{s-2}$ and X_2' s.t. $\{\sigma_n(g_1), \sigma_n(g_2)\} \cup \sigma_n(X_2')$ is $w_{\max}$-dense. Denote $Z_2 := \{g_1, g_2\} \cup X_2'$, etc. Thus, we can build a sequence of sets $Z_1, \ldots, Z_{s+1}$, s.t. for all $1 \leq t \leq s+1$, $\sigma_n(Z_t)$ is $w_{\max}$-dense, $g = g_0 \in Z_1$ and for each $1 \leq t \leq s$ there exists $g_t \in G_W^{s-t}$ s.t. $g_t \in Z_t \cap Z_{t+1}$, and in particular, $g_s \in G_W^0$.

It is easy to see that for two w-dense sets A, A', if $A \cap A' \neq \emptyset$ then $A \cup A'$ is also w-dense, and hence we get $Z_g := \bigcup_{t=1}^{s+1} Z_t$ is $w_{\max}$-dense. Note that $\sigma_0(G_W^0)$ is $w_{\max}$-dense, and hence there exists $\hat{Z}, G_W^0 \subseteq \hat{Z} \subseteq C \setminus \{p\}$ s.t. $\sigma_n(\hat{Z})$ is $w_{\max}$-dense. Hence, $\sigma_n(Z_g \cup \hat{Z})$ is $w_{\max}$-dense.

The sets $Z_g \cup \hat{Z}$, for all $g \in G_W$, all intersect in $\hat{Z}$, and their union is $\hat{Z} \cup \bigcup_{g \in G_W} Z_g$. We deduce that $\{\sigma_n(g) \mid g \in \hat{Z} \cup \bigcup_{g \in G_W} Z_g\}$ is $w_{\max}$-dense. By Lemma 3.6, for all $h \in \hat{Z} \cup \bigcup_{g \in G_W} Z_g$, if $h \notin G_W$, then $\sigma_n(h) \leq \min_{g \in G_W}\{\sigma_n(g)\}$, and hence $\sigma_n(G_W)$ is a also $w_{\max}$-dense. □

Lemma 3.13. *Let W be a list of weights, $|W| = n$, $w_{\max} = \max(W)$. Let G_W be as before, and denote $q(W) = \frac{1}{|G_W|}\sum_{g \in G_W} \sigma_n(g)$, as before. Then $\max_{g \in G_W}\{\sigma_n(g)\} - q(W) \leq w_{\max}\frac{m-2}{2}$.*

Proof. Sort the members of G_W by their scores after the n-th stage, i.e., $G_W = \{g_1, \ldots, g_{|G_W|}\}$ s.t. for all $1 \leq t \leq |G_W|-1$, $\sigma_n(g_t) \geq \sigma_n(g_{t+1})$. Denote for $1 \leq t \leq |G_W|$, $u_t = \sigma_n(g_1) - w_{\max}(t-1)$, and let $U = \{u_1, \ldots, u_{|G_W|}\}$. $|U| = |G_W|$, $\max U = \sigma_n(g_1) = \max_{g \in G_W}\{\sigma_n(g)\}$. By Lemma 3.12, it is easy to see that for all $1 \leq t \leq |G_W|$, $\sigma_n(g_t) \geq u_t$. Consequently, $q(W) \geq \frac{1}{|G_W|}\sum_{t=1}^{|G_W|} u_t$, hence

$$\max_{g \in G_W}\{\sigma_n(g)\} - q(W) = u_1 - q(W) \leq u_1 - \frac{1}{|G_W|}\sum_{t=1}^{|G_W|} u_t = w_{\max}\frac{|G_W|-1}{2} \leq w_{\max}\frac{m-2}{2}.$$

□

We are finally ready to prove Theorem 3.4.

Proof of Theorem 3.4. Regarding part 1, Algorithm 2 returns *true* only if it constructs a (valid) ballot that makes p win, and thus if there is no ballot making p win, Algorithm 2 will return *false*.

We now prove part 2 of the theorem. Suppose that there exists a ballot making p win for weight list W, $|W| = n$. Let $W' := W + \{w_1', \ldots, w_k'\}$ for $k \geq 1, \sum_{i=1}^{k} w_i' \geq \max(W)$. By Lemma 3.7, $\sigma_n(p) > q(W)$. From Lemma 3.8 we get by induction that

$$q(W') \leq q(W) + \sum_{i=1}^{k} w_i' \cdot \frac{m-2}{2} \tag{10}$$

By Lemma 3.13 and (10) we get:

$$\begin{aligned}\max_{g\in G_{W'}}\{\sigma_{n+k}(g)\} &\le q(W') + \max(W')\cdot\frac{m-2}{2}\\ &\le q(W') + \sum_{i=1}^{k} w_i'\cdot\frac{m-2}{2}\\ &\le q(W) + \sum_{i=1}^{k} w_i'\cdot(m-2)\\ &< \sigma_n(p) + \sum_{i=1}^{k} w_i'\cdot(m-1) = \sigma_{n+k}(p)\end{aligned}$$

and hence, by Lemma 3.7 the algorithm will find a ballot making p win for set T' with weights W', and will return *true*. This completes the proof of Theorem 3.4. □

The following is an example where there is a manipulation for weight list W, but Algorithm 2 will find a manipulation only for weight list $W + \{w'\}$.

Example 3.14. In our example $W = \{1,1,1,1\}$, $w' = 1$, so we are actually talking about the special case of unweighted coalitions. Consider the set $C = \{p,1,2,3,4,5,6\}, m = |C| = 7$, $N = |S| = 5$. 3 voters in S voted $6 \succ 5 \succ 4 \succ 3 \succ 2 \succ p \succ 1$, and the other 2 voters in S voted $2 \succ 3 \succ 4 \succ 5 \succ 6 \succ p \succ 1$. When applying Algorithm 2 to this input, the voters in T will award the candidates with the following scores (we denote by $\alpha_j(c)$ the points that voter j gives to candidate c):[2]

Candidate $c \in C$	p	1	2	3	4	5	6
$\sigma_0(c)$	5	0	18	19	20	21	22
$\alpha_1(c)$	6	5	4	3	2	1	0
$\alpha_2(c)$	6	5	0	1	2	3	4
$\alpha_3(c)$	6	5	4	3	2	1	0
$\alpha_4(c)$	6	5	0	1	2	3	4
$\alpha_5(c)$	6	5	4	3	2	1	0

So the cumulative scores will be as follows:

Candidate $c \in C$	p	1	2	3	4	5	6
$\sigma_0(c)$	5	0	18	19	20	21	22
$\sigma_1(c)$	11	5	22	22	22	22	22
$\sigma_2(c)$	17	10	22	23	24	25	26
$\sigma_3(c)$	23	15	26	26	26	26	26
$\sigma_4(c)$	29	20	26	27	28	29	30
$\sigma_5(c)$	35	25	30	30	30	30	30

Note that after 4 stages, the algorithm still did not find a manipulation: $\sigma_4(p) = 29 < 30 = \sigma_4(6)$. However, if we change the votes of the third and fourth voters of T, then we find an appropriate ballot:

[2]We assumed here that when two candidates have the same scores up until a certain stage, the current voter will award fewer points to the candidate with lower index, but any tie-breaking rule will give the same results.

Candidate $c \in C$	p	1	2	3	4	5	6
$\sigma_0(c)$	5	0	18	19	20	21	22
$\alpha_1(c)$	6	5	4	3	2	1	0
$\alpha_2(c)$	6	5	0	1	2	3	4
$\alpha'_3(c)$	6	5	3	4	0	1	2
$\alpha'_4(c)$	6	5	3	1	4	2	0

Now the cumulative scores are:

Candidate $c \in C$	p	1	2	3	4	5	6
$\sigma_0(c)$	5	0	18	19	20	21	22
$\sigma_1(c)$	11	5	22	22	22	22	22
$\sigma_2(c)$	17	10	22	23	24	25	26
$\sigma'_3(c)$	23	15	25	27	24	26	28
$\sigma'_4(c)$	29	20	28	28	28	28	28

Evidently, for any $c \in C \setminus \{p\},\ \sigma'_4(p) = 29 > \sigma'_4(c)$.

3.2 Maximin

In this subsection, we show that Algorithm 1 also does well with respect to the Maximin rule.

Lemma 3.15. *Maximin is monotone in weights.*

Proof. Let X_S be the preference orders of the voters in S, and let X_T be the preference orders of the voters in T that make p win. We need to show that there are preference orders for $T' = T + \{v\}$ with weight list $W' = W + \{w\}$ where $w \geq 1$ is an integer, that make p win. Fix some order on voters in T. By definition, for all $c \in C \setminus \{p\}$, $\sigma_n(c) < \sigma_n(p)$. Let the additional voter of T' vote with p at the first place, and some arbitrary order on the other candidates. Then for all $c \in C \setminus \{p\}$, $\sigma_{n+1}(p) = \sigma_n(p) + w > \sigma_n(c) + w \geq \sigma_{n+1}(c)$, and so we got the ballot of votes of T' to make p win. □

Theorem 3.16. *In CCWM under Maximin, let C be the set of candidates with $p \in C$ the preferred candidate, and S the set of voters who already cast their votes. Let W be the weight list for the set T. Then:*

1. *If there is no ballot making p win the election, then Algorithm 1 will return* false.
2. *If there is a ballot making p win the election, then for the same instance with weight list W' s.t. $W' \supseteq W + W$ (i.e., W' contains two copies of W), Algorithm 1 will return* true.

Let us introduce some more notation. For candidates $g, g' \in C$ and $0 \leq j \leq n$ we denote by $N_j(g, g')$ the total weight of the voters after j stages (including the voters in S) that prefer g over g'. So $\sigma_j(g) = \min_{g' \in C \setminus \{g\}} N_j(g, g')$. We also denote for $g \in C$, $0 \leq j \leq n$:

$$\mathrm{MIN}_j(g) = \{h \in C \setminus \{g\} \mid N_j(g, h) = \sigma_j(g)\}.$$

In words, $\mathrm{MIN}_j(g)$ is the set of candidates that constitute the worst opponents of g in pairwise elections at stage j. Put differently, these are the candidates whose competition against g defines the Maximin score of g at stage j.

Fixing the set C, $p \in C$, and an order on the weight list W, we denote by $f(j)$ the maximal score of p's opponents distributed by Algorithm 1 after j stages:

$$f(j) = \max_{g \in C \setminus \{p\}} \sigma_j(g).$$

In Algorithm 1, p is always placed at the top of each preference, and so with each voter its score grows by the weight of this voter. In our next lemma we will put forward an upper bound on the growth rate of the scores of p's opponents.

Lemma 3.17. *Consider Algorithm 1 applied to the Maximin rule. Denote by w_j the weight of the j-th voter processed by the algorithm. Then for all $0 \leq j \leq n-2$, $f(j+2) \leq f(j) + \max\{w_{j+1}, w_{j+2}\}$.*

To intuitively see why the lemma implies Theorem 3.16, notice that if there are two copies of W, the score of p would increase by $2 \cdot \sum_{w \in W} w$, whereas by the lemma the score of the strongest candidate would increase by at most $\sum_{w \in W} w$. We now prove the lemma; the theorem will follow easily.

Proof. Let $0 \leq j \leq n-2$. Let $g \neq p$ be a candidate. By definition $\sigma_j(g) \leq f(j)$. We would like to show that $\sigma_{j+2}(g) \leq f(j) + \max\{w_{j+1}, w_{j+2}\}$. If $\sigma_{j+1}(g) \leq f(j)$, then $\sigma_{j+2}(g) \leq \sigma_{j+1}(g) + w_{j+2} \leq f(j) + \max\{w_{j+1}, w_{j+2}\}$, and we are done. So let us assume now that $\sigma_{j+1}(g) > f(j)$.

Define a directed graph $G = (V, E)$, where

$$V = \{g\} \cup \{x \in C \setminus \{p\} \mid x \text{ was ranked below } g \text{ in stage } j+1\},$$

and $(x, y) \in E$ iff $y \in \mathrm{MIN}_j(x)$. There is at least one outgoing edge from g in E, since otherwise there was $g' \in \mathrm{MIN}_j(g)$ that voter $j+1$ ranked above g, and then $\sigma_{j+1}(g) = \sigma_j(g) \leq f(j)$, a contradiction.

In addition, we claim that for all $x \in V \setminus \{g\}$ there is at least one outgoing edge from x in E. Indeed, otherwise there is $x' \in \mathrm{MIN}_j(x)$ that was ranked above g in stage $j+1$. Hence, we have that

$$\sigma_{j+1}(x) = \sigma_j(x) \leq f(j) < \sigma_{j+1}(g).$$

This implies that Algorithm 1 should have ranked x above g in stage $j+1$, which is a contradiction to the fact that $x \in V \setminus \{g\}$.

For $x \in V$, denote by $V(x)$ all the vertices y in V such that there exists a directed path from x to y. Denote by $G(x)$ the sub-graph of G induced by $V(x)$. It is easy to see that $G(g)$ contains at least one cycle. Let U be one such cycle. Let $g' \in U$ be the vertex that was ranked highest among the vertices of U in stage $j+1$. Let g'' be the vertex before g' in the cycle: $(g'', g') \in U$. Since g'' was ranked below g' at stage $j+1$, it follows that $\sigma_{j+1}(g'') = \sigma_j(g'') \leq f(j)$.

Suppose, for contradiction, that $\sigma_{j+2}(g) > f(j) + \max\{w_{j+1}, w_{j+2}\}$. g was ranked by $j+2$ at place t^*. Then g'' was ranked by $j+2$ above t^*, since otherwise when we had reached the place t^*, we would pick g'' (with score $\sigma_{j+2}(g'') \leq f(j) + w_{j+2} < \sigma_{j+2}(g)$) instead of g—a contradiction.

Denote by X_1 all the vertices in $V(g)$ that have an outgoing edge to g'' in $G(g)$. For all $x \in X_1$, $g'' \in \mathrm{MIN}_j(x)$, i.e., $\sigma_j(x) = N_j(x, g'')$. All $x \in X_1$ were ranked by $j+2$ above g, since otherwise, if there was $x \in X_1$, s.t. until the place t^* it still was not added to the preference list, then when evaluating its score on place t^*, we would get: $\sigma_{j+2}(x) \leq N_{j+2}(x, g'') = N_{j+1}(x, g'') \leq N_j(x, g'') + w_{j+1} = \sigma_j(x) + w_{j+1} < \sigma_{j+2}(g)$, and so we would put x instead of g.

Denote by X_2 all the vertices in $V(g)$ that have an outgoing edge to some vertex $x \in X_1$. In the same manner we can show that all the vertices in X_2 were ranked in stage $j+2$ above g. We continue in this manner, by defining sets $X_3, \ldots$, where the set X_l contains all vertices in $V(g)$

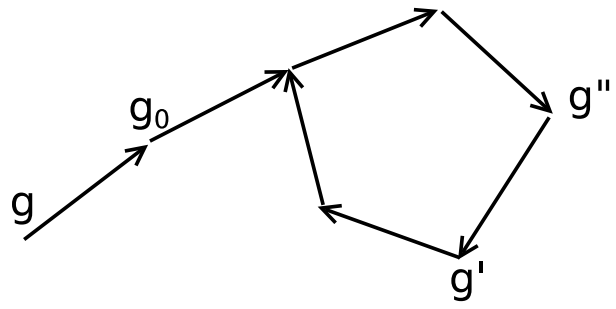

Figure 1: The induced sub-graph $G(g)$

that have an outgoing edge to some vertex in X_{l-1}; the argument above shows that all elements of these sets are ranked above g in stage $j+2$. As there is a path from g to g'' in $G(g)$, we will eventually reach g in this way, i.e., there is some l such that X_l contains a vertex g_0 with an edge from g to g_0 (see Figure 1).

Thus,

$$\begin{aligned}\sigma_{j+2}(g) &\leq N_{j+2}(g,g_0) = N_{j+1}(g,g_0) \leq N_j(g,g_0) + w_{j+1} \\ &= \sigma_j(g) + w_{j+1} \leq f(j) + \max\{w_{j+1}, w_{j+2}\} \\ &< \sigma_{j+2}(g),\end{aligned}$$

a contradiction. □

We are now ready to prove Theorem 3.16.

Proof. We prove part 1. Algorithm 1 returns *true* only if it constructs a (valid) ballot that makes p win, and thus if there is no ballot making p win, Algorithm 1 will return *false*.

We now prove part 2. Suppose that there exists a ballot Z_T making p win for weight list $W = \{w_1, \ldots, w_n\}$. Let $\sigma'_j(g)$ be the scores implied by Z_T. Then:

$$f(0) < \sigma'_n(p) \leq \sigma_0(p) + \sum_{i=1}^{n} w_i \tag{11}$$

Let $W' = W + W + X$, where X is some list of weights (possibly empty). We need to show that $\sigma_{|W'|}(p) > f(|W'|)$. In Algorithm 1, after sorting the weights of W', the equal weights of two copies of W will be adjacent, i.e., the order of weights in W' will be of the form:

$$x_1, \ldots, x_{q_1}, w_1, w_1, x_{q_1+1}, \ldots, x_{q_2}, w_2, w_2, \ldots, w_n, w_n, x_{q_n+1}, \ldots, x_{|X|}.$$

By Lemma 3.17, one can prove by induction that:

$$f(|W'|) \leq f(0) + \sum_{i=1}^{|X|} x_i + \sum_{i=1}^{n} w_i \tag{12}$$

And so by (11) and (12) we have:

$$\sigma_{|W'|}(p) = \sigma_0(p) + \sum_{i=1}^{|X|} x_i + 2\sum_{i=1}^{n} w_i > f(0) + \sum_{i=1}^{|X|} x_i + \sum_{i=1}^{n} w_i \geq f(|W'|)$$

□

In Appendix B we give a simple algorithm, which is tailor-made for Maximin, and also enjoys the implications of Theorem 3.16. However, this algorithm does not extend to other voting rules, as Algorithm 1 does. Moreover, we believe that Algorithm 1 does better when it comes to unweighted manipulation (see Sections 4 and 6).

3.3 Plurality with Runoff

In this subsection we present a heuristic algorithm for the CCWM problem in Plurality with Runoff. The algorithm receives as a parameter a size of window $0 \leq u \leq \max(W)$ where it can give a wrong answer. Its running time depends on the size of its input and on u (see below). We begin by noting:

Lemma 3.18. *Plurality with Runoff is monotone in weights.*

Proof. Let C be the candidates, $p \in C$ is the preferred candidate, S is the set of truthful voters, and W are the weights for manipulators of T. Suppose that there is a ballot of votes of T that makes p win the election. We need to show that there is a ballot making p win for the set $W' = W + \{w\}$, where $w \geq 1$. Let g be the candidate that proceeds with p to the second round in the winning ballot for W. Let the additional voter vote $p \succ \ldots$. Then the plurality score of p and g will not decrease, while the plurality score of any other candidate will remain the same, and so p and g will proceed to the next round in the new ballot as well. In the second round p will beat g in the new ballot, since the total weight of the voters who prefer p to g increased, while the total weight of voters who prefer g to p remained the same. Thus, p will win the election in the new ballot. □

We will now give an informal description of the algorithm. We go over all the candidates other than p. To each candidate g we try to assign the voters with minimal total weight, such that if these voters place g first, g continues to the second round; the rest of the voters rank p first. If we succeeded in this way to make g and p survive the first round, and in the second round p beats g, then we found a valid ballot for making p win the election. If no candidate g was found in this way, then we report that there is no ballot.

A formal description of this algorithm, Algorithm 3, is given below. The following additional notations are required. Denote by $\beta_X(g)$ the plurality score of g from voter set X (i.e., the sum of weights of the voters in X that put g at the top of their preferences). We also use $N_X(g, g') = \sum_{v \in U} w_v$, where U is the set of all the voters in X that prefer g to g', and w_v is the weight of voter v. Finally, for $g, g' \in C$ we denote $g \gg g'$ if a tie between g and g' is broken in favor of g.

Remark 3.19. In Algorithm 3 we do not rely on the assumption that for all $g \neq p$, $g \gg p$. In fact, the algorithm can deal with any tie-breaking mechanism such that for every two distinct candidates x and y, either $x \gg y$ or $y \gg x$, regardless of how the manipulators cast their votes. An example of such a tie-breaking mechanism is to favor candidates with smaller indices, according to some order on the candidates. This is not necessarily a reasonable way to break ties in, say, political elections, but roughly speaking it is more general than asking that p be a unique winner, the assumption underlying our previous results.

More precisely, Plurality with Runoff differs from Borda and Maximin in the sense that it has two different rounds, and therefore two different "scores". Hence, the unique winner model can be interpreted ambiguously in this context. If we always break ties against p (the algorithm supports this), p might be tied against another candidate for the second ticket to the second round, and lose, whereas under another interpretation p would have advanced to the second round, and would have won the second round by a vast majority, thus becoming a unique winner.

Algorithm 3 Decides CCWM in Plurality with Runoff with desired accuracy

procedure PLURALITY-WITH-RUNOFF(C, p, X_S, W_S, W, u) ▷ X_S is the set of preferences of voters in S, W_S are the weights of voters in S, $W = \{w_1, \ldots, w_n\}$ are the weights of voters in T, u is the size of error window
  for g in $C \setminus \{p\}$ **do** ▷ Go over candidates in $C \setminus \{p\}$
    if there exists $g' \in \operatorname{argmax}_{g' \in C \setminus \{p\}} \beta_S(g')$, $g' \neq g$ s.t. $g' \gg g$ **then**
      $\lambda_g \leftarrow \max_{g' \in C \setminus \{p\}} \beta_S(g') - \beta_S(g) + 1$
    else
      $\lambda_g \leftarrow \max_{g' \in C \setminus \{p\}} \beta_S(g') - \beta_S(g)$
    end if
    if $\lambda_g > \sum_{i=1}^{n} w_i$ **then** ▷ If we cannot make g pass to the next round
      continue ▷ Go to the next candidate in the main loop
    end if
    $x \leftarrow$ SUBSET-OF-WEIGHTS-APPROXIMATE(W, λ_g, u)
      ▷ $x \in \{0,1\}^n$ minimizes $\{\sum_{j=1}^{n} w_j x_j \mid \sum_{j=1}^{n} w_j x_j \geq \lambda_g, \forall j, x_j \in \{0,1\}\}$
    All the voters j s.t. $x_j = 1$ vote $g \succ \ldots$ ▷ Order of candidates except g is arbitrary
    All the voters j s.t. $x_j = 0$ vote $p \succ \ldots$
    if $\exists g' \in C \setminus \{p, g\}$ s.t. $\big(\beta_S(g') > \beta_S(p) + \beta_T(p)\big)$
      or $\big(\beta_S(g') = \beta_S(p) + \beta_T(p)$ and $g' \gg p\big)$ **then**
      continue ▷ p does not pass to next round
    end if
    if $\big(N_{S \cup T}(p, g) > N_{S \cup T}(g, p)\big)$ or $\big(N_{S \cup T}(p, g) = N_{S \cup T}(g, p)$ and $p \gg g\big)$ **then**
      return true ▷ p beats g in the second round
    else
      continue
    end if
  end for
  return false ▷ No appropriate g was found
end procedure

procedure SUBSET-OF-WEIGHTS-APPROXIMATE(W, λ_g, u) ▷ $W = \{w_1, \ldots, w_n\}$ are the weights of voters in T, λ_g is the minimum total sum of desired weights, u is the size of error window
  Check that $0 \leq u \leq \max(W)$
  $k_u \leftarrow \left\lfloor \frac{u}{2n} \right\rfloor + 1$
  Solve by dynamic prog.: $\max\{\sum_{j=1}^{n} \left\lfloor \frac{w_j}{k_u} \right\rfloor \overline{x}_j \mid \sum_{j=1}^{n} w_j \overline{x}_j \leq \sum_{j=1}^{n} w_j - \lambda_g, \forall j,\ \overline{x}_j \in \{0,1\}\}$
  Let $\overline{x} \in \{0,1\}^n$ be the vector that maximizes the above sum
  return $\vec{1} - \overline{x}$ ▷ $\vec{1}$ is the vector of n 1's
end procedure

In the next theorem we prove the correctness of Algorithm 3, and analyze its time complexity. We will see that for getting an exact answer ($u = 0$), we will need running time which is polynomial in $\max(W)$ and the rest of the input. As the weights in W are specified in binary representation, this requires exponential time. However, when the size of the error window increases, the complexity decreases, so for $u = \Omega(\frac{\max(W)}{\log(\max(W))})$ the complexity of the algorithm is polynomial in its input.

Theorem 3.20. *In CCWM under Plurality with Runoff, let C be the set of candidates with $p \in C$ the preferred candidate, and S be the set of voters who already cast their votes. Let W be the weight list for the set T, and let $u \geq 0$ be the error window. Then:*

1. *If there is no ballot making p win the election, then Algorithm 3 will return* false.
2. *If there is a ballot making p win the election, then for the same problem with voter set $T' = T + \{v_{n+1}, \ldots, v_{n+l}\}$ with weight list $W' = W + \{w_{n+1}, \ldots, w_{n+l}\}$, where $l \geq 0, \sum_{j=1}^{l} w_{n+j} \geq u$, Algorithm 3 will return* true.
3. *On input C, p, X_S, W_S, W, u, where $|C| = m, |S| = N, |W| = n$, u is an integer, s.t. $0 \leq u \leq \max(W)$, the running time of Algorithm 3 is polynomial in $m, N, \log(\max(W_S)), n$ and $\frac{\max(W)}{u+1}$.*

Proof. We start with part 1. Note that

$$\overline{x} = (\overline{x}_1, \ldots, \overline{x}_n) \text{ satisfies } \sum_{j=1}^{n} w_j \overline{x}_j \leq \sum_{j=1}^{n} w_j - \lambda_g \iff x = \vec{1} - \overline{x} \text{ satisfies } \sum_{j=1}^{n} w_j x_j \geq \lambda_g, \quad (13)$$

where λ_g is defined in Algorithm 3 as the total weight of the votes g needs in order to proceed to the second round, and $\overline{x}$ is the binary vector of length n computed in the algorithm's subroutine. Thus when voters corresponding to weights returned by the function SUBSET-OF-WEIGHTS-APPROXIMATE() (see Algorithm 3) vote $g \succ \ldots$, they ensure that g proceeds to the second round. It is easy to see that whenever Algorithm 3 returns *true*, it actually finds a (valid) ballot making p win the election, and so if there is no such ballot, then the algorithm will return *false*.

We now move on to part 2. Let A_W be an instance of the problem with weight list W. Suppose that there exists ballot X_T of votes in T s.t. combined with preferences X_S of voters of S, it makes p win the election in A_W. We will denote by $\beta'_Y(g)$ the plurality score of g from voter set Y under the preferences $X_S \cup X_T$. Also, we denote $N'_Y(g, g') = \sum_{v \in U_{X_S \cup X_T}} w_v$, where $U_{X_S \cup X_T}$ is the set of all the voters in Y that prefer g to g' under $X_S \cup X_T$. Let $0 \leq u \leq \max(W)$, $W' = W + \{w_{n+1}, \ldots, w_{n+l}\}$, where $l \geq 0$, $\sum_{j=1}^{l} w_{n+j} \geq u$. We need to show that Algorithm 3 will return *true* on the input W', u.

There is a candidate $g \neq p$ that passes together with p to the second round when applying the preferences X_T together with X_S on A_W, and thus for each candidate $g' \notin \{p, g\}$ and $c \in \{p, g\}$, if $c \gg g'$, then $\beta'_S(c) + \beta'_T(c) \geq \beta'_S(g') + \beta'_T(g')$, and if $g' \gg c$, then $\beta'_S(c) + \beta'_T(c) > \beta'_S(g') + \beta'_T(g')$. Also,

$$\beta'_T(p) + \beta'_T(g) \leq \sum_{j=1}^{n} w_j \quad (14)$$

Now consider Algorithm 3 applied to $A_{W'}$. If it does not reach g in the main loop, then it will exit earlier returning "true", meaning that it will find a desired ballot making p win. Otherwise, it will reach the candidate g. λ_g is the minimal sum of weights that ensures that g will continue to the second round, and hence

$$\lambda_g \leq \beta'_T(g) \leq \sum_{j=1}^{n} w_j \leq \sum_{j=1}^{n+l} w_j \quad (15)$$

We will reach the function SUBSET-OF-WEIGHTS-APPROXIMATE(), and enter it with arguments W', λ_g and u. By (13), the vector $x = (x_1, \ldots, x_{n+l})$ returned by SUBSET-OF-WEIGHTS-APPROXIMATE() satisfies $\sum_{j=1}^{n+l} w_j x_j \geq \lambda_g$, and so g will continue to the next round. Now we show that p will also continue to the next round. Denote by H the maximization problem

$$\begin{aligned} &\max \sum_{j=1}^{n+l} w_j \overline{x}_j \\ &\text{s.t. } \sum_{j=1}^{n+l} w_j \overline{x}_j \leq \sum_{j=1}^{n+l} w_j - \lambda_g \\ &\overline{x}_j \in \{0,1\}, \text{ for } j = 1, \ldots, n+l \end{aligned} \tag{16}$$

Let $J^* = \{j \mid \overline{x}_j = 1, \overline{x} = (\overline{x}_1, \ldots, \overline{x}_{n+l}) \text{ is the optimal solution to } H\}$. Denote $P^* = \sum_{j \in J^*} w_j$. Let $H(k)$ be the scaled version of the above maximization problem:

$$\begin{aligned} &\max \sum_{j=1}^{n+l} \Big\lfloor \frac{w_j}{k} \Big\rfloor \overline{x}_j \\ &\text{s.t. } \sum_{j=1}^{n+l} w_j \overline{x}_j \leq \sum_{j=1}^{n+l} w_j - \lambda_g \\ &\overline{x}_j \in \{0,1\}, \text{ for } j = 1, \ldots, n+l \end{aligned} \tag{17}$$

Let $J(k) = \{j \mid \overline{x}_j = 1, \overline{x} = (\overline{x}_1, \ldots, \overline{x}_{n+l}) \text{ is the optimal solution to } H(k)\}$. Let $P(k) = \sum_{j \in J(k)} w_j$. Now, $\overline{x} = (\overline{x}_1, \ldots, \overline{x}_{n+l})$ which we obtained in SUBSET-OF-WEIGHTS-APPROXIMATE() satisfies, for $k_u = \big\lfloor \frac{u}{2(n+l)} \big\rfloor + 1$:

$$\begin{aligned} \sum_{j=1}^{n+l} w_j \overline{x}_j &= \sum_{j \in J(k_u)} w_j \geq \sum_{j \in J(k_u)} k_u \Big\lfloor \frac{w_j}{k_u} \Big\rfloor \geq \sum_{j \in J^*} k_u \Big\lfloor \frac{w_j}{k_u} \Big\rfloor \geq \sum_{j \in J^*} (w_j - (k_u - 1)) \\ &= \sum_{j \in J^*} w_j - (k_u - 1)|J^*| = P^* - (k_u - 1)|J^*| \end{aligned} \tag{18}$$

Hence, the vector $x = \vec{1} - \overline{x}$ returned by the function, satisfies:

$$\begin{aligned} \sum_{j=1}^{n+l} w_j x_j &\leq \sum_{j=1}^{n+l} w_j - P^* + (k_u - 1)|J^*| \\ &\leq \sum_{j=1}^{n+l} w_j - P^* + \Big\lfloor \frac{u}{2(n+l)} \Big\rfloor (n+l) \\ &\leq \sum_{j=1}^{n+l} w_j - P^* + \Big\lfloor \frac{u}{2} \Big\rfloor \end{aligned} \tag{19}$$

By definition of P^*, we get:

$$\begin{aligned}\sum_{j=1}^{n+l} w_j - P^* &= \min\{\sum_{j=1}^{n+l} w_j x_j \mid \sum_{j=1}^{n+l} w_j x_j \geq \lambda_g, x_j \in \{0,1\}, 1 \leq j \leq n+l\} \\ &\leq \min\{\sum_{j=1}^{n} w_j x_j \mid \sum_{j=1}^{n} w_j x_j \geq \lambda_g, x_j \in \{0,1\}, 1 \leq j \leq n\} \\ &\leq \beta'_T(g)\end{aligned} \tag{20}$$

Combining (19) and (20), we get that for vector x returned by the function SUBSET-OF-WEIGHTS-APPROXIMATE():

$$\beta_{T'}(g) = \sum_{j=1}^{n+l} w_j x_j \leq \beta'_T(g) + \left\lfloor \frac{u}{2} \right\rfloor \tag{21}$$

In the algorithm, all the voters j s.t. $x_j = 0$ will vote $p \succ \dots$, and so we will have

$$\begin{aligned}\beta_{T'}(p) &= \sum_{j=1}^{n+l} w_j - \sum_{j=1}^{n+l} w_j x_j \geq \sum_{j=1}^{n} w_j + u - (\beta'_T(g) + \left\lfloor \frac{u}{2} \right\rfloor) \\ &= \sum_{j=1}^{n} w_j - \beta'_T(g) + \left\lceil \frac{u}{2} \right\rceil \geq \beta'_T(p) + \left\lceil \frac{u}{2} \right\rceil\end{aligned} \tag{22}$$

For any candidate c such that $c \notin \{p, g\}$, c was never ranked at the top of the preference lists by Algorithm 3, and so $\beta_{S \cup T'}(c) = \beta_S(c) \leq \beta'_{S \cup T}(c)$. On the other hand, by (22),

$$\beta_{S \cup T'}(p) = \beta_S(p) + \beta_{T'}(p) \geq \beta_S(p) + \beta'_T(p) + \left\lceil \frac{u}{2} \right\rceil = \beta'_{S \cup T}(p) + \left\lceil \frac{u}{2} \right\rceil \geq \beta'_{S \cup T}(p).$$

Recall that p beats c in the first round under X_T. It follows that p beats c in the first round under Algorithm 3, and so p will continue to the next round.

We now prove that p beats g in the next round. If $g \gg p$, then in the winning ballot X_T, $N'_S(p,g) + N'_T(p,g) > N'_S(g,p) + N'_T(g,p)$, otherwise $N'_S(p,g) + N'_T(p,g) \geq N'_S(g,p) + N'_T(g,p)$. From (21) we get:

$$N_{T'}(g,p) = \beta_{T'}(g) \leq \beta'_T(g) + \left\lfloor \frac{u}{2} \right\rfloor \leq N'_T(g,p) + \left\lfloor \frac{u}{2} \right\rfloor \tag{23}$$

Thus, from (23):

$$N_{T'}(p,g) = \sum_{j=1}^{n+l} w_j - N_{T'}(g,p) \geq \sum_{j=1}^{n} w_j + u - (N'_T(g,p) + \left\lfloor \frac{u}{2} \right\rfloor) = N'_T(p,g) + \left\lceil \frac{u}{2} \right\rceil \tag{24}$$

So, for $g \gg p$ we get

$$\begin{aligned}N'_S(p,g) + N_{T'}(p,g) &\geq N'_S(p,g) + N'_T(p,g) + \left\lceil \frac{u}{2} \right\rceil \\ &> N'_S(g,p) + N'_T(g,p) + \left\lfloor \frac{u}{2} \right\rfloor \\ &\geq N'_S(g,p) + N_{T'}(g,p)\end{aligned} \tag{25}$$

In the same way, for $p \gg g$ we get

$$N'_S(p,g) + N_{T'}(p,g) \geq N'_S(g,p) + N_{T'}(g,p) \tag{26}$$

Therefore, p wins the second round of the election, and hence the entire election; the algorithm will return *true*.

Next, we prove part 3. Using the notation of the previous part, let $P(k)$ be the maximum sum of weights from $W = \{w_1, \ldots, w_n\}$, solving the scaled maximization problem $H(k)$.[3] There is a well-known dynamic programming algorithm solving the knapsack problem $H(k)$ in time $O(nP(k))$ (see, e.g. [19, chapter 9]). Furthermore, $P(k) \leq \sum_{j=1}^{n} \left\lfloor \frac{w_j}{k} \right\rfloor \leq n \left\lfloor \frac{\max(W)}{k} \right\rfloor \leq n\frac{\max(W)}{k}$. The algorithm sets $k_u = \left\lfloor \frac{u}{2n} \right\rfloor + 1 \geq \frac{u+1}{2n}$, and so we have:

$$P(k_u) \leq n\frac{\max(W)}{k_u} \leq n\frac{\max(W)}{\frac{u+1}{2n}} = 2n^2\frac{\max(W)}{u+1} \tag{27}$$

Thus we can solve $H(k_u)$ in $O(nP(k_u)) = O(n^3 \cdot \frac{\max(W)}{u+1})$. It is easy to see that all the other steps of Algorithm 3 are polynomial in its inputs; hence, the proof is completed. □

4 Unweighted Coalitional Manipulation

In this section, we discuss the application of the results given above to unweighted coalitional manipulation (the CCUM problem), and present a new theorem. We will see that some of our theorems can be translated into approximation (in the classical sense) results in this natural setting.

It is known that the CCUM problem is tractable—with respect to any voting rule that can be computed in polynomial time—when the number of candidates is constant [7]. However, to the best of our knowledge (at the time of submission) there are no results regarding the complexity of the problem when the number of candidates is not constant, except for the cases of STV and Second Order Copeland where CCUM is hard even when there is only a single manipulator [3, 2]. We conjecture that CCUM in Borda and Maximin is $\mathcal{NP}$-complete.

In the context of unweighted manipulation, one can consider the following optimization problem:

Definition 4.1. In the Constructive Coalitional Unweighted Optimization (CCUO) problem, we are given the (unweighted) votes of the truthful voters. We must find the minimum number of manipulators needed in order to make p win (i.e., the minimum number of manipulators that can cast their (unweighted) votes in a way that makes p win).

Then, our theorems almost directly imply the following corollary:

Corollary 4.2.

1. *Algorithm 2 approximates CCUO in Borda up to an additive error of 1.*
2. *Algorithm 1 is a 2-approximation algorithm for CCUO in Maximin.*

Proof. It is enough to show that the minimum number of manipulators needed in order to make p win, in Borda and Maximin, must be polynomial in the rest of the input. Indeed, in this case we can apply brute-force search using Algorithms 2 and 1, respectively, in order to approximate the answer. In other words, we run the algorithm once for every number of manipulators $k \in \{0, \ldots, p(n)\}$ for some polynomial p. The minimum k which gives a *true* answer in Borda (resp., Maximin) is guaranteed to be larger by at most 1 (resp., twice as large) than the optimal answer by Theorem 3.4 (resp., Theorem 3.16).

So, it is sufficient to prove the following two Lemmata.

[3] We slightly abuse notation here, as we defined the optimization problems for weight set $W' = \{w_1, \ldots, w_{n+l}\}$, but the definition for the set W is analogous.

Lemma 4.3. *Let $\langle \alpha_1, \ldots, \alpha_m \rangle$ be a scoring protocol where $\alpha_1 - \alpha_m > 0$. In the CCUO problem, let $C = \{c_1, \ldots, c_{m-1}, p\}$ be the candidates, S be the set of the truthful voters, $|S| = N$, and n^* be the minimal number of manipulators such that there exists a ballot making p win. Then $n^* \leq (N+1)(m-1)$.*

Proof. We show that there exists a ballot making p win for $n^* = (N+1)(m-1)$. Let the manipulator $1 \leq j \leq (N+1)(m-1)$ vote $p \succ \ldots \succ c_{i+1}$, where $j-1 \equiv i \mod (m-1)$, $0 \leq i \leq m-2$, and the rest of the order is arbitrary. With every $m-1$ voters the difference between the scores of any candidate c_j and p decreases by at least $\alpha_1 - \alpha_m$. Moreover, for any $1 \leq j \leq m-1$, $\sigma_0(c_j) \leq \sigma_0(p) + N(\alpha_1 - \alpha_m)$, and so we get: $\sigma_{(N+1)(m-1)}(c_j) \leq \sigma_{(N+1)(m-1)}(p) - (\alpha_1 - \alpha_m) < \sigma_{(N+1)(m-1)}(p)$. Hence, p will win the election. □

Lemma 4.4. *Consider the CCUO problem in the Maximin protocol. In the notation of Lemma 4.3, $n^* \leq N + 1$.*

Proof. We show that there exists a ballot making p win for $n^* = N + 1$. Let every manipulator vote $p \succ \ldots$. Then for every candidate c_j we get: $\sigma_{N+1}(c_j) \leq N_{N+1}(c_j, p) \leq N$. Moreover, for any candidate $c_j \neq p$, $N_{N+1}(p, c_j) \geq N + 1$, and so $\sigma_{N+1}(p) \geq N + 1$. Hence we get for every candidate c_j, $\sigma_{N+1}(c_j) < \sigma_{N+1}(p)$, implying that p will win. □

This concludes the proof of the corollary. □

On the other hand, we have the following results:

Corollary 4.5. *Algorithm 3 efficiently solves the CCUM problem in Plurality with Runoff.*

Proof. Follows as a special case of Theorem 3.20, where the error window is $u = 0$, the number of additional voters is $l = 0$, and all the weights equal 1. □

Theorem 4.6. *Algorithm 2 efficiently solves the CCUM problem in Veto.*

A short discussion is in order regarding CCUM in Veto. Indeed, this problem can be solved efficiently by a trivial algorithm. The fact that each manipulator can veto a single candidate may be interpreted as follows: each manipulator picks one candidate such that the score of p increases by 1 relative to that candidate without changing with respect to any other candidate. Thus, we simply have to count the number of manipulators needed to guarantee that p has more points than any other candidate. Formally, if we denote by $\sigma_0(c)$ the score of candidate c based on the votes in S, then clearly there exists a vote for T making p win if and only if

$$\sum_{c \in C \setminus \{p\}} \max(\sigma_0(c) - \sigma_0(p) + 1, 0) \leq |T|.$$

In the context of CCUM in Veto, Algorithm 2 is, in a sense, an instantiation of the simple scheme described above. However, our direct proof of Theorem 4.6, given in Appendix C, is a simpler, but analogous, version of the proof of Theorem 3.4.

Finally, note that Corollary 4.5 and Theorem 4.6 imply that CCUO in Plurality with Runoff and Veto is also in $\mathcal{P}$.

5 Relation to Work on Frequency of Manipulation

At this point, we would like to give a more in-depth exposition of previous work regarding frequency of manipulation, and connect it with this paper.

An interesting approach to the abovementioned issue was presented by Conitzer and Sandholm [6]. They noticed that an election instance can be manipulated efficiently if it satisfies two properties: weak monotonicity—a property which is satisfied by many prominent voting rules—and another, more arguable property: the manipulators must be able to make one of *exactly* two candidates win the election. Conitzer and Sandholm empirically showed that the second property holds with high probability in different standard voting rules. This empirical validation was carried out only with respect to small coalitions of voters and skewed distributions over election instances.

Procaccia and Rosenschein [23] leveraged some of the intuitions provided by Conitzer and Sandholm. They analyzed the probability of the manipulators being able to affect the outcome of the election (i.e., make one of *at least* two candidates win), conditioned on the fraction of manipulators. They found that for quite general distributions over election instances, if $n = o(\sqrt{N})$, the manipulators cannot affect the outcome with high probability; the opposite is true if $n = \omega(\sqrt{N})$. These results extended previous work on asymptotic strategy proofness [1, 28].

Another result was recently presented by Friedgut, Kalai and Nisan [13]. They showed that a single manipulator can find a manipulation with relatively good probability by simply switching to randomly chosen linear preferences (in particular, high probability of success can be achieved by repeating this process a polynomial number of times). This is true provided the voting rule in question is "far from dictatorial" in some well-defined sense. The proof of this theorem is beautiful, but sadly the current proof only works for at most 3 candidates.

Most closely related to this paper is another work by Procaccia and Rosenschein [24], who have attempted to establish a framework which would enable showing that manipulations are typically *easy*. For this purpose, they have defined the notion of *junta distributions*, which are intuitively (and arguably) "hard to manipulate", over election instances in the coalitional manipulation setting. Moreover, they have defined a voting rule to be susceptible to manipulation if there is an algorithm that decides CCWM with high probability of success, when the instances are distributed according to a junta distribution. The rationale is that if there is an algorithm that does well with respect to these especially hard junta distributions, it would also do well with respect to other reasonable distributions.

Procaccia and Rosenschein's main result is that scoring rules are susceptible to manipulation, according to the foregoing definition. Technically, Procaccia and Rosenschein's result is in fact a very loose bound on the window of error of Algorithm 2. Although their analysis holds for any scoring rule, it suffers from two major shortcomings. First, it is much looser than the one given in this paper, and consequently does not allow for corollaries regarding unweighted coalitional manipulation. In contrast, our result regarding Borda is far stronger, since the window of error is much more accurately characterized. The stronger result allowed, e.g., for Corollary 4.2. A second major disadvantage of Procaccia and Rosenschein's analysis is that it only applies to a constant number of candidates, i.e., $m = O(1)$. However, since the result in Procaccia and Rosenschein deals with scoring rules in general and here the only Scoring rules we deal with are Borda and Veto, neither result strictly subsumes the other.

Erdelyi et al. [11] discuss the notion of junta distributions at length. They show that the idea of junta distributions, when applied to the SAT problem, is not sufficient to classify hard-to-decide distributions. Their work is inconclusive, however, when it comes to the application of junta distributions to hardness of manipulation problems.

Still, it seems that at this point we lack a link between a mathematical framework dealing with

frequency of manipulation, and hardness on average. In light of this, we shall shortly consider the intuitive frequency of manipulation implications of our results, without being too formal. Our theorems imply that our algorithms err on only very specific configurations of the voters' weights. It might be productive to imagine points on the real line as representing the total weight of T. In the case of Borda, then, our algorithm would give a correct negative answer on all points to the left of some point x, and a correct positive answer on all points to the right of $x + \max W$. The range between x and $x + \max W$ is the window of error. This is a simplification of the situation, but a useful one nonetheless.

Now, intuitively consider some "reasonable" distribution over the instances of the CCUM problem (such that weights are randomly selected). The fact that the distribution is "reasonable" guarantees that the manipulators' total weight is distributed over a large range. Therefore, the probability of hitting the tiny window of error is extremely small. This (once again, intuitively) means that with high probability, our algorithms would correctly decide the manipulation problem.

6 Discussion

We would like to devote this final section to a short discussion regarding extensions of our results, and their applications to other voting rules.

We have noted above (and elaborate on, in Appendix A) that Copeland's rule is not monotone in weights. This seems to preclude the type of analysis which we have presented here. Nevertheless, it might be possible to obtain similar results if one endows the manipulators with the option to abstain from voting. In this way, any voting rule must be monotone in weights, as additional manipulators can always abstain. This is also not a major departure from our model, where the manipulators can coordinate their votes; it is only natural to assume that they can also agree not to vote at all.

The prominent Single Transferable Vote (STV) rule is one that we have not discussed above. In STV, the election proceeds in rounds; each voter casts his vote for the candidate he ranks first among the remaining candidates; the candidate with lowest score is eliminated. It is difficult to apply our approach to STV, for two reasons. First, it does not have a notion of score (but this is also true for Plurality with Runoff). Second, it is a very hard voting rule to manipulate. Indeed, it is well known that STV is hard to manipulate even for a single manipulator [2]. However, in theory STV is amenable to our type of analysis; this remains a fascinating direction for future research.

Finally, we conjecture that our analysis of the performance of Algorithm 1 with respect to CCUM in Maximin is not tight: it might be possible to lower the bound from 2 to 3/2 by using a close variant of the algorithm.

7 Acknowledgments

The authors would like to thank Vincent Conitzer for excellent comments on a draft of this paper, and in particular for pointing out the alternative 2-approximation algorithm for Maximin given in Appendix B. The authors also thank the anonymous AIJ reviewers for insightful comments. This work was partially supported by Israel Science Foundation grant #898/05.

References

[1] E. Baharad and Z. Neeman. The asymptotic strategyproofness of scoring and Condorcet consistent rules. *Review of Economic Design*, 4:331–340, 2002.

[2] J. Bartholdi and J. Orlin. Single transferable vote resists strategic voting. *Social Choice and Welfare*, 8:341–354, 1991.

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

[4] E. H. Clarke. Multipart pricing of public goods. *Public Choice*, 11:17–33, 1971.

[5] V. Conitzer and T. Sandholm. Universal voting protocol tweaks to make manipulation hard. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI'03)*, pages 781–788, Acapulco, Mexico, 2003.

[6] V. Conitzer and T. Sandholm. Nonexistence of voting rules that are usually hard to manipulate. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI'06)*, pages 627–634, Boston, 2006.

[7] V. Conitzer, T. Sandholm, and J. Lang. When are elections with few candidates hard to manipulate? *Journal of the ACM*, 54(3):1–33, 2007.

[8] E. Elkind and H. Lipmaa. Hybrid voting protocols and hardness of manipulation. In *ISAAC*, Lecture Notes in Computer Science, pages 206–215. Springer-Verlag, 2005.

[9] E. Elkind and H. Lipmaa. Small coalitions cannot manipulate voting. In *FC*, Lecture Notes in Computer Science. Springer-Verlag, 2005.

[10] E. Ephrati and J. S. Rosenschein. A heuristic technique for multiagent planning. *Annals of Mathematics and Artificial Intelligence*, 20:13–67, 1997.

[11] G. Erdélyi, L. A. Hemaspaandra, J. Rothe, and H. Spakowski. On approximating optimal weighted lobbying, and frequency of correctness versus average-case polynomial time. In *Fundamentals of Computation Theory*, volume 4639 of *Lecture Notes in Computer Science*, pages 300–311. Springer-Verlag, 2007.

[12] P. Faliszewski, E. Hemaspaandra, and L. A. Hemaspaandra. The complexity of bribery in elections. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI 2006)*, Boston, 2006.

[13] E. Friedgut, G. Kalai, and N. Nisan. Elections can be manipulated often. In *Proceedings of the Forty-Ninth Conference on Foundations of Computer Science (FOCS'08)*, 2008. To appear.

[14] S. Ghosh, M. Mundhe, K. Hernandez, and S. Sen. Voting for movies: the anatomy of a recommender system. In *Proceedings of the Third Annual Conference on Autonomous Agents*, pages 434–435, Seattle, 1999.

[15] A. Gibbard. Manipulation of voting schemes. *Econometrica*, 41:587–602, 1973.

[16] T. Groves. Incentives in teams. *Econometrica*, 41:617–631, 1973.

[17] T. Haynes, S. Sen, N. Arora, and R. Nadella. An automated meeting scheduling system that utilizes user preferences. In *Proceedings of the First Annual Conference on Autonomous Agents*, pages 308–315, Marina del Rey, California, 1997.

[18] E. Hemaspaandra, L. A. Hemaspaandra, and J. Rothe. Anyone but him: The complexity of precluding an alternative. *Artificial Intelligence*, 171(5–6):255–285, 2007.

[19] D. S. Hochbaum. *Approximation Algorithms for NP-Hard Problems*. PWS Publishing Company, 1997.

[20] H. Moulin. On strategy-proofness and single peakedness. *Public Choice*, 35:437–455, 1980.

[21] K. Oflazer and G. Tür. Morphological disambiguation by voting constraints. In *Proceedings of the 8th Conference of the European Chapter of the Association for Computational Linguistics (EACL 1997)*, pages 222–229, 1997.

[22] D. Pennock, E. Horvitz, and L. Giles. Social choice theory and recommender systems: Analysis of the axiomatic foundations of collaborative filtering. In *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI 2000)*, pages 729–734, 2000.

[23] A. D. Procaccia and J. S. Rosenschein. Average-case tractability of manipulation in elections via the fraction of manipulators. In *The Sixth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2007)*, pages 718–720, Honolulu, Hawaii, May 2007.

[24] A. D. Procaccia and J. S. Rosenschein. Junta distributions and the average-case complexity of manipulating elections. *Journal of Artificial Intelligence Research*, 28:157–181, 2007.

[25] A. D. Procaccia, J. S. Rosenschein, and A. Zohar. Multi-winner elections: Complexity of manipulation, control and winner-determination. In *The Twentieth International Joint Conference on Artificial Intelligence (IJCAI 2007)*, pages 1476–1481, Hyderabad, India, January 2007.

[26] M. Satterthwaite. Strategy-proofness and Arrow's conditions: Existence and correspondence theorems for voting procedures and social welfare functions. *Journal of Economic Theory*, 10:187–217, 1975.

[27] G. Sigletos, G. Paliouras, C. Spyropoulos, and M. Hatzopoulos. Combining information extractions systems using voting and stacked generalization. *Journal of Machine Learning Research*, 6:1751–1782, 2006.

[28] A. Slinko. How large should a coalition be to manipulate an election? *Mathematical Social Sciences*, 47(3):289–293, 2004.

[29] W. Vickrey. Counter speculation, auctions, and competitive sealed tenders. *Journal of Finance*, 16(1):8–37, 1961.

A Copeland is Not Monotone in Weights

When discussing Scoring rules, Maximin, and Plurality with Runoff, we are motivated to look for approximate solutions to the CCWM problem by the fact that these voting rules are monotone in weights. In contrast, Copeland is not monotone in weights. The next example illustrates this fact. Consider the following setting: $C = \{p, 1, 2, 3\}$, $N = |S| = 6$. All the weights equal 1. The votes of the voters in S are shown in the following table:

Voter in S	Vote
1	$p \succ 1 \succ 2 \succ 3$
2	$p \succ 2 \succ 1 \succ 3$
3	$3 \succ p \succ 1 \succ 2$
4	$3 \succ p \succ 2 \succ 1$
5	$1 \succ 2 \succ 3 \succ p$
6	$2 \succ 1 \succ 3 \succ p$

The pairwise results are given in the next table. In the cell corresponding to the row of candidate g and the column of candidate g', we write "$a : b$" to indicate that g is preferred to g' by a voters, and g' is preferred to g by b voters (i.e., $a = N_0(g, g'),\ b = N_0(g', g)$):

	p	1	2	3
p		4:2	4:2	2:4
1	2:4		3:3	4:2
2	2:4	3:3		4:2
3	4:2	2:4	2:4	

From the above table we calculate that $\sigma_0(p) = 1,\ \sigma_0(1) = \sigma_0(2) = 0,\ \sigma_0(3) = -1$, so p wins the election in this setting. However, if we add another voter (with weight 1), then no matter what his vote would be, p would not win the election: if the additional voter puts 1 above 2, then 1 will win, and otherwise 2 will win.

Remark A.1. It is easy to see, however, that whenever there is a manipulation for the coalition with weights W, then there is also a manipulation for coalition with weights $W + \{w\} + \{w\}$, where $w \geq 1$ is an integer: the first additional voter makes an arbitrary vote, and the second additional voter reverses the first's ranking.

B Alternative Algorithm for CCWM in Maximin

Consider the following simple algorithm, which we refer to as Algorithm 4. Given the list of weights W', let $W = \{w_1, \ldots, w_k\}$ be the maximal (with respect to set inclusion) list of weighted votes such that W' contains two copies of W, i.e., $W_1 + W_2 \subseteq W'$, $W_1 = W_2 = W$.[4] Each manipulator in W_1 votes $p \succ c_1 \succ \ldots \succ c_{m-1}$, while every manipulator in W_2 votes $p \succ c_{m-1} \succ \ldots \succ c_1$. The remaining manipulators all rank p first, and the other candidates arbitrarily. The algorithm returns *true* iff this ballot makes p win.

We will now easily show that Theorem 3.16 also applies to Algorithm 4.

Theorem B.1. *In the Maximin rule, let C be the set of candidates with $p \in C$ the preferred candidate, and S the set of voters who already cast their votes. Let W be the weight list for the set T. Then:*

1. *If there is no ballot making p win the election, then Algorithm 4 will return* false.
2. *If there is a ballot making p win the election, then for the same instance with weight list W' s.t. $W' \supseteq W + W$ (i.e., W' contains two copies of W), Algorithm 4 will return* true.

[4] To simplify notation we overload W and identify it with the list of manipulators. It is straightforward that this set can be efficiently found. Indeed, for each weight in the list W', simply check if there is another copy, and if so, place one of them in W_1 and one in W_2.

Proof. Item 1, as always, is obvious since the algorithm is constructive. For Item 2, let $\sigma^*(c)$ be candidate c's Maximin score based on the votes in S and the manipulator weights W which make p win. Let $\sigma'(c)$ be c's Maximin score based on the votes in S and the votes in $W + W$, according to the algorithm (notice that $W \subseteq W_1$, $W \subseteq W_2$). Finally, let $\sigma(c)$ be c's score according to the algorithm, on the weight list W'. As before, denote $W = \{w_1, \ldots, w_k\}$.

First, note that $\sigma'(p) \geq \sigma^*(p) + \sum_{i=1}^{k} w_k$. Moreover, clearly for any $c \in C \setminus \{p\}$, $\sigma'(c) \leq \sigma^*(c) + \sum_{i=1}^{k} w_k$, as for each $c' \in C \setminus \{c, p\}$ and each w_i in the multiset W there is exactly one manipulator in $W_1 + W_2$ with weight w_i which ranks c above c'. Since $\sigma^*(p) > \sigma^*(c)$ for any $c \in C \setminus \{p\}$, we conclude that

$$\forall c \in C \setminus \{p\},\ \sigma'(p) > \sigma'(c). \tag{28}$$

In order to complete the proof, we note that $\sigma(p) - \sigma'(p) \geq \sigma(c) - \sigma'(c)$ for any $c \in C \setminus \{p\}$, as all the manipulators with weights $W' \setminus (W_1 + W_2)$ rank p first. Together with the above, we get that $\sigma(p) > \sigma(c)$ for all $c \in C \setminus \{p\}$. □

C Proof of Theorem 4.6

We prove this theorem via the Lemmata C.1–C.4. The proof technique is similar to that of Theorem 3.4, but the proof is easier.

First, we define the set $X_n = \{x \in C \setminus \{p\} \mid x \text{ was ranked last in stage } j \text{ for } 1 \leq j \leq n\}$. In addition, define $Y_n = \{y \in C \setminus \{p\} \mid \sigma_n(y) \geq \min(\sigma_n(X_n))\}$. From the definition, $X_n \subseteq Y_n$. Also, by definition:

$$\forall g \notin Y_n \cup \{p\},\ \sigma_n(g) < \min(\sigma_n(Y_n)) \tag{29}$$

We denote by $\alpha_j(x)$ the number of points x was awarded in stage j.

Lemma C.1. *For all $y_1, y_2 \in Y_n$, $|\sigma_n(y_1) - \sigma_n(y_2)| \leq 1$.*

Proof. Let $x^* \in X_n$ s.t. $\sigma_n(x^*) = \min(\sigma_n(X_n))$. Let $y \in Y_n$. By definition, $\sigma_n(x^*) \leq \sigma_n(y)$. We would like to show that $\sigma_n(y) \leq \sigma_n(x^*) + 1$. Suppose for contradiction that $\sigma_n(y) - \sigma_n(x^*) \geq 2$. Let $1 \leq j \leq n$ maximal s.t. $\alpha_j(x^*) = 0$. Then:

$$\begin{aligned}
\sigma_j(y) - \sigma_j(x^*) &= \Big[\sigma_n(y) - \sum_{k=j+1}^{n} \alpha_k(y)\Big] - \Big[\sigma_n(x^*) - \sum_{k=j+1}^{n} \alpha_k(x^*)\Big] \\
&\geq \big[\sigma_n(y) - (n-j)\big] - \big[\sigma_n(x^*) - (n-j)\big] \\
&= \sigma_n(y) - \sigma_n(x^*) \geq 2
\end{aligned}$$

Therefore $\sigma_{j-1}(y) - \sigma_{j-1}(x^*) \geq 1$, and so $\sigma_{j-1}(y) > \sigma_{j-1}(x^*)$, a contradiction to $\alpha_j(x^*) = 0$. We showed that for all $y \in Y_n$, $\sigma_n(x^*) \leq \sigma_n(y) \leq \sigma_n(x^*) + 1$, and hence for all $y_1, y_2 \in Y_n$, $|\sigma_n(y_1) - \sigma_n(y_2)| \leq 1$. □

Lemma C.2. *Define $q(n) := \frac{1}{|Y_n|} \sum_{y \in Y_n} \sigma_n(y)$. Let Z_T be a preference list of voters in T, and $\sigma'_n(g)$ be the scores of $g \in C$ which are implied by Z_T (including votes in S). Then $q'(n) := \frac{1}{|Y_n|} \sum_{y \in Y_n} \sigma'_n(y) \geq q(n)$.*

Proof. The above fact is true since in the algorithm, at every stage j, there is some $x \in X_n \subseteq Y_n$ such that $\alpha_j(x) = 0$, and so for every j, the sum $\sum_{y \in Y_n} \alpha_j(y) = |Y_n| - 1$ is minimal. Formally, let

us denote by $\alpha'_j(g)$ the number of points candidate g gets from voter j in Z_T. Then:

$$\begin{aligned} q(n) &= \frac{1}{|Y_n|}\Big(\sum_{y\in Y_n}\sigma_0(y) + n(|Y_n|-1)\Big) \\ &\le \frac{1}{|Y_n|}\Big(\sum_{y\in Y_n}\sigma_0(y) + \sum_{j=1}^{n}\sum_{y\in Y_n}\alpha'_j(y)\Big) \\ &= \frac{1}{|Y_n|}\sum_{y\in Y_n}\sigma'_n(y) = q'(n) \end{aligned}$$

□

Lemma C.3. *If $\sigma_n(p) > \max(\sigma_n(Y_n))$ then Algorithm 2 will find the manipulation that makes p win.*

Proof. By Equation (29), for all $g \in C \setminus \{p\}$, $\sigma_n(g) \le \max(\sigma_n(Y_n))$, and so if $\sigma_n(p) > \max(\sigma_n(Y_n))$, then for all $g \in C \setminus \{p\}$, $\sigma_n(p) > \sigma_n(g)$, and so the algorithm will find the manipulation. □

Lemma C.4. *If $\sigma_n(p) \le \max(\sigma_n(Y_n))$ then there exists no manipulation.*

Proof. Let Z_T be a set of preferences of voters in T, and let $\sigma'_n(g)$, $q'(n)$ and $\alpha'_j(g)$ be as in Lemma C.2. As for all j, $\alpha_j(p) = 1 \ge \alpha'_j(p)$, it follows that $\sigma_n(p) \ge \sigma'_n(p)$. There is at least one $g_0 \in Y_n$ s.t. $\sigma'_n(g_0) \ge \lceil q'(n) \rceil$. By Lemma C.2, $\lceil q'(n) \rceil \ge \lceil q(n) \rceil$. By Lemma C.1, $\lceil q(n) \rceil = \max(\sigma_n(Y_n))$. Combining the foregoing steps, we obtain:

$$\sigma'_n(g_0) \ge \lceil q'(n) \rceil \ge \lceil q(n) \rceil = \max(\sigma_n(Y_n)) \ge \sigma_n(p) \ge \sigma'_n(p)$$

We conclude that p does not win under Z_T, and hence there is no ballot of votes in T that makes p win the election. □

The proof of the Theorem 4.6 is completed.