Ties Matter: Complexity of Voting Manipulation Revisited

Svetlana Obraztsova
School of Physical and Mathematical Sciences
Nanyang Technological University, Singapore
SVET0001@ntu.edu.sg

Edith Elkind
School of Physical and Mathematical Sciences
Nanyang Technological University, Singapore
eelkind@ntu.edu.sg

Noam Hazon
Robotics Institute
Carnegie Mellon University, USA
noamh@cs.cmu.edu

ABSTRACT

In their groundbreaking paper, Bartholdi, Tovey and Trick [1] argued that many well-known voting rules, such as Plurality, Borda, Copeland and Maximin are easy to manipulate. An important assumption made in that paper is that the manipulator's goal is to ensure that his preferred candidate is among the candidates with the maximum score, or, equivalently, that ties are broken in favor of the manipulator's preferred candidate. In this paper, we examine the role of this assumption in the easiness results of [1]. We observe that the algorithm presented in [1] extends to all rules that break ties according to a fixed ordering over the candidates. We then show that all scoring rules are easy to manipulate if the winner is selected from all tied candidates uniformly at random. This result extends to Maximin under an additional assumption on the manipulator's utility function that is inspired by the original model of [1]. In contrast, we show that manipulation becomes hard when arbitrary polynomial-time tie-breaking rules are allowed, both for the rules considered in [1], and for a large class of scoring rules.

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Multiagent Systems; F.2 [**Theory of Computation**]: Analysis of Algorithms and Problem Complexity

General Terms

Algorithms, Theory

Keywords

voting, manipulation, tie-breaking rules, complexity

1. INTRODUCTION

Computational social choice is an actively growing subarea of multiagent systems that provides theoretical foundations for preference aggregation and collective decision-making in multiagent domains. One of the most influential early contributions to this area is the paper by Bartholdi, Tovey, and Trick entitled "The computational difficulty of manipulating an election" [1]. In this paper, the authors suggested that computational complexity can serve as a barrier to dishonest behavior by the voters, and proposed classifying voting rules according to how difficult it is to manipulate them. In particular, they argued that such well-known voting rules as Plurality, Borda, Copeland and Maximin are easy to manipulate, yet a variant of the Copeland rule known as second-order Copeland is computationally resistant to manipulation. In a subsequent paper, Bartholdi and Orlin [2] showed that another well-known voting rule, namely, STV, is NP-hard to manipulate as well.

Since then, the computational complexity of manipulation under various voting rules, either by a single voter or by a coalition of voters, received considerable attention in the literature, both from the theoretical and from the experimental perspective (see, in particular, [20, 19] and the recent survey [9] for the former, and [17, 5] for the latter). While it has been argued that worst-case complexity does not provide adequate protection against malicious behavior (see, e.g. [15, 18, 10, 13]), determining whether a given voting rule is NP-hard to manipulate is still a natural first step in evaluating its resistance to manipulation in realistic scenarios.

An important property of the voting rules discussed in [1] is that they may produce multiple winners, i.e., they are, in fact, voting correspondences (see Section 2 for the formal definitions). It is not immediately clear what it means for manipulation to be successful in such a case. Bartholdi, Tovey and Trick take a rather liberal approach in their paper: they define a manipulation to be successful if, as a result, the manipulator's preferred candidate is *one of the election winners*. This approach is equivalent to assuming that ties are broken in favor of the manipulator. Now, a careful examination of the algorithm in [1] shows that it works as long as ties are broken either adversarially to the manipulator or according to an arbitrary fixed lexicographic order over the candidates. However, in real-life settings, when an election ends in a tie, it is not uncommon to choose the winner using a tie-breaking rule that is non-lexicographic in nature. Indeed, perhaps the most common approach is to toss a coin, i.e., select the winner uniformly at random among all tied alternatives. A more sophisticated example is provided by the second-order Copeland rule studied in [1], which is effectively the Copeland rule combined with a rather involved tie-breaking method. Despite its apparent complexity, the second-order Copeland is the voting rule of choice for several organizations [1]. Thus, it is natural to ask under what conditions on the tie-breaking rule the voting correspondences considered in [1] remain easy to manipulate.

In this paper, we make two contributions towards answering this question. We first consider the randomized tie-breaking rule, which chooses the winner uniformly at random among all tied candidates. Now, to formalize the notion of a successful manipulation under this rule, we need additional information about the manipulator's preferences: knowing the manipulator's preference order is insufficient for determining whether he prefers a tie between his top candidate and his least favorite candidate to his second choice becom-

Cite as: Ties Matter: Complexity of Voting Manipulation Revisited, S. Obraztsova, E. Elkind, N. Hazon, *Proc. of 10th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2011)*, Tumer, Yolum, Sonenberg and Stone (eds.), May, 2–6, 2011, Taipei, Taiwan, pp. XXX-XXX.

ing the unique winner. Thus, following [6], we endow the manipulator with utilities for all candidates, and seek a manipulation that maximizes his expected utility, where the expectation is taken over the random bits used to select the winner. We demonstrate that for all scoring rules such a manipulation can be found in polynomial time. This is also true for Maximin as long as the manipulator's utility function has a special form that is inspired by the notion of manipulation employed in [1]: namely, the manipulator values one of the candidates at 1 and the rest of the candidates at 0.

Given these easiness results, it is natural to ask whether all (efficiently computable) tie-breaking rules produce easily manipulable rules when combined with the voting correspondences considered in [1]. Now, paper [1] shows that for Copeland this is not the case, by proving that the second-order Copeland rule is hard to manipulate. However, prior to our work, no such result was known for other rules considered in [1]. Our second contribution is in demonstrating that Maximin and Borda, as well as many families of scoring rules, become hard to manipulate if we allow arbitrary polynomial-time tie-breaking rules. This holds even if we require that the tie-breaking rule only depends on the set of the tied alternatives, rather than the voters' preferences over them; we will refer to such tie-breaking rules as *simple*. Our proof also works for Copeland, thus strengthening the hardness result of [1] to simple tie-breaking rules. One can view these results as a continuation of the line of work suggested in [3, 7], namely, identifying minor tweaks to voting rules that make them hard to manipulate. Indeed, here we propose to "tweak" a voting rule by combining it with an appropriate tie-breaking rule; arguably, such a tweak affects the original rule less than the modifications proposed in [3] and [7] (i.e., combining a voting rule with a preround or taking a "hybrid" of the rule with itself or another rule). We remark, however, that our hardness result is not universal: Plurality and other scoring rules that correspond to scoring vectors with a bounded number of non-zero coordinates are easy to manipulate under any polynomial-time simple tie-breaking rule, However, if non-simple tie-breaking rules are allowed, Plurality can be shown to be hard to manipulate as well.

The rest of the paper is organized as follows. We cover the preliminaries and introduce the necessary notation in Section 2. Section 3 discusses the algorithm and the formal model of [1]. We describe the algorithms for scoring rules and Maximin under randomized tie-breaking in Section 4, and prove our hardness results in Section 5. Section 6 concludes.

2. PRELIMINARIES

An *election* is specified by a set of candidates C, $|C| = m$, and a set of *voters* $V = \{v_1, \dots, v_n\}$, where each voter v_i is associated with a linear order R_i over the candidates in C; this order is called v_i's *preference order*. We denote the space of all linear orderings over C by $\mathcal{L}(C)$. For readability, we will sometimes denote R_i by $\succ_i$. When $a \succ_i b$ for some $a, b \in C$, we say that voter v_i prefers a to b. The vector $\mathcal{R} = (R_1, \dots, R_n)$, where each R_i is a linear order over C, is called a *preference profile*. A *voting rule* $\mathcal{F}$ is a mapping that, given a preference profile $\mathcal{R}$ over C outputs a candidate $c \in C$; we write $c = \mathcal{F}(\mathcal{R})$. Many classic voting rules, such as the ones defined below, are, in fact, *voting correspondences*, i.e., they map a preference profile $\mathcal{R}$ to a non-empty subset S of C. Voting correspondences can be transformed into voting rules using tie-breaking rules. A *tie-breaking rule* for an election (C, V) is a mapping $T = T(\mathcal{R}, S)$ that for any $S \subseteq C$, $S \neq \emptyset$, outputs a candidate $c \in S$. A tie-breaking rule T is called *simple* if it does not depend on $\mathcal{R}$, i.e., the value of $T(\mathcal{R}, S)$ is uniquely determined by S. Such rules have the attractive property that if a manipulator cannot change the set of tied candidates, he cannot affect the outcome of the election. Further, we say that T is *lexicographic* with respect to a preference ordering $\succ$ over C if for any preference profile $\mathcal{R}$ over C and any $S \subseteq C$ it selects the most preferred candidate from S with respect to $\succ$, i.e., we have $T(S) = c$ if and only if $c \succ a$ for all $a \in S \setminus \{c\}$.

A *composition* of a voting correspondence $\mathcal{F}$ and a tie-breaking rule T is a voting rule $T \circ \mathcal{F}$ that, given a preference profile $\mathcal{R}$ over C, outputs $T(\mathcal{R}, \mathcal{F}(\mathcal{R}))$. Clearly, $T \circ \mathcal{F}$ is a voting rule and $T \circ \mathcal{F}(\mathcal{R}) \in \mathcal{F}(\mathcal{R})$.

We will now describe the voting rules (correspondences) considered in this paper. All these rules assign scores to candidates; the winners are the candidates with the highest scores.

Scoring rules Any vector $\alpha = (\alpha_1, \dots, \alpha_m) \in \mathbb{R}^m$ with $\alpha_1 \geq \cdots \geq \alpha_m$ defines a *scoring rule* $\mathcal{F}_\alpha$. Under this rule, each voter grants α_i points to the candidate it ranks in the i-th position; the score of a candidate is the sum of the scores it receives from all voters. The vector α is called a *scoring vector*. A scoring rule is said to be *faithful* if $\alpha_1 > \cdots > \alpha_m$. We are interested in scoring rules that are succinctly representable; therefore, throughout this paper we assume that the coordinates of α are nonnegative integers given in binary. We remark that scoring rules are defined for a fixed number of candidates. Therefore, we will often consider families of scoring rules, i.e., collections of the form $(\alpha^m)_{m=1}^\infty$, where $\alpha^m = (\alpha_1^m, \dots, \alpha_m^m)$. We require such families to be polynomial-time computable, i.e., we only consider families of voting rules $(\alpha^m)_{m=1}^\infty$ for which there exists a polynomial-time algorithm that given an $m \in \mathbb{N}$ outputs $\alpha_1^m, \dots, \alpha_m^m$. Two well-known examples of polynomial-time computable families of scoring rules are *Borda*, given by $\alpha^m = (m-1, \dots, 1, 0)$, and *$k$-approval*, given by $\alpha_i^m = 1$ if $i \leq k$, $\alpha_i^m = 0$ if $i > k$. 1-approval is also known as *Plurality*.

Copeland We say that a candidate a wins a *pairwise election* against b if more than half of the voters prefer a to b; if exactly half of the voters prefer a to b, then a is said to *tie* his pairwise election against b. Given a rational value $\alpha \in [0, 1]$, under the Copeland$^\alpha$ rule each candidate gets 1 point for each pairwise election he wins and α points for each pairwise election he ties.

Maximin The Maximin score of a candidate $c \in C$ is equal to the number of votes he gets in his worst pairwise election, i.e., $\min_{d \in C \setminus \{c\}} |\{i \mid c \succ_i d\}|$.

Given a preference profile $\mathcal{R}$ over a set of candidates C, for any preference order L over C we denote by $(\mathcal{R}_{-i}, L)$ the preference profile obtained from $\mathcal{R}$ by replacing R_i with L. We say that a voter v_i can successfully *manipulate* an election (C, V) with a preference profile $(R_1, \dots, R_n)$ with respect to a voting rule $\mathcal{F}$ if $\mathcal{F}(\mathcal{R}_{-i}, L) \succ_i \mathcal{F}(\mathcal{R})$. We will now define the computational problem that corresponds to this notion.

An instance of the $\mathcal{F}$-MANIPULATION problem is given by a set of candidates C, a set of voters V, a preference profile $\mathcal{R}$, and the manipulating voter v_i. It is a "yes"-instance if there exists a vote L such that $\mathcal{F}(\mathcal{R}_{-i}, L) \succ_i \mathcal{F}(\mathcal{R})$ and a "no"-instance otherwise.

3. THE MODEL AND THE ALGORITHM OF BARTHOLDI, TOVEY AND TRICK

Before we describe the algorithm presented in [1], we remark that the definition of successful manipulation given in [1] differs from our definition of $\mathcal{F}$-MANIPULATION (which is modeled after the standard social choice definition, see, e.g. [12, 16]), even if we assume that $\mathcal{F}$ is a voting rule rather than a voting correspondence. Specifically, in [1] it is assumed that the manipulator has a preferred candidate p, and his goal is to make p elected; we will refer

to this problem as $\mathcal{F}$-MANIPULATION(p). However, a polynomial-time algorithm for $\mathcal{F}$-MANIPULATION(p) can be converted into a polynomial-time algorithm for $\mathcal{F}$-MANIPULATION: we can simply run $\mathcal{F}$-MANIPULATION(p) on all candidates ranked by the manipulator above the current winner, and pick the best among the candidates for which $\mathcal{F}$-MANIPULATION(p) outputs "yes". Thus, if $\mathcal{F}$-MANIPULATION is hard, $\mathcal{F}$-MANIPULATION(p) is hard, too. Moreover, all of our hardness reductions directly show hardness of both variants of the problem.

The algorithm for $\mathcal{F}$-MANIPULATION(p) proposed in [1] assumes that the voting rule assigns scores to all candidates, and the winners are the candidates with the highest scores. Let v be the manipulator, and let p be her preferred candidate. The algorithm places p first, and then fills in the remaining positions in the vote from top to bottom, searching for a candidate that can be placed in the next available position in v's vote so that his score does not exceed that of p. This approach works as long as the rule is monotone and we can determine a candidate's final score given his position in v's vote and the identities of the candidates that v ranks above him. It is not hard to show that Plurality and Borda (and, in fact, all scoring rules), as well as Copeland and Maximin have this property.

We can easily modify this algorithm for the setting where the ties are broken adversarially to the manipulator: in that case, when the manipulator fills a position i in his vote, $i > 1$, he needs to ensure that the score of the candidate in that position is strictly less than that of p. Generally, if ties are broken according to a lexicographic ordering $\succ$ over the candidates, when placing a candidate c with $c \succ p$, the manipulator needs to make sure that c's score is less than that of p, and when placing a candidate c with $c \prec p$, he needs to make sure that c's score does not exceed that of p.

4. RANDOMIZED TIE-BREAKING RULES

In this section, we consider a very common approach to tie-breaking, namely, choosing the winner uniformly at random among all tied candidates. In this case, knowing the manipulator's preference ordering is not sufficient to determine his optimal strategy. For example, suppose that voter v prefers a to b to c, and by voting strategically he can change the output of the voting correspondence from b to $\{a, c\}$. It is not immediately clear if this manipulation is beneficial. Indeed, if v strongly prefers a, but is essentially indifferent between b and c, then the answer is probably positive, but if v strongly dislikes c and slightly prefers a to b, the answer is likely to be negative (of course, this also depends on v's risk attitude).

Thus, to model this situation appropriately, we need to know the utilities that the manipulator assigns to all candidates. Under the natural assumption of risk neutrality, the manipulator's utility for a set of candidates is equal to his expected utility when the candidate is drawn from this set uniformly at random, or, equivalently, to his *average* utility for a candidate in this set. Since we are interested in computational issues, it is reasonable to assume that all utilities are rational numbers; by scaling, we can assume that all utilities are positive integers given in binary.

Formally, given a set of candidates C, we assume that the manipulator is endowed with a utility function $u : C \to \mathbb{N}$. This function can be extended to sets of candidates by setting $u(S) = \frac{1}{|S|}\sum_{c\in S} u(c)$ for any $S \subseteq C$. Given a voting correspondence $\mathcal{F}$ and an election (C, V) with a preference profile $\mathcal{R}$, we say that a vote L is *optimal* for a voter $v_i \in V$ with a utility function $u_i : C \to \mathbb{N}$ with respect to $\mathcal{F}$ combined with the randomized tie-breaking rule if $u_i(\mathcal{F}(\mathcal{R}_{-i}, L)) \geq u_i(\mathcal{F}(\mathcal{R}_{-i}), L')$ for all $L' \in \mathcal{L}(C)$. We say that v_i has a *successful manipulation* if his optimal vote L satisfies $u_i(\mathcal{F}(\mathcal{R}_{-i}, L)) > u_i(\mathcal{F}(\mathcal{R}))$. In the rest of this section, we will explore the complexity of finding an optimal vote with respect to scoring rules and Maximin.

4.1 Scoring rules

All scoring rules turn out to be easy to manipulate under randomized tie-breaking.

THEOREM 4.1. *For any election $E = (C, V)$ with $|C| = m$, any voter $v \in V$ with a utility function $u : C \to \mathbb{N}$, and any scoring vector $\alpha = (\alpha_1, \ldots, \alpha_m)$, we can find in polynomial time an optimal vote for v with respect to the scoring rule $\mathcal{F}_\alpha$ combined with the randomized tie-breaking rule.*

PROOF. Fix a voter $v \in V$ with a utility function u, and let $\mathcal{R}'$ denote the preference profile consisting of all other voters' preferences. Let s_i denote the score of candidate c_i after all voters other than v have cast their vote. Let us renumber the candidates in order of increasing score, and, within each group with the same score, in order of decreasing utility. That is, under the new ordering we have $s_1 \leq \cdots \leq s_m$ and if $s_i = s_j$ for some $i < j$ then $u(c_i) \geq u(c_j)$. We say that two candidates c_i, c_j with $s_i = s_j$ belong to the same *level*. Thus, all candidates are partitioned into $h \leq m$ levels $H_1, \ldots, H_h$, so that if $c_i \in H_k$ and $c_j \in H_\ell$, $k < \ell$, then $s_i < s_j$.

Consider first the vote L_0 given by $c_1 \succ \ldots \succ c_m$, and let T be the number of points obtained by the winner(s) in $(\mathcal{R}', L_0)$. We claim that for any $L \in \mathcal{L}(C)$, in the preference profile $(\mathcal{R}', L)$ the winner(s) will get at least T points. Indeed, let c_i be the last candidate to get T points in $(\mathcal{R}', L_0)$, and suppose that there exists a vote L such that c_i gets less than T points in $(\mathcal{R}', L)$. By the pigeonhole principle, this means that L assigns at least α_i points to some c_j with $j > i$, and we have $s_j + \alpha_i \geq s_i + \alpha_i = T$, i.e., some other candidate gets at least T points, as claimed. We will say that a vote L is *conservative* if the winners' score in $(\mathcal{R}', L)$ is T.

We will now argue that if L maximizes v's utility, then either L is conservative or it can be chosen so that $\mathcal{F}_\alpha$ has a unique winner under $(\mathcal{R}', L)$. Indeed, suppose that this is not the case, i.e., any vote L that maximizes v's utility is such that the set $S = \mathcal{F}_\alpha(\mathcal{R}', L)$ is of size at least 2, and all candidates in S get $T' > T$ points. Let c_i be v's most preferred candidate in S; we have $u(c_i) \geq u(S)$. Suppose that L grants α_j points to c_i. Since we have $c_i + \alpha_j > T$, it follows that $j < i$. Now, consider the vote obtained from L_0 by swapping c_i and c_j. Clearly, all candidates in $C \setminus \{c_i, c_j\}$ get at most T points, and c_i gets $T' > T$ points. Further, c_j gets $s_j + \alpha_i \leq s_j + \alpha_j \leq T$ points. Thus, in this case c_i is a unique winner and $u(c_i) \geq u(S)$, a contradiction.

Therefore, to find an optimal manipulation, it suffices to (i) check for each candidate $c \in C$ whether c can be made the unique winner with a score that exceeds T and (ii) find an optimal conservative vote. The optimal manipulation can then be selected from the ones found in (i) and (ii).

Step (i) is easy to implement. Indeed, a candidate c_i can be made the unique winner with a score that exceeds T if and only if $s_i + \alpha_1 > T$. To see this, observe that if $s_i + \alpha_1 > T$, we can swap c_1 and c_i in L_0: c_i will get more than T points, and all other candidates will get at most T points. Conversely, if $s_i + \alpha_1 \leq T$, then the score of c_i is at most T no matter how v votes.

Thus, it remains to show how to implement (ii). Intuitively, our algorithm proceeds as follows. We start with the set of winners produced by L_0; we will later show that this set is minimal, in the sense that if it contains x candidates from some level, then for any vote the set of winners will contain at least x candidates from that level. Note also that due to the ordering of the candidates we select the best candidates from each level at this step. We then try

to increase the average utility of the winners' set. To this end, we order the remaining candidates by their utility, and try to add them to the set of winners one by one as long as this increases its average utility. We will now give a formal description of our algorithm and its proof of correctness.

Let $S_0 = \mathcal{F}_\alpha(\mathcal{R}', L_0)$. We initialize S and L by setting $S = S_0$, $L = L_0$. Let $\succ^*$ be some ordering of the set C that ranks the candidates in S_0 first, followed by the candidates in $C \setminus S_0$ in the order of decreasing utility, breaking ties arbitrarily. We order the candidates from $C \setminus S_0$ according to $\succ^*$, and process the candidates in this ordering one by one. For each candidate c_i, we check if $u(c_i) > u(S)$; if this is not the case, we terminate, as all subsequent candidates have even lower utility. Otherwise, we check if we can swap c_i with another candidate that is currently not in S and receives $T - s_i$ points from L (so that c_i gets T points in the resulting vote). If this is the case, we update L by performing the swap and set $S = S \cup \{c_i\}$. We then proceed to the next candidate on the list.

We claim that the vote L obtained in the end of this process is optimal for the manipulator, among all conservative votes. We remark that at any point in time S is exactly the set of candidates that get T points in $(\mathcal{R}', L)$. Thus, we claim that any conservative vote $\hat{L}$ satisfies $u(\mathcal{F}_\alpha(\mathcal{R}', \hat{L})) \leq u(S)$.

Assume that this is not the case. Among all optimal conservative votes, we will select one that is most "similar" to L in order to obtain a contradiction. Formally, let $\mathcal{L}_0$ be the set of all optimal conservative votes, and let $\mathcal{L}_1$ be the subset of $\mathcal{L}_0$ that consists of all votes L' that maximize the size of the set $\mathcal{F}_\alpha(\mathcal{R}', L') \cap S$. The ordering $\succ^*$ induces a lexicographic ordering on the subsets of C. Let $\hat{L}$ be the vote such that the set $\mathcal{F}_\alpha(\mathcal{R}', \hat{L})$ is minimal with respect to this ordering, over all votes in $\mathcal{L}_1$. Set $\hat{S} = \mathcal{F}_\alpha(\mathcal{R}', \hat{L})$; by our assumption we have $u(\hat{S}) > u(S)$.

Observe first that our algorithm never removes a candidate from S: when we want to add c_i to S and search for an appropriate swap, we only consider candidates that have not been added to S yet. Also, at each step of our algorithm the utility of the set S strictly increases. These observations will be important for the analysis of our algorithm.

We will first show that $\hat{S} \setminus S$ is empty.

LEMMA 4.2. *We have $\hat{S} \setminus S = \emptyset$.*

PROOF. Suppose that the lemma is not true, and let c_i be a candidate in $\hat{S} \setminus S$. Suppose that c_i appears in the j-th position in our ordering of $C \setminus S_0$. If our algorithm terminated at or before the j-th step, we have $u(c_i) < u(S) < u(\hat{S})$, and hence $u(\hat{S} \setminus \{c_i\}) > u(\hat{S})$, a contradiction with the optimality of $\hat{L}$.

Thus, when our algorithm considered c_i, it could not find a suitable swap. Since $c_i \in \hat{S}$, it has to be the case that there exists an entry of the scoring vector that equals $T - s_i$; however, when our algorithm processed c_i it was unable to place c_i in a position that grants $T - s_i$ points. This could only happen if all candidates that were receiving $T - s_i$ points from L at that point were in S at that time; denote the set of all such candidates by B_i. Note that all candidates in B_i belong to the same level as c_i. Also, all candidates in $B_i \cap S_0$ have the same or higher utility than c_i, because initially we order the candidates at the same level by their utility, so that L_0 grants a higher score to the best candidates at each level. On the other hand, all candidates in $B_i \setminus S_0$ were added to S at some point, which means that they have been processed before c_i. Since at this stage of the algorithm we order the candidates by their utility, it means that they, too, have the same or higher utility than c_i.

Now, since $\hat{L}$ grants $T - s_i$ points to c_i, it grants less than $T - s_i$ points to one of the candidates in B_i. Let c_k be any such candidate, and consider the vote $\hat{L}'$ obtained from $\hat{L}$ by swapping c_i and c_k. Let $\hat{S}' = \mathcal{F}_\alpha(\mathcal{R}', \hat{L}')$; we have $\hat{S}' = (\hat{S} \setminus \{c_i\}) \cup \{c_k\}$. By the argument above, we have either $u(c_k) > u(c_i)$ or $u(c_k) = u(c_i)$. In the former case, we get $u(\hat{S}') > u(\hat{S})$. In the latter case, we get $u(\hat{S}') = u(\hat{S})$ and $|\hat{S}' \cap S| > |\hat{S} \cap S|$. In both cases, we obtain a contradiction with our choice of $\hat{L}$. □

Thus, we have $\hat{S} \subseteq S$, and it remains to show that $S \subseteq \hat{S}$. We will first show that $\hat{S}$ contains all candidates in S_0.

LEMMA 4.3. *We have $S_0 \subseteq \hat{S}$.*

PROOF. Suppose that $|S_0 \cap H_k| = m_k$ for $k = 1, \ldots, h$. We will first show that $|\hat{S} \cap H_k| \geq m_k$ for $k = 1, \ldots, h$. Indeed, fix a $k \leq h$, and suppose that the first candidate in the k-th level is c_i. Then in $(\mathcal{R}', L_0)$ the scores of the candidates in H_k are $s_i + \alpha_i, \ldots, s_i + \alpha_j$ for some $j \geq i$. If $s_i + \alpha_i < T$, then $m_k = 0$ and our claim is trivially true for this value of k. Otherwise, by the pigeonhole principle, if it holds that in $(\mathcal{R}', \hat{L})$ less than m_k voters in H_k get T points, it has to be the case that at least one candidate in $H_{k+1} \cup \cdots \cup H_h$ receives at least α_i points from $\hat{L}$. However, for any $c_\ell \in H_{k+1} \cup \cdots \cup H_h$ we have $s_\ell > s_i$, so $s_\ell + \alpha_i > T$, a contradiction with our choice of $\hat{L}$.

Now, suppose that $S_0 \cap H_k \not\subseteq \hat{S} \cap H_k$ for some $k \leq h$, and consider a candidate $c_\ell \in (S_0 \cap H_k) \setminus (\hat{S} \cap H_k)$. Since we have argued that $|\hat{S} \cap H_k| \geq m_k$, it must be the case that there also exists a candidate $c_j \in (\hat{S} \cap H_k) \setminus (S_0 \cap H_k)$. It is easy to see that S_0 contains the m_k best candidates from H_k, so $u(c_\ell) \geq u(c_j)$. The rest of the proof is similar to that of Lemma 4.2: Consider the vote $\hat{L}'$ obtained from $\hat{L}$ by swapping c_ℓ and c_j and let $\hat{S}' = \mathcal{F}_\alpha(\mathcal{R}', \hat{L}')$. Since c_ℓ and c_j belong to the same level, we have $\hat{S}' = (\hat{S} \setminus \{c_\ell\}) \cup \{c_k\}$. Thus, either $u(\hat{S}') > u(\hat{S})$ or $u(\hat{S}') = u(\hat{S})$ and $|\hat{S}' \cap S| > |\hat{S} \cap S|$. In both cases we get a contradiction. Thus, we have $S_0 \cap H_k \subseteq \hat{S} \cap H_k$. Since this holds for every value of k, the proof is complete. □

Given Lemma 4.2 and Lemma 4.3, it is easy to complete the proof. Suppose that $\hat{S}$ is a strict subset of S. Observe first that for any subset S' of S there is a vote L' such that $\mathcal{F}_\alpha(\mathcal{R}', L') = S'$: we can simply ignore the candidates that are not members of S' when running our algorithm, as this only increases the number of "available" swaps at each step. Now, order the candidates in $C \setminus S_0$ according to $\succ^*$. Let c_i be the first candidate in this order that appears in S, but not in $\hat{S}$. If there is a candidate c_j that appears later in the sequence and is contained in both S and $\hat{S}$, consider the set $S' = \hat{S} \setminus \{c_j\} \cup \{c_i\}$. As argued above, there is a vote L' such that $\mathcal{F}_\alpha(\mathcal{R}', L') = S'$. Now, if $u(c_i) > u(c_j)$, this set has a higher average utility that $\hat{S}$. Thus, this is a contradiction with our choice of $\hat{L}$. On the other hand, if $u(c_j) = u(c_i)$, then we have $u(S') = u(\hat{S})$, $|S \cap S'| = |S \cap \hat{S}|$, and S' precedes $\hat{S}$ is the lexicographic ordering induced by $\succ^*$, a contradiction with the choice of $\hat{L}$ again. Therefore, none of the candidates in S that appear after c_i in the ordering belongs to $\hat{S}$. Now, when we added c_i to S, we did so because its utility was higher than the average utility of S at that point. However, by construction, the latter is exactly equal to $u(\hat{S})$. Thus, $u(\hat{S} \cup \{c_i\}) > u(\hat{S})$, a contradiction again. Therefore, the proof is complete. □

4.2 Maximin

For Maximin with randomized tie-breaking, we have not been able to design an efficient algorithm for finding an optimal manipulation in the general utility model. However, we will now present a

polynomial-time algorithm for this problem assuming that the manipulator's utility function has a special structure. Specifically, recall that in the model of [1] the manipulator's goal is to make a specific candidate p a winner. This suggests that the manipulator's utility can be modeled by setting $u(p) = 1$, $u(c) = 0$ for all $c \in C \setminus \{p\}$. We will now show that for such utilities there exists a poly-time algorithm for finding an optimal manipulation under Maximin combined with the randomized tie-breaking rule.

THEOREM 4.4. *If the manipulator's utility function is given by $u(p) = 1$, $u(c) = 0$ for $c \in C \setminus \{p\}$, the problem of finding an optimal manipulation under Maximin combined with the randomized tie-breaking rule is in* P.

PROOF. Consider an election $E = (C, V)$ with the candidate set $C = \{c_1, \ldots, c_m\}$ and the voter set $V = \{v_1, \ldots, v_n\}$, and let v_n be the manipulating voter. In this proof, we denote by $s(c_i)$ the Maximin score of a candidate $c_i \in C$ in the election $E' = (C, V')$, where $V' = \{v_1, \ldots, v_{n-1}\}$. Let $s = \max_{c_i \in C} s(c_i)$.

For any $c_i \in C$, the manipulator's vote increases the score of c_i either by 0 or by 1. Thus, if $s(p) < s - 1$, the utility of the manipulator will be 0 irrespective of how he votes.

Now, suppose that $s(p) = s - 1$. The manipulator can increase the score of p by 1 by ranking p first. Thus, his goal is to ensure that after he votes (a) no other candidate gets $s + 1$ point and (b) the number of candidates in $C \setminus \{p\}$ with s points is as small as possible. Similarly, if $s(p) = s$, the manipulator can ensure that p gets $s + 1$ points by ranking him first, so his goal is to rank the remaining candidates so that in $C \setminus \{p\}$ the number of candidates with $s + 1$ points is as small as possible. We will now describe an algorithm that works for both of these cases.

We construct a directed graph G with the vertex set C that captures the relationship among the candidates. Namely, we have an edge from c_i to c_j if there are $s(c_j)$ voters in V' that rank c_j above c_i. Observe that, by construction, each vertex in G has at least one incoming edge. We say that c_i is a *parent* of c_j in G whenever there is an edge from c_i to c_j. We remark that if the manipulator ranks one of the parents of c_j above c_j in his vote, then c_j's score does not increase. We say that a vertex c_i of G is *purple* if $s(c_i) = s(p) + 1$, *red* if $s(c_i) = s(p)$ and $c_i \neq p$, and *green* otherwise; note that by construction p is green. Observe also that if $s(p) = s$, there are no purple vertices in the graph. We will say that a candidate c_j is *dominated* in an ordering L (with respect to G) if at least one of c_j's parents in G appears before c_j in L. Thus, our goal is to ensure that the set of dominated candidates includes all purple candidates and as many red candidates as possible.

Our algorithm is based on a recursive procedure $\mathcal{A}$, which takes as its input a graph H with a vertex set $U \subseteq C$ together with a coloring of U into green, red and purple; intuitively, U is the set of currently unranked candidates. It returns "no" if the candidates in U cannot be ranked so that all purple candidates in U are dominated by other candidates in U with respect to H. Otherwise, it returns an ordered list L of the candidates in U in which all purple candidates are dominated, and a set S consisting of all red candidates in U that remain undominated in L with respect to H.

To initialize the algorithm, we call $\mathcal{A}(G)$. The procedure $\mathcal{A}(H)$ is described below.

1. Set $L = \emptyset$.
2. If H contains p, set $L = [p]$, and remove p from H.
3. While H contains a candidate c that is green or has a parent that has already been ranked, set $L :: [c]$ (where :: denotes the list concatenation operation) and remove c from H.
4. If H is empty, return $(L, \emptyset)$.
5. If there is a purple candidate in H with no parents in H, return "no".
6. If there is a red candidate c in H with no parents in H, let H' be the graph obtained from H by coloring c green. Compute $\mathcal{A}(H')$. If $\mathcal{A}(H')$ returns "no", return "no". Otherwise, if $\mathcal{A}(H')$ returns (L', S'), return $(L :: L', S' \cup \{c\})$.
7. At this point in the algorithm, each vertex of H has a parent. Hence, H contains a cycle. Let T be some such cycle. Collapse T, i.e., (a) replace T with a single vertex t, and (b) for each $y \notin T$, add an edge (t, y) if H contained an edge (x, y) for some $x \in T$ and add an edge (y, t) if H contained a vertex z with $(y, z) \in H$. Color t red if T contains at least one red vertex, and purple otherwise. Let H' be the resulting graph and call $\mathcal{A}(H')$. If $\mathcal{A}(H')$ returns "no", return "no". Now, suppose that $\mathcal{A}(H')$ returns (L', S').

   Suppose that $t \in S'$. At any point in the algorithm, we only put a vertex in S if it is red, so t must be red, and hence T contains a red vertex. Let c be some red vertex in T, and let $\hat{L}$ be an ordering of the vertices in T that starts with c and follows the edges of T. Let L'' be the list obtained from L' by replacing t with $\hat{L}$ (i.e., if $L' = L_1 :: [t] :: L_2$, then $L'' = L_1 :: \hat{L} :: L_2$). Return $(L :: L'', (S' \setminus \{t\}) \cup \{c\})$.

   If $t \notin S'$, then by Lemma 4.5 (see below) t is dominated in H'. Let a be a parent of t that precedes it in L'. Then T contains a child of a. Let c be some such child, and let $\hat{L}$ be an ordering of the vertices in T that starts with c and follows the edges of T. Let L'' be the list obtained from L' by replacing t wish $\hat{L}$. Return $(L :: L'', S')$.

We will now argue that our algorithm outputs "no" if and only if no matter how v_n votes, some candidate in $C \setminus \{p\}$ gets $s(p) + 2$ points. Moreover, if $\mathcal{A}(G) = (L, S)$ and the set S contains r red candidates, then whenever v_n votes so that after his vote all other candidates have at most $s(p) + 1$ points, there are at least r red candidates with $s(p) + 1$ points.

We will split the proof into several lemmas.

LEMMA 4.5. *At any point in the execution of the algorithm, if $\mathcal{A}(H) = (L, S)$, then each candidate in $U \setminus S$ is dominated in H.*

PROOF. The proof is by induction on the recursion depth. Consider a candidate $x \in U \setminus S$. Clearly, if there are no recursive calls, $\mathcal{A}$ ranks x at Step 3, and the claim is obviously true.

For the induction step, suppose that the claim is true if we have d nested recursive calls, and consider an execution that makes $d + 1$ nested calls. Again, consider a candidate $x \in U \setminus S$. As in the base case, if x has been ranked in Step 3 the claim is clearly true. If x was ranked in Step 6, it follows that $x \notin S'$, and the claim follows by the inductive assumption. Now, suppose that x was ranked in Step 7 when we collapsed some cycle T. If $x \notin T$, then $x \notin S'$ and the claim follows by the inductive assumption. In particular, if x was ranked after t before the expansion, there is some vertex y in T such that H contains the edge (y, x), so after expansion x will be dominated by y.

Now, suppose that $x \in T$. If t was in S', but x was not added to S, it means that x was not the first vertex of T to appear in the ranking, i.e., x was ranked after its predecessor in T. If t was not in S', then by the inductive assumption t was ranked after its parent in H', i.e., there is a $z \in H' \setminus \{t\}$ such that z is ranked before t in L' and there is an edge (z, t) in H'. By construction of t, this

means that there is a vertex $y \in T$ such that there is an edge (z, y) in H. Thus, when we expanded t into T, the first vertex of T to be ranked was placed after its parent, and all subsequent vertices of T were placed after their predecessors in T. Thus, all vertices in T and, in particular, x, are dominated. □

We are now ready to prove that our algorithm correctly determines whether the manipulator can ensure that no candidate gets more than $s(p) + 1$ points.

LEMMA 4.6. *The algorithm outputs "no" if and only if for any vote L there is a purple candidate that is undominated.*

PROOF. Observe that the algorithm only outputs "no" if it finds a purple candidate with no parents. Let c be some such candidate. Now, in the original graph G each vertex has a parent. Further, if there was an edge from some x to c, and we collapsed a cycle T that contains x, but not c, there is still an edge from the resulting vertex t to c. Thus, the only way to obtain a purple vertex with no incoming edges is by collapsing a cycle T such that T contains purple vertices only, and no vertex of T has an incoming edge. By induction on the execution of the algorithm, it is easy to see that if we obtained a purple vertex with no incoming edges at some point, then in the original graph there was a group of purple vertices such that there was no edge from any red or green vertex to any of the vertices in the group. Now, in any ordering on C one of the candidates in this group would have to be ranked first. By construction, this candidate would be ranked before all its parents, so it is undominated.

Conversely, suppose that the algorithm does not answer "no", and outputs a pair (L, S) instead. We have observed that S consists of red vertices only. Thus, by Lemma 4.5 each purple vertex is dominated. □

It remains to show that the set S output by the algorithm contains as few candidates as possible.

LEMMA 4.7. *At any point in the execution of the algorithm, if $\mathcal{A}(H) = (L, S)$, then in any ordering of the candidates in U in which each purple vertex in U is dominated, at least $|S|$ red vertices in U are undominated.*

PROOF. The proof is by induction on the recursion depth. Suppose first that we make no recursive calls. Then our algorithm outputs $S = \emptyset$, and our claim is trivially true. Now, suppose that our claim is true if we make d nested calls. Consider an execution of $\mathcal{A}$ which makes $d + 1$ nested call, and suppose that when we call $\mathcal{A}(H')$ within this execution, it returns (L', S').

Suppose first that we made the recursive call in Step 6 of the algorithm, and therefore set $S = S' \cup \{c\}$. Suppose for the sake of contradiction that there exists a ranking of the candidates in U such that at most $|S| - 1$ candidate is undominated. Since c has no parents in H, there are at most $|S| - 2$ other red candidates that are undominated. In other words, if we recolor c green, in the resulting instance (which is exactly the instance passed to $\mathcal{A}$ during the recursive call), there are at most $|S| - 2$ undominated red candidates. Since $|S'| = |S| - 1$, this is a contradiction with the inductive assumption.

Now, suppose that we made the recursive call in Step 7 of the algorithm, and collapsed a cycle T into a vertex t. Again, assume for the sake of contradiction that there exists a ranking $\bar{L}$ of the candidates in U such that at most $|S| - 1$ candidates are undominated. Let c be the first vertex of T to appear in $\bar{L}$. Consider the ranking of U' obtained by removing all vertices of $T \setminus \{c\}$ from $\bar{L}$ and replacing c with t; denote this ranking by $\bar{L}'$. We claim that in $\bar{L}'$ at most $|S| - 1$ vertices of H' are undominated. Indeed, any parent of c in H is a parent of t in H', so t is undominated if and only if c was. On the other hand, if for some vertex x the only parent that preceded it in $\bar{L}$ was a vertex $y \in T \setminus \{c\}$, then in H' there is an edge from t to x, i.e., x is preceded by its parent t in $\bar{L}'$. For all other vertices, if they were preceded by some parent z in $\bar{L}$, they are preceded by the same parent in $\bar{L}'$. Since $|S| = |S'|$, we have shown that U' can be ordered so that at most $|S'| - 1$ vertices are undominated, a contradiction with the inductive assumption. □

Combining Lemma 4.6 and Lemma 4.7, we conclude that if our algorithm outputs (L, S), then L is the optimal vote for v_n and if our algorithm outputs "no", then the manipulator's utility is 0 no matter how he votes. Also, it is not hard to see that the algorithm runs in polynomial time. Thus, the proof is complete. □

5. HARDNESS RESULTS

We will now demonstrate that if we allow arbitrary polynomial-time tie-breaking rules, the algorithmic results presented in the previous sections no longer hold. In fact the problem of finding a beneficial manipulation becomes NP-hard. We will first present a specific simple tie-breaking rule T. We will then show that manipulating the composition of this rule with Borda, Copeland or Maximin is NP-hard.

Recall that an instance $\mathcal{C}$ of 3-SAT is given by a set of s variables $X = \{x_1, \ldots, x_s\}$ and a collection of t clauses $Cl = \{c_1, \ldots, c_t\}$, where each clause $c_i \in Cl$ is a disjunction of three *literals* over X, i.e., variables or their negations; we denote the negation of x_i by $\overline{x}_i$. It is a "yes"-instance if there is a truth assignment for the variables in X such that all clauses in Cl are satisfied, and a "no"-instance otherwise. This problem is known to be hard even if we assume that all literals in each clause are distinct, so from now on we assume that this is the case. Now, given s variables $x_1, \ldots, x_s$, there are exactly $\ell = \binom{2s}{3}$ 3-literal clauses that can be formed from these variables (this includes clauses of the form $x_1 \wedge \overline{x}_1 \wedge x_2$). Ordering the literals as $x_1 < \overline{x}_1 < \cdots < x_s < \overline{x}_s$ induces a lexicographic ordering over all 3-literal clauses. Let ϕ_i denote the i-th clause in this ordering. Thus, we can encode an instance $\mathcal{C}$ of 3-SAT with s variables as a binary string $\sigma(\mathcal{C})$ of length ℓ, where the i-th bit of $\sigma(\mathcal{C})$ is 1 if and only if ϕ_i appears in $\mathcal{C}$.

We are ready to describe T. Given a set $S \subseteq C$ of candidates, where $|C| = m$, T first checks if $m = \ell + 2s + 4$ for some $s > 0$ and $\ell = \binom{2s}{3}$. If this is not the case, it outputs the lexicographically first candidate in S and stops. Otherwise, it checks whether $c_m \in S$ and for every $i = 1, \ldots, s$, the set S satisfies $|S \cap \{c_{\ell+2i-1}, c_{\ell+2i}\}| = 1$. If this is not the case, it outputs the lexicographically first candidate in S and stops. If the conditions above are satisfied, it constructs an instance $\mathcal{C} = (X, Cl)$ of 3-SAT by setting $X = \{x_1, \ldots, x_s\}$, $Cl = \{\phi_i \mid 1 \leq i \leq \ell, c_i \in S\}$. Next, it constructs a truth assignment $(\xi_1, \ldots, \xi_s)$ for $\mathcal{C}$ by setting $\xi_i = \top$ if $c_{\ell+2i-1} \in S$, $c_{\ell+2i} \notin S$ and $\xi_i = \bot$ if $c_{\ell+2i-1} \notin S$, $c_{\ell+2i} \in S$. Finally, if $\mathcal{C}(\xi_1, \ldots, \xi_s) = \top$, it outputs c_m and otherwise it outputs the lexicographically first candidate in S. Clearly, T is simple and polynomial-time computable, and hence the problem $T \circ \mathcal{F}$-MANIPULATION is in NP for any polynomial-time computable rule $\mathcal{F}$ (and, in particular, for Borda, Maximin and Copeland). In the rest of this section, we will show that $T \circ \mathcal{F}$-MANIPULATION is NP-hard for all these rules.

5.1 Borda and other scoring rules

We will first consider the Borda rule. We will then show that essentially the same proof works for a large class of scoring rules. To simplify notation, in the proof of Lemma 5.1 and Theorem 5.2

we will denote the Borda score of a candidate x in a preference profile $\mathcal{R}$ by $s(\mathcal{R}, x)$.

LEMMA 5.1. *For any set of candidates $C = \{c_1, \ldots, c_m\}$ with $m \geq 4$ and any vector $(\beta_1, \ldots, \beta_{m-1})$ with $\beta_i \in \{0, 1, \ldots, m\}$ for $i = 1, \ldots, m-1$ and $\beta_1 > 0$, we can efficiently construct a preference profile $\mathcal{R} = (R_1, \ldots, R_n)$ with $n = m(m-1)$ voters such that for some $K \geq m^2 + m + 1$ and some $u \leq m(m-1)$ the Borda scores of all candidates satisfy $s(\mathcal{R}, c_i) = K + \beta_i$ for $i = 1, \ldots, m-1$ and $s(\mathcal{R}, c_m) = u$.*

The proof of Lemma 5.1 is similar to that of Theorem 3.1 in [5] and is omitted due to space constraints.

THEOREM 5.2. *$T \circ$ Borda-MANIPULATION is* NP*-hard.*

PROOF. Suppose that we are given an instance $\mathcal{C}$ of 3-SAT with s variables. Note that this instance can be encoded by a binary vector $(\sigma_1, \ldots, \sigma_\ell)$, where $\ell = \binom{2s}{3}$, as described in the construction of T: $\sigma_i = 1$ if and only if $\mathcal{C}$ contains the i-th 3-variable clause with respect to the lexicographic order. We will now construct an instance of our problem with $m = \ell + 2s + 4$ candidates $c_1, c_2, \ldots, c_m$. For readability, we will also denote the first ℓ candidates by $u_1, \ldots, u_\ell$, the next $2s$ candidates by $x_1, y_1, \ldots, x_s, y_s$, and the last four candidates by d_1, d_2, w, and c.

Let $U = \{u_1, \ldots, u_\ell\}$, let $Q = \{c_i \in U \mid \sigma_i = 1\}$, and let $q = |Q|$. For convenience, we renumber the candidates in U so that $Q = \{u_1, \ldots, u_q\}$.

We will now use Lemma 5.1 to construct a preference profile $\mathcal{R} = (R_1, \ldots, R_n)$ with the following scores:

- $s(\mathcal{R}, w) = K + m$, $s(\mathcal{R}, c) = K + 1$;
- $s(\mathcal{R}, u_i) = K + m - i$ for $i = 1, \ldots, q$;
- $s(\mathcal{R}, u_i) = K$ for $i = q + 1, \ldots, \ell$;
- $s(\mathcal{R}, x_i) = s(\mathcal{R}, y_i) = K + i + 1$ for $i = 1, \ldots, s$;
- $s(\mathcal{R}, d_1) = K$, $s(\mathcal{R}, d_2) = u$,

where $K > m^2 + m + 1$ and $u \leq m(m-1)$.

Now, consider an election with the set of candidates C and a set of voters $V = \{v_1, \ldots, v_{n+1}\}$, where for $i \leq n$ the preferences of the i-th voter are given by R_i, and the preferences of the last voter (who is also the manipulating voter) are given by

$$c \succ w \succ x_1 \succ y_1 \succ \ldots \succ x_s \succ y_s \succ u_1 \succ \ldots \succ u_\ell \succ d_1 \succ d_2.$$

Observe that if v_{n+1} votes truthfully, w wins. Thus, a manipulation is successful if and only if v_{n+1} manages to vote so that c gets elected.

Suppose first that we have started with a "yes"-instance of 3-SAT, and let $(\xi_1, \ldots, \xi_s) \in \{\top, \bot\}^s$ be the corresponding truth assignment. For $i = 1, \ldots, s$, set $z_i = x_i$ if $\xi_i = \top$ and $z_i = y_i$ if $\xi_i = \bot$. Suppose that v_{n+1} submits a vote L in which he ranks $c, z_1, \ldots, z_s$ in the top $s+1$ positions (in this order), $u_q, \ldots, u_1, w$ in the bottom $q+1$ positions (in this order), and all other candidates in the remaining positions in between.

It is not hard to see that in this case the candidates c, w, $z_1, \ldots, z_s$ and all candidates in Q get $K + m$ points, while all other candidates get less than $K + m$ points. Thus, the set of tied candidates S is $Q \cup \{c, w, z_1, \ldots, z_s\}$. Therefore, given the set S, our tie-breaking rule will reconstruct $\mathcal{C}$, check whether $z_1, \ldots, z_s$ encode a satisfying truth assignment for $\mathcal{C}$ (which is indeed the case), and output $c_m = c$. Thus, in this case L is a successful manipulation.

Conversely, suppose that v_{n+1} submits a vote L so that c gets elected. Since we have $s(\mathcal{R}, w) - s(\mathcal{R}, c) = m - 1$, it follows that L ranks c first and w last, and hence both of them get $K + m$ points. Similarly, we can show by induction on i that for all $i = 1, \ldots, q$ it holds that v_{n+1} ranks u_i in the $(m-i)$-th position; thus, each candidate in Q also gets $K + m$ points. Moreover, all other candidates in $U \setminus Q$ get less than $K + m$ points, i.e, the manipulator cannot change the formula encoded by the set of tied candidates. Let S be the set of all candidates with the top score. Since c wins the election, it has to be the case that the set $S \cap \{x_1, y_1, \ldots, x_s, y_s\}$ encodes a satisfying truth assignment for $\mathcal{C}$, i.e., $\mathcal{C}$ is satisfiable. Thus, the proof is complete. □

Theorem 5.2 can be generalized to other families of scoring rules, including k-approval, where k is polynomially related to m (i.e., $m = \text{poly}(k)$). Note, however, that we cannot hope to prove an analogue of Theorem 5.2 for *all* scoring rules as long as we insist that the tie-breaking rule is simple: we have to require that the scoring vector has a superlogarithmic number of non-zero coordinates. Indeed, if the number of non-zero coordinates k satisfies $k = O(\log m)$, the manipulator can simply try all possible placements of the candidates into the top k positions in polynomial time. This strategy works for any simple polynomial-time tie-breaking rule, since the set of tied candidates only depends on the top k positions in the manipulator's vote. On the other hand, if we drop the simplicity requirement, there are tie-breaking rules for which even Plurality is hard to manipulate.

THEOREM 5.3. *There exists a tie-breaking rule T' such that $T' \circ$ Plurality-MANIPULATION is* NP*-complete.*

We omit the formal proof of Theorem 5.3 due to space constraints; intuitively, we can consider a tie-breaking rule T' that interprets the set of winners as a boolean formula and views the manipulator's vote as a truth assignment.

5.2 Copeland and Maximin

We will now show that $T \circ$ Copeland and $T \circ$ Maximin are hard to manipulate using essentially the same construction as in the proof of the Theorem 5.2.

THEOREM 5.4. *$T \circ$ Maximin-MANIPULATION is* NP*-hard.*

PROOF. Given a 3-SAT formula $\mathcal{C}$, we construct an election $E = (C, V)$ where C, U, Q and q are as in the proof of Theorem 5.2.

We can encode an election over a set of candidates C as a matrix $\{a(i,j)\}_{i,j \in C}$, where for all $i \neq j$ the entry $a(i,j)$ equals the number of voters that prefer i to j. By McGarvey's theorem [14], for some $n = \text{poly}(m)$ we can efficiently construct a preference profile $\mathcal{R} = (R_1, \ldots, R_n)$ corresponding to the following matrix:

- $a(u_i, u_{i+1}) = b + 1$ for $i = 1, \ldots, q$;
- $a(u_i, u_{i+1}) = b - 1$ for $i = q + 1, \ldots, \ell$;
- $a(x_i, y_i) = a(y_i, u_i) = b$ for $i = 1, \ldots, s$;
- $a(c, w) = a(d_1, c) = b$, $a(w, d_1) = b + 1$;
- $a(c, d_2) = g + 1$;
- $a(x, y) = g + b - a(y, x)$ if $a(y, x)$ has been defined above;
- $a(x, y) = \frac{b+g}{2}$ for all other pairs $(x, y) \in C \times C$,

where $u_{\ell+1} := u_1$, $b < m$, $g > 2m$, and $b + g = n$. Now, consider an election with the set of candidates C and a set of voters $V = \{v_1, \ldots, v_{n+1}\}$, where for $i \leq n$ the preferences of the i-th

voter are given by R_i, and the preferences of the last voter (who is also the manipulating voter) are given by

$$c \succ w \succ d_1 \succ d_2 \succ x_1 \succ y_1 \succ \ldots \succ x_s \succ y_s \succ u_\ell \ldots \succ u_1.$$

Observe that if v_{n+1} votes truthfully, then $a(w, d_1) = b + 2$, $a(w, x) > b + 2$ for all $x \in C \setminus \{d_1\}$, while the Maximin score of any other candidate is at most $b + 1$, so w is the election winner. Hence, a manipulation is successful if and only if v_{n+1} manages to vote so that c gets elected. It can be shown that this is possible if and only if we have started with a "yes"-instance of 3-SAT; we omit the proof due to lack of space. □

THEOREM 5.5. *$T \circ \text{Copeland}^\alpha$-MANIPULATION is NP-hard for any $\alpha \in [0, 1]$.*

PROOF. Given a 3-SAT formula $\mathcal{C}$, we construct an election $E = (C, V)$ where C, U, Q and q are the same as in the proof of the Theorem 5.2. We say that x *safely wins* a pairwise election against y (and y *safely loses* a pairwise election against x) if at least $\frac{n}{2} + 2$ voters prefer x to y. For any candidate $x \in C$, let $\text{SW}(x)$ and $\text{SL}(x)$ denote the number of pairwise elections that x safely wins and safely loses, respectively. By McGarvey's theorem [14], we can construct a preference profile $\mathcal{R} = (R_1, \ldots, R_n)$ with the following properties:

- $\text{SW}(u_i) = \frac{m+1}{2}$, $\text{SL}(u_i) = \frac{m-3}{2}$ for $i = 1, \ldots, q$;
- $\text{SW}(u_i) = \frac{m-1}{2}$, $\text{SL}(u_i) = \frac{m-1}{2}$ for $i = q + 1, \ldots, \ell$;
- $\text{SW}(x_i) = \text{SW}(y_i) = \frac{m-1}{2}$ for $i = 1, \ldots, s$;
- $\text{SL}(x_i) = \text{SL}(y_i) = \frac{m-3}{2}$ for $i = 1, \ldots, s$;
- $\text{SW}(c) = \frac{m-1}{2}$, $\text{SL}(c) = \frac{m-3}{2}$;
- $\text{SW}(w) = \frac{m+1}{2}$, $\text{SL}(w) = \frac{m-9}{2}$;
- $\text{SW}(d_1) = \lfloor \frac{\ell-q+s-1}{2} \rfloor$, $\text{SL}(d_1) = \lceil \frac{\ell+q+3s+5}{2} \rceil$;
- $\text{SW}(d_2) = \lceil \frac{\ell-q+s-1}{2} \rceil$, $\text{SL}(d_2) = \lfloor \frac{\ell+q+3s+5}{2} \rfloor$;
- there is a tie between c and w, w and d_1, w and d_2, and x_i and y_i for $i = 1, \ldots, s$.

It is easy to check that for each candidate the total number of wins, losses and ties that involve him equals $m - 1$; in particular, $\ell - q + s - 1$ and $\ell + q + 3s + 5$ have the same parity, so $\text{SW}(d_1) + \text{SL}(d_1) = \text{SW}(d_2) + \text{SL}(d_2) = \ell + 2s + 2 = m - 2$. Moreover, the total number of wins equals the total number of losses. Thus, such a profile can indeed be constructed.

Now, consider an election with the set of candidates C and a set of voters $V = \{v_1, \ldots, v_{n+1}\}$, where for $i \leq n$ the preferences of the i-th voter are given by R_i, and the preferences of the last voter (who is also the manipulating voter) are given by

$$c \succ w \succ d_1 \succ d_2 \succ x_1 \succ y_1 \succ \ldots \succ x_s \succ y_s \succ u_\ell \ldots \succ u_1.$$

If v_{n+1} votes truthfully, w wins. Hence, a manipulation is successful if and only if v_{n+1} manages to vote so that c gets elected. It can be shown that this is possible if and only if we started with a "yes"-instance of 3-SAT; we omit the proof due to lack of space. □

6. CONCLUSIONS AND FUTURE WORK

We have explored the complexity of manipulating many common voting rules under randomized tie-breaking as well as under arbitrary polynomial-time tie-breaking procedures. Our results for randomized tie-breaking are far from complete, and a natural research direction is to extend them to other voting rules, such as Copeland or Bucklin, as well as to the Maximin rule with general utilities. Other interesting questions include identifying natural tie-breaking rules that make manipulation hard and extending our results to multi-winner elections.

7. ACKNOWLEDGMENTS

This research was supported by National Research Foundation (Singapore) under grant 2009-08, and by NTU SUG (Edith Elkind).

8. REFERENCES

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

[2] J. J. Bartholdi, III and J. B. Orlin. Single transferable vote resists strategic voting. *Social Choice and Welfare*, 8(4):341–354, 1991.

[3] V. Conitzer and T. Sandholm. Universal voting protocol tweaks to make manipulation hard. In *IJCAI'03*, pp. 781–788, 2003.

[4] V. Conitzer, T. Sandholm, and J. Lang, When ere elections with few candidates hard to manipulate?. *J. ACM*, 54:1–33, 2007.

[5] J. Davies, G. Katsirelos, N. Narodytska, and T. Walsh. An empirical study of Borda manipulation. In *COMSOC'10*, pp. 91–102, 2010.

[6] Y. Desmedt, E. Elkind. Equilibria of plurality voting with abstentions. In ACM EC'10, pp. 347–356, 2010.

[7] E. Elkind, H. Lipmaa, Hybrid voting protocols and hardness of manipulation. In *ISAAC'05*, pp. 206–215, 2005.

[8] P. Faliszewski, E. Hemaspaandra, and H. Schnoor. Copeland voting: ties matter. In *AAMAS'08*, pp. 983–990, 2008.

[9] P. Faliszewski, A. Procaccia. AI's war on manipulation: are we winning? In *AI Magazine*, 31(4):53–64, 2010.

[10] E. Friedgut, G. Kalai, and N. Nisan. Elections can be manipulated often. In *FOCS'08*, pp. 243–249, 2008.

[11] M. R. Garey, D. S. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness, 1979.

[12] A. F. Gibbard. Manipulation of voting schemes: a general result. *Econometrica*, 41:597–601, 1973.

[13] M. Isaksson, G. Kindler, and E. Mossel. The geometry of manipulation—a quantitative proof of the Gibbard–Satterthwaite theorem. In *FOCS'10*, pp. 319–328, 2010.

[14] D. C. McGarvey. A Theorem on the Construction of Voting Paradoxes. *Econometrica*, 21(4):608–610, 1953.

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

[16] M. A. 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.

[17] T. Walsh. Where are the really hard manipulation problems? The phase transition in manipulating the Veto rule. In *IJCAI'09*, pp. 324–329, 2009.

[18] L. Xia and V. Conitzer, A sufficient condition for voting rules to be frequently manipulable. In *EC'08*, pp. 99–108, 2008.

[19] L. Xia, V. Conitzer, A. Procaccia. A scheduling approach to coalitional manipulation. In *EC'10*, pp. 275–284, 2010.

[20] L. Xia, M. Zuckerman, A. Procaccia, V. Conitzer, and J. Rosenschein. Complexity of unweighted coalitional manipulation under some common voting rules. In *IJCAI'09*, pp. 348–352, 2009.