# Algorithms for Destructive Shift Bribery

Andrzej Kaczmarczyk
AGH University
Krakow, Poland
kaczmarczyk.andrzej@gmail.com

Piotr Faliszewski
AGH University
Krakow, Poland
faliszew@agh.edu.pl

## ABSTRACT

We study the complexity of Destructive Shift Bribery. In this problem, we are given an election with a set of candidates and a set of voters (ranking the candidates from best to worst, each), a despised candidate $d$ (typically, one of the current winners), a budget $B$, and prices for shifting $d$ down in voters' rankings. The goal is to ensure that $d$ is not a winner of the election. We show that this problem is polynomial-time solvable for all scoring protocols (encoded in unary) and the Maximin rule, but is NP-hard for the Copeland rule. This contrasts with the results for the constructive setting (known from the literature), for which the problem is polynomial-time solvable for $k$-Approval family of rules, but is NP-hard for the Borda, Copeland, and Maximin rules.

## Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent systems*

## Keywords

algorithms, computational complexity, voting, campaign management, bribery

## 1. INTRODUCTION

We study the complexity of the destructive variant of the Shift Bribery problem. We consider the family of all (unary encoded) scoring protocols (including the Borda rule and all $k$-Approval rules), and the Copeland and maximin rules. It turns out that for all of them—except for the Copeland rule—the problem can be solved in polynomial time. This stands in sharp contrast to the constructive case, where the problem is NP-hard [14] (and hard in the parameterized sense [4]) for the Borda, Copeland, and Maximin rules (and is in P for the $k$-Approval family of rules).

The Shift Bribery problem was introduced by Elkind et al. [14] to model (a kind of) campaign management problem in elections. The problem is as follows: We are given an election, i.e., a set of candidates and a set of voters that rank the candidates from the most to the least desirable one, a preferred candidate $p$, a budget $B$, and the costs for shifting $p$ up in voters' rankings. Our goal is to decide if it is possible to ensure $p$'s victory by shifting $p$ up, without exceeding the budget. In this paper we study a destructive variant of the problem, where the goal is to ensure that a given despised candidate $d$ loses the election, by shifting him or her down in voters' rankings (but, again, each shift comes at a cost and we cannot exceed the given budget).

Studying destructive variants of election problems where the goal is to change the current winner (such as manipulation [8], control [20], and bribery [16, 21, 6, 32]) is a common practice in computational social choice, but our setting is somewhat special. So far, all the destructive variants of the problems were defined by changing the goal from "ensure the victory of the preferred candidate" to "preclude the despised candidate from winning," but the set of available actions remained unaffected e.g. in both the constructive and the destructive problem of control by adding voters, we can add some voters from a given pool of voters and in both the constructive and the destructive bribery problem, we can pay voters to change their votes in some way. In our case, we feel that it is natural to divert from this practice and change the ability of "shifting the distinguished candidate up" to the ability of "shifting the distinguished candidate down". Below we explain why.

If, when defining our destructive problem, we stuck with the "ability to shift up," as in the Constructive Shift Bribery problem, we would get the following problem: Ensure that a given despised candidate does not win the election by shifting him or her forward in some of the votes (without exceeding the budget). However, for a monotone voting rule, if a candidate is already winning an election, then shifting him or her forward certainly cannot preclude him or her from winning. This would make our problem interesting only for a relatively small set of nonmonotone rules.[1] Further, the problem certainly would not be modeling what one would intuitively think of as negative, destructive campaigning. Thus, while it certainly would be interesting to study how to exploit nonmonotonicity of rules such as STV and Dodgson to preclude someone from winning, it would not be the most practical way of defining Destructive Shift Bribery.

**Related Work.** In recent years, Constructive Shift Bribery received quite some attention. The problem was defined by Elkind et al. [14], as a simplified variant of Swap Bribery (which itself received some attention, e.g., in the

**Appears in:** *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2016), J. Thangarajah, K. Tuyls, C. Jonker, S. Marsella (eds.), May 9–13, 2016, Singapore.*

[1]Nonetheless, there are interesting nonmonotone rules, such as the single transferable rule (STV) and the Dodgson rule.

works of Dorn and Schlotter [12], Faliszewski et al. [17], and papers regarding combinatorial domains, such as those of Mattei et al. [23] and Dorn and Krüger [11]; importantly, Shiryaev et al. [31] studied a variant of DestructivSwap Bribery and we comment on their work later). Elkind et al. have shown that Shift Bribery is NP-hard for Borda, Copeland, and Maximin voting rules, but polynomial-time solvable for the $k$-Approval family of rules. They also gave a 2-approximation algorithm for the case of Borda, that was later generalized to the case of all scoring rules by Elkind and Faliszewski [13]. Chen et al. [4] considered parametrized complexity of Constructive Shift Bribery, and have shown a varied set of results (in general, parametrization by the number of positions by which the preferred candidate is shifted tends to lead to FPT algorithms, parametrization by the number of affected voters tends to lead to hardness results, and parametrization by the available budget gives results between these two extremes). Very recently, Bredereck et al. [5] studied the complexity of Combinatorial Shift Bribery, where a single shift action can affect several voters at a time. The paper is quite related to ours, because it was the first one in which shifting a candidate backward was possible (albeit as a negative side effect, since the authors study the constructive setting).

Shift Bribery falls into the general group of bribery-related problems, that were first studied by Faliszewski et al. [15] (see also the work of Hazon et al. [19] for a similar problem), and that received significant attention within computational social choice literature (see the survey of Faliszewski and Rothe [18]). Briefly put, in the regular Bribery problem, the goal is to ensure that a given candidate is a winner of an election by modifying—in an arbitrary way—up to $k$ votes, where $k$ is part of the input. Many types of bribery problems were already studied, including—in addition to Swap and Shift Bribery—Support Bribery [30], Extension Bribery [2, 17], and others (e.g., in judgment aggregation [1], and in the setting of voting in combinatorial domains [23, 11, 22]). However, from our point of view the most interesting variant of the problem is Destructive Bribery first studied by Faliszewski et al. [16], and then later—independently—by Magrino et al. [21] and Cary [6], and followed by Xia [32] and Dey and Narahari [10], under the name of Margin of Victory. The idea behind the Margin of Victory problem is that it can be very helpful in validating election results: If it is possible to change the election result by changing (bribing) relatively few votes, then one may suspect that—possibly—the election was tampered with. Bribery problems are also related to lobbying problems [7, 3, 25].

A variant of Destructive Swap Bribery was studied by Shiryaev et al. in their work about robustness of winners [31]. They presented an analysis of the case where every swap has a unit price and a budget, which cannot be exceeded, is given. Authors show that the problem is easy for scoring protocols and the Condorcet rule. This work is very closely related to ours but the definition of the problem is somewhat different (more general types of swaps, but less general price functions).

**Our Contribution.** We believe that Destructive Shift Bribery is worth studying for three main reasons. First, it simply is a natural variant of the Constructive Shift Bribery problem, which received quite some attention already. Second, it models natural negative campaigning actions, aimed at decreasing the popularity of a given candidate. Third, it serves a similar purpose as the Margin of Victory problem [21, 32]: If it is possible to preclude a given candidate from winning through a low-cost destructive shift bribery, then it can be taken as a signal that the election was tampered with, or that some agent performed (possibly) an illegal form of campaigning. Below we summarize the main contributions of this paper:

1. We define the Destructive Shift Bribery problem and justify why a definition that diverts from the usual way of defining destructive election problems is appropriate in this case.

2. We show that Destructive Shift Bribery is a significantly easier problem than constructive shift bribery. To this end, we show polynomial time algorithms for destructive shift bribery for $k$-Approval family of rules, for Borda, and for Maximin.

3. We show that in spite of our easiness results, there still are voting rules for which the problem is computationally hard. We exemplify this by proving NP-hardness for the case of Copeland$^\alpha$ family of rules.

We believe that Destructive Shift Bribery is an interesting problem and it is worthwhile to study it in detail.

## 2. PRELIMINARIES

For each positive integer $t$, by $[t]$ we mean the set $\{1, \dots, t\}$. We assume the reader is familiar with standard notions regarding algorithms and complexity theory, as presented in the textbook of Papadimitriou [29]. Occasionally, we will also use basic knowledge regarding parametrized complexity (see, e.g., the textbooks of Niedermeier [26] and Cygan et al. [9]). Below we present the necessary background regarding elections, and define our problem.

### 2.1 Elections and Election Rules

An election is a pair $E = (C, V)$, where $C = \{c_1, ..., c_m\}$ is a set of candidates and $V = \{v_1, ..., v_n\}$ is a multiset of voters. Each voter is associated with his or her preference order $\succ_i$, i.e., a strict ranking of the candidates from the best to the worst (according to this voter). For example, we may have election $E = (C, V)$ with $C = \{c_1, c_2, c_3\}$ and $V = \{v_1, v_2\}$, where $v_1$ has preference order $v_1 \colon c_1 \succ c_2 \succ c_3$. If $c$ is a candidate and $v$ is a voter, we write $\mathrm{pos}_v(c)$ to denote the position of $c$ in $v$'s ranking (e.g., in the preceding example we would have $\mathrm{pos}_{v_1}(c_1) = 1$). Given an election $E = (C, V)$, by $N_E(c, d)$, where $c$ and $d$ are two candidates, we mean the number of voters who prefer $c$ over $d$.

An election rule $\mathcal{R}$ is a function that given an election $E = (C, V)$ outputs a set $W \subseteq C$ of tied election winners (typically, we expect to have a single winner, but due to symmetries in the profile, it is necessary to allow for the possibility of ties). We use the unique-winner model, i.e., we require a candidate to be the only member of $\mathcal{R}(E)$ to be considered $E$'s winner (see the works of Obraztsova et al. [27, 28] for various other tie-breaking mechanisms and algorithmic consequences of implementing them; there are situations where the choice of the tie-breaking rule affects the complexity of election-related problems).

We consider the following voting rules (for the description below, we consider election $E = (C, V)$ with $m$ candidates;

for each rule we describe the way it computes candidates' scores, so that the candidates with the highest score are the winners):

**Scoring protocols.** A scoring protocol is defined through vector $\alpha = (\alpha_1, \dots, \alpha_m)$ of nonincreasing, nonnegative integers. The $\alpha$-score of a candidate $c \in C$ is defined as $\sum_{v \in V} \alpha_{\mathrm{pos}_v(c)}$. The most popular scoring protocols include the family of $k$-Approval rules (for each $k$, $k$-Approval scoring protocol is defined through vector of $k$ ones followed by zeros) and the Borda rule, defined by vector $(m-1, m-2, \dots, 0)$. 1-Approval is known as the Plurality rule.

**Copeland** . Let $\alpha$, $0 \le \alpha \le 1$, be a rational number. The score of a candidate $c$ in Copeland$^\alpha$ election is defined as:

$$|\{d \in C \setminus \{c\} \mid N_E(c,d) > N_E(d,c)\}| \\ +\alpha|\{d \in C \setminus \{c\} \mid N_E(c,d) = N_E(d,c)\}|.$$

In other words, candidate $c$ receives one point for each candidate that he or she defeats in their head-to-head contest (i.e., to which $c$ is preferred by a majority of voters) and $\alpha$ points for each candidate with whom $c$ ties their head-to-head contest.

**Maximin.** The Maximin score of candidate $c$ is defined as $\min_{d \in C \setminus \{c\}} N_E(c,d)$.

We write $\mathrm{score}_E(c)$ to denote the score of candidate $c$ in election $E$ (the election rule will be clear from the context).

## 2.2 Destructive Shift Bribery

The Destructive Shift Bribery problem for a given election rule $\mathcal{R}$ is defined as follows. We are given an election $E = (C, V)$, a despised candidate $d \in C$ (typically, the current election winner), budget $B$ (a nonnegative integer), and the prices for shifting $d$ backward for each of the voters (see below). The goal is to ensure, by shifting $d$ backward, that he or she is not the unique $\mathcal{R}$-winner of the resulting election (without exceeding the budget).

We model the "prices for shifting $d$ backward" as destructive shift-bribery price functions. Let us fix an election $E = (C, V)$, with $C = \{c_1, ..., c_m\}$, $V = \{v_1, ..., v_n\}$, and let the despised candidate be $d$. Let $v$ be some voter and let $j = \mathrm{pos}_v(d)$. Function $\rho \colon \mathbb{N} \to \mathbb{N} \cup \{+\infty\}$ is a destructive shift-bribery price function for voter $v$ if it satisfies the following conditions:

1. $\rho(0) = 0$,
2. for each two $i, i'$, $i < i' \le m - j$, it holds that $\rho(i) \le \rho(i')$ for $i < i' \le m - j$, and
3. $\rho(i) = +\infty$ for $i > m - j$

For each $i$, we interpret the value $\rho(i)$ as the price for shifting $d$ down by $i$ positions on $v$'s preference order. The value $+\infty$ indicates that shifting $d$ below the last position is impossible. We assume that in each instance, the price functions are encoded by simply listing their values for all arguments for which it is not $+\infty$.

**Example 1.** *Let us consider an instance of* Destructive Shift Bribery*, for the Borda rule, with the following election:*

$$v_1 \colon b \succ a \succ c \succ d, \qquad v_2 \colon d \succ b \succ a \succ c,$$
$$v_3 \colon d \succ c \succ a \succ b, \qquad v_4 \colon d \succ a \succ b \succ c.$$

*We take $d$ to be the despised candidate, and assume that each voter has the same price function, $\rho(i) = i$. In this election, candidate $d$ wins with 9 points (the second-best candidate, $a$, has 6 points). However, if we shift $d$ two positions backward in $v_4$'s preference order, then $d$'s score will decrease to 7 and $a$'s score will increase to 7. Thus, $d$ will no longer be the unique winner. Thus, for budget $B = 2$ this would be a "yes"-instance of* Shift Bribery*, whereas for $B = 1$ it would be a "no"-instance.*

## 3. RESULTS

We now present our main results regarding the Destructive Shift Bribery problem. We show polynomial-time algorithms for the $k$-Approval family of rules, the Borda rule, all scoring protocols (provided that they are encoded in unary or that the destructive shift bribery price functions are encoded in unary), and the Maximin rule. For the Copeland$^\alpha$ family of rules, we prove NP-hardness.

## 3.1 The k-Approval Family of Rules

We start with the $k$-Approval family of rules. In this case, our algorithm is very simple: If $d$ is the despised candidate then in each vote we should either shift him or her from one of the top $k$ positions (where each candidate receives a single point) to the $(k+1)$'st position (where he or she would receive no points), in effect shifting the candidate previously at the $(k+1)$'st position one place up, or we should not shift $d$ at all. Choosing which action to do for each particular voter is easy via the following greedy/brute-force algorithm.

**Theorem 1.** *For each $k \in \mathbb{N}$, the* Destructive Shift Bribery *problem for the $k$-Approval rule is in* P.

*Proof.* Let $E = (C, V)$ be the input election with $C = \{c_1, \dots, c_m\}$ and $V = \{v_1, \dots, v_n\}$, let $d \in C$ be the despised candidate, let $B$ be the budget, and let $(\rho_1, \dots, \rho_n)$ be the destructive shift-bribery price functions for the voters. Our algorithm works as follows.

For every candidate $c \in C \setminus \{d\}$, we test if it is possible to guarantee that the score of $c$ is at least as high as that of $d$, by spending at most $B$ units of budget, as follows:

1. We partition the voters into three groups $V_{d,c}$, $V_d$, and $V'$ such that: $V_{d,c}$ contains exactly the voters that rank $c$ on the $(k+1)$'st position and that rank $d$ above $c$, $V_d$ contains the remaining voters that rank $d$ among top $k$ positions, and $V'$ contains the other remaining voters.
2. We guess two numbers, $a$ and $b$, such that $|V_{d,c}| \le a$ and $|V_d| \le b$.
3. We pick $a$ voters from $V_{d,c}$ for which shifting $d$ to the $(k+1)$'st position is least expensive, and perform the shift. Similarly, we pick $b$ voters from $V_d$ for which shifting $d$ to the $(k+1)$'st position is least expensive and perform the shift.

4. If $d$ is not the unique winner in the resulting election and the total cost of performed shifts is smaller than or equal to budget $B$, then we accept. Otherwise, we either try a different candidate $c$ or different values of $a$ and $b$. After trying all possible combinations, we reject.

The algorithm runs in polynomial time. It requires trying at most $O(m)$ different candidates and $O(n^2)$ different values of $a$ and $b$. All the other parts of the algorithm require polynomial time (in fact, a careful implementation can achieve running time $O(mn^2)$).

A proof of correctness is pretty straightforward. Certainly, it is never beneficial to shift $d$ below position $k+1$. Further, an optimal solution after which some candidate $c$ has at least as high a score as $d$ can consist solely of actions that shift $d$ backward from one of the top $k$ positions to the $(k+1)$'st one, in effect either promoting $c$ to the $k$'th position (shifts in voter group $V_{d,c}$), or promoting some other candidate (shifts in voter group $V_d$). We simply guess how many of each type of the shifts to perform and execute the least costly ones. □

The polynomial time algorithm for Constructive Shift Bribery, due to Elkind et al. [14], is based on a similar idea.

## 3.2 The Borda Rule and All Scoring Rules

Let us now consider the Borda rule and then move to the discussion of all scoring rules. For the case of the Borda rule, we also obtain a polynomial-time algorithm, but this time we resort to dynamic programming.

**Theorem 2.** *The* Destructive Shift Bribery *problem for the Borda rule is in* P.

*Proof.* Let $E = (C, V)$ be the input election with $C = \{c_1, \dots, c_m\}$ and $V = \{v_1, \dots, v_n\}$, let $d \in C$ be the despised candidate, let $B$ be the budget, and let $(\rho_1, \dots, \rho_n)$ be the destructive shift-bribery price functions for the voters. As in the case of the proof of Theorem 1, we give an algorithm that first guesses some candidate $c \in C \setminus \{d\}$ and then checks if it is possible to ensure—by shifting $d$ backward without exceeding budget $B$—that $c$ has at least as high a score as $d$. The algorithm performing this test is based on dynamic programming. However, before we describe it, we need to introduce some additional notation.

We fix $c$ to be the candidate that we want to have score at least as high as $d$. For each $j \in [n]$ and each $k \in [m]$, we write $P(j,k)$ to denote the cost of shifting candidate $d$ on $v_j$'s preference order by $k$ positions down (if moving $d$ by $k$ positions is impossible, we have $P(j,k) = \infty$). Further, for each $j \in [n]$ and $k \in [m]$, we set $A(j,k)$ to be 1 if $v_j$ ranks $c$ among first $k$ positions below $d$, and we set it to be 0 otherwise. Finally, we write $s$ to denote the difference between the scores of $d$ and $c$, i.e., $s = \mathrm{score}_E(d) - \mathrm{score}_E(c)$ (it must be that $s > 0$; otherwise we could accept immediately since $d$ would not be the unique winner of the election).

For each $j \in [n]$ and each positive integer $k$, we define $f(j,k)$ to be the smallest cost for shifting $d$ backward in the preference orders of the voters from the set $\{v_1 \dots, v_j\}$, so that, if $E'$ is the resulting election, it holds that:

$$s - (\mathrm{score}_{E'}(d) - \mathrm{score}_{E'}(c)) \geq k.$$

In other words, $f(j,k)$ is the lowest cost of shifting $d$ backward in the preference orders of voters from the set $\{v_1, \dots, v_j\}$ so that the relative score of $c$ with respect to $d$ increases by at least $k$ points. Our goal is to compute $f(n,s)$; if it is at most $B$ then it means that we can ensure that $c$ has at least as high a score as $d$ by spending at most $B$ units of the budget.

Before we give an algorithm for computing the values $f(\cdot,\cdot)$, we make the following observation.

**Observation 1.** *Shifting candidate $d$ backward by some $k$ positions decreases the score of $d$ by $k$ points and, if $d$ passes $c$, increases the score of $c$ by one point. In effect, relative to $d$, $c$ gains, respectively, $k$ or $k+1$ points.*

Now, we can express function $f$ as follows. We have that $f(0,0) = 0$ and for each positive $k$ we have $f(k,0) = \infty$ (for technical reasons, whenever $k < 0$, we take $f(j,k) = 0$). For each $j \in [n]$, we have:

$$f(j,k) = \min_{k' \leq k} f(j-1, k-(k'+A(j,k'))) + P(j,k').$$

To explain the formula, we make the following observations:

1. The minimum is taken over the value $k'$; $k'$ gives the number of positions by which we shift $d$ backward in vote $v_j$.

2. When we shift $d$ backward by $k'$ positions, relative to $d$, candidate $c$ gains $k' + A(j,k')$ points.

3. The cost of this shift is $P(j,k')$.

Based on these observations, we can conclude that the formula is correct. It is clear that based on this formula and standard dynamic programming techniques, we can compute $f(n,s)$ in polynomial time with respect to $n$ and $m$. This means that we can test in polynomial time, for a given candidate $c$, if it is possible to ensure that $c$'s score is at least as high as that of $d$.

Our algorithm for the Destructive Shift Bribery for Borda simply considers each candidate $c \in C \setminus \{d\}$ and tests if, within budget $B$, it is possible to ensure that $c$ has at least as high a score as $d$. If this test succeeds for some $c$, we accept. Otherwise we reject. □

A careful inspection of the above proof shows that there is not much in it that is specific to the Borda rule (as opposed to other scoring protocols). Indeed, there are only the following two dependencies:

1. The fact that the difference between the scores of candidate $d$ and candidate $c$ (value $s$) can be bounded by a polynomial of the number of voters and the number of candidates (dynamic programming requires us to store a number of values of the function $f$ that is proportional to $s$, so it is important that this value is polynomially bounded).

2. The way we compute the increase of the score of $c$, relative to $d$, in the recursive formula for $f(j,k)$.

Since it is easy to modify the formula for $f(j,k)$ to work for an arbitrary scoring protocol $\alpha$, we have the following corollary (the assumption about unary encoding of the input scoring protocol ensures that the first point in the above list of dependencies is not violated).

**Corollary 1.** *There exists an algorithm that given as input a scoring protocol $\alpha$ (for $m$ candidates) encoded in unary and an instance $I$ of the* Destructive Shift Bribery *problem (with the same number $m$ of candidates), tests in polynomial time if $I$ is a "yes"-instance for the voting rule defined by the scoring protocol $\alpha$.*

What can we do if, in fact, our scoring protocol is impractical to encode in unary (e.g., if our scoring protocol is of the form $(2^{m-1}, 2^{m-2}, \ldots, 1)$)? In this case, it is easy to show the following result.

**Corollary 2.** *There exists an algorithm that given as input a scoring protocol $\alpha$ (for $m$ candidates) and an instance $I$ of the* Destructive Shift Bribery *problem (with the same number $m$ of candidates), with price functions encoded in unary, tests in polynomial time if $I$ is a "yes"-instance for the voting rule defined by the scoring protocol $\alpha$.*

To prove this result, we use the same argument as before, but now function $f$ has slightly different arguments: $f(j,t)$ is the maximum increase of the score of $c$, relative to $d$, that one can achieve by spending at most $t$ units of budget (using such "dual" formulation is standard for bribery problems and was applied, e.g., by Faliszewski et al. [15, Theorem 3.8] in the first paper regarding the complexity of bribery problems).

On the other hand, if the scoring protocol is encoded in binary and the price functions are encoded in binary, and the scoring protocol is part of the input, then the problem is NP-complete. (This follows via a simple reduction from the standard Partition problem.)

## 3.3 The Copeland Rule

Let us now move on to the case of Copeland$^\alpha$ family of rules. We show that irrespective of the choice of $\alpha$, $0 \leq \alpha \leq 1$, the Destructive Shift Bribery problem is NP-complete for Copeland$^\alpha$. We give a reduction from the classic NP-complete problem, Exact Cover by 3-Sets (X3C).

**Definition 1.** *In the* X3C *problem, we are given a set $U = \{u_1, u_2, ..., u_{3k}\}$ and family $S = \{s_1, s_2, \ldots, s_n\}$ of three-element subsets of $U$. We ask if there exists a family $S'$, $S' \subseteq S$, such that $|S'| = k$ and $U = \bigcup_{s_i \in S'} s_i$.*

We use the following notation in the proof below. By putting some subset of $U$ in a preference order (where $U$ is the set from an X3C instance), we mean listing the contents of the set in the order of increasing indices (i.e., if we wrote $U$ in a preference order, it would mean $u_1 \succ u_2 \succ \cdots \succ u_{3k}$). In the proof below, we sometimes also list sets other than subsets of $U$ within preference orders; in their case we also mean listing their members in the order of increasing indices.

**Theorem 3.** *For each (rational) value of $\alpha$, $0 \leq \alpha \leq 1$, the* Destructive Shift Bribery *problem for the Copeland$^\alpha$ rule is* NP*-complete.*

*Proof.* We can immediately claim that our problem belongs to NP (we can guess in which votes to shift the despised candidate and by how many positions; checking if this ensures that the despised candidate is not the unique winner is an easy, polynomial-time task). It remains to show that the problem is NP-hard.

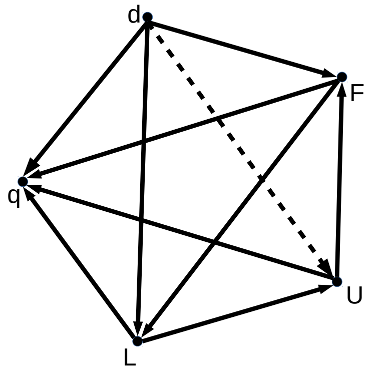

**Figure 1: A graph presenting the relations between candidates in the constructed election.**

Let us fix a value of $\alpha$. We give a reduction from the X3C problem. Consider an instance $I$ of X3C with $U = \{u_1, \ldots, u_{3k}\}$ and $S = \{s_1, \ldots, s_n\}$. We construct an election $E = (C, V)$ as follows. We let the set of candidates be $C = \{d, q\} \cup U \cup F \cup L$, where $F = \{f_1, f_2, ..., f_{3k}\}$, $L = \{l_1, l_2, ..., l_{3k}\}$ are dummy candidates needed for our construction. We introduce the following voters:

1. For each $i \in [n]$, we introduce a single voter, $v_i$, with preference order $v_i \colon d \succ s_i \succ q \succ U \setminus s_i \succ F \succ L$, and a single voter $v_i'$ whose preference order is the reverse of that of $v_i$.

2. We additionally introduce $6k + 1$ voters with the following preference orders:

| | |
|---|---|
| $k$ votes: | $U \succ d \succ F \succ L \succ q$ |
| $k$ votes: | $U \succ d \succ F \succ L \succ q$ |
| $k$ votes: | $d \succ L \succ U \succ F \succ q$ |
| $k$ votes: | $q \succ d \succ L \succ U \succ F$ |
| $k$ votes: | $F \succ L \succ U \succ q \succ d$ |
| $k$ votes: | $d \succ F \succ L \succ U \succ q$ |
| 1 vote : | $q \succ d \succ U \succ F \succ L$ |

We set the budget to be $B = k$, and we set the price functions as follows. For each of the voters $v_1, \ldots, v_n$, we use price function $\rho$ such that $\rho(0) = 0$, $\rho(1) = \rho(2) = \rho(3) = 1$, and $\rho(t) = B + 1$ for all $t \geq 4$. For all the other voters, we use price function $\rho'$ such that $\rho'(0) = 0$ and $\rho'(t) = B + 1$ for all $t \geq 1$.

To clarify the relations between the candidates in our election, we provide Figure 1 with the following graph: The vertices correspond to particular candidates or groups of candidates and the edges are pointing from a candidate (or a group of candidates) that is winning the given head-to-head contest to a candidate (or a group of them) that is losing (e.g., an edge from $d$ to $F$ means that $d$ wins his or her head-to-head contests against each member of $F$). Solid edges represent the results of head-to-head contests that cannot change by applying shifts within given budget.

Figure 1 does not show results of the head-to-head contests between candidates within particular sets. Nonetheless, given the graph, the description of the voters, and the convention regarding putting sets in preference orders (see the paragraph just before the theorem statement), it is straightforward to calculate the scores of all the candidates. We present them in Table 1 (and below explain how it to verify these values easily; note that the number of voters is odd and so the value of $\alpha$ is irrelevant).

We see that $d$ wins with every other candidate and $q$ loses with everyone. For candidates from $U$, $F$, and $L$, the situation is symmetric so let us consider some candidate $u_i \in U$. He or she wins their head-to-head contest with candidate $q$ and all the candidates from $L$. This gives him or her $3k+1$ points. Moreover, $u_i$ wins with all the candidates $u_j \in U$ such that $j > i$. This give him or her $3k - i$ points. Altogether, the score of $u_i$ is $6k + 1 - i$.

We set up the election in such a way that the only results of head-to-head elections that can change due to shifts (within the budget) are between $d$ and members of $U$. In Table 2 we summarize the values of function $N_E(\cdot,\cdot)$. The major conclusion from Table 2 is as follows:

$$N(d, u_i) = N(u_i, d) + 1 \qquad (1)$$

Based on the above discussion, we now make a series of observations:

**Observation 2.** *The only voters in whose preference orders $d$ can be shifted without exceeding the available budget are voters $v_1, \dots, v_n$.*

**Observation 3.** *Only the candidates from $U$ can have their scores increased as a result of (within budget) shifts, and only $d$ can have its score decreased by such shifts (this is, in particular, due to Equation* (1)*).*

**Observation 4.** *We can shift $d$ in the preference orders of at most $k$ voters. Each time, we can shift $d$ below, at most,* 3 *(distinct) candidates from $U$. This means that $d$ can be shifted below at most $3k$ distinct candidates from $U$. (This is due to the value of the budget, the nature of our price functions, and the structure of preferences of the voters $v_1, \dots, v_n$.)*

**Observation 5.** *The lowest score that candidate $d$ can have after (within budget) shifts is $6k + 1$. This happens if and only if he or she passes each candidate from $U$ at least once.*

**Observation 6.** *The highest score a candidate from $U$ may have is $6k + 1$. This happens if $d$ is shifted below $u_1$.*

| candidate | score |
|---|---|
| $d$ | $9k+1$ |
| $q$ | $0$ |
| $u_i \in U$ | $6k+1-i$ |
| $f_i \in F$ | $6k+1-i$ |
| $l_i \in L$ | $6k+1-i$ |

**Table 1: Table containing scores of all of the candidates in the constructed election.**

| $N(\cdot,\cdot)$ | $d$ | $u_i \in U$ |
|---|---|---|
| $d$ | – | $n+3k+1$ |
| $u_i \in U$ | $n+3k$ | – |

**Table 2: Table presenting values of function $N$ for candidate $d$ and candidates in $U$.**

By combining the above observations, we see that $d$ can stop being the unique winner of our election if and only if there is a sequence of shifts, on the preference orders of the voters $v_1, \dots, v_n$, each moving $d$ by three positions down, such that $d$ passes each member of $U$. However, this is possible if and only if there is a solution for the input X3C instance. Since the reduction is computable in polynomial time, this completes the proof. $\square$

The above result and proof requires some comments. As a technical remark, the reader may wonder what the role of candidate $q$ is. For the result as we state it, it would be possible to simply remove him or her. However, our construction allows for a simple tweak: If we extended the price functions of voters $v_1, \dots, v_n$ to allow shifting $d$ backward one position more, to pass $q$, our proof would work for the case where we have to ensure that the resulting election has a unique winner (this is not a requirement in our model, but we find it appealing that the result appears to be immune to various tie-breaking tweaks).

The fact that Destructive Shift Bribery is NP-complete immediately raises the question about its parameterized complexity. While in this paper we, mostly, focus on classic complexity results, the proof of Theorem 3 allows us to make some conclusions. First, instead of reducing from X3C, we could have given a reduction from SetCover (a problem where neither the size of $U$ nor the sizes of the sets in the family $S$ are constrained, and where $k$—the number of sets one can use in the solution—is an additional part of the input) with only minor modifications. Since SetCover is well-known to be $W[2]$-hard for the parameter $k$ and since our construction uses budget $B = k$, we immediately get that Destructive Shift Bribery for Copeland$^\alpha$ rules is $W[2]$-hard when parameterized by the budget value, or when parameterized by the total number of affected voters (see the work of Chen et al. [4] for a discussion of these parameters; for other parameters considered by Chen et al., such as the number of unit shifts used or the number of voters, our proof does not apply and we would need additional constructions).

**Corollary 3.** *For each (rational) value of $\alpha$, $0 \le \alpha \le 1$, the* Destructive Shift Bribery *problem for the Copeland$^\alpha$ rule is $W[2]$-hard for the parametrization by the budged value, and for the parametrization by the number of affected voters.*

## 3.4 The Maximin Rule

In spite of the hardness results from the previous section, it certainly is not the case that Destructive Shift Bribery is hard for all Condorcet-consistent rules. We now show a polynomial time algorithm for the case of Maximin.

**Theorem 4.** *The* Destructive Shift Bribery *problem for the Maximin rule is in* P.

*Proof.* Let $E = (C, V)$ be the input election with $C = \{c_1, \dots, c_m\}$ and $V = \{v_1, \dots, v_n\}$, let $d \in C$ be the despised candidate, let $B$ be the budget, and let $(\rho_1, \dots, \rho_n)$ be the destructive shift-bribery price functions for the voters. Before we give our algorithm for the problem, we first identify a certain property that we can assume our solutions to have.

A solution for our instance of Destructive Shift Bribery can be understood as a vector of $n$ integers,

$(k_1, \dots, k_n)$, each specifying by how many positions we shift $d$ back on the preference order of the respective voter. So, let $(k_1, \dots, k_n)$ be some solution for our instance, and let $E' = (C, V')$ be an election that we obtain after, for each $i \in [n]$, we shift $d$ back by $k_i$ positions on $v_i$'s preference order. We denote the voters in $V'$ as $v'_1, \dots, v'_n$, where for each $i \in [n]$, $v'_i$ corresponds to $v_i$.

Let $w \in C \setminus \{d\}$ be a Maximin winner of $E'$ (we can assume that $w \neq d$ because $(k_1, \dots, k_n)$ is a solution for the problem). Also, let $t$ be a candidate such that $\text{score}_{E'}(d) = N_{E'}(d, t)$. That is, $t$ is the candidate that "implements" the score of $d$ in the final election. We claim that we can assume that for every voter $v'_i$ such that $k_i \neq 0$, it holds that either $v'_i$ ranks $w$ just above $d$, or $v'_i$ ranks $t$ just above $d$ (we refer to solutions where this holds as *tight*). Below we argue why this is the case.

Let us consider some voter $v'_i$. We consider the following cases:

1. If $v'_i$ ranks $d$ above both $w$ and $t$, then we modify the solution $(k_1, \dots, k_n)$ so that $k_i = 0$ (i.e., we undo whatever shift was applied to $v_i$). The reason is that since $d$ did not pass either $w$ or $t$, the shift within $v_i$ could not have affected the scores of $w$ and $d$ (due to our choice of $t$). Thus, even after undoing this shift, $w$ still has at least as high a score as $d$.

2. If $v'_i$ ranks $d$ below $w$, or below $t$, or below both of them, then we decrease $k_i$ so that $d$ either ranks $d$ just below $w$ or just below $t$ (whichever requires decreasing $k_i$ by a smaller value). If this is impossible (e.g., because the relative order of $d$, $t$, and $w$ is the same for both $v_i$ and $v'_i$) then we set $k_i = 0$. The reasoning is the same as in the above step: The introduced change does not affect the scores of $d$ and $w$.

The changes to the solution described above, when implemented for each of the voters, do not change the scores of $d$ and $w$, but may lower the cost of the solution. Thus it is correct to assume that whenever there is some solution for our instance, there is also a tight one.

Our algorithm proceeds by first guessing the candidates $w$ and $t$ (from $C \setminus \{d\}$) and then checks if there is a tight solution for them that does not exceed the budget. We use the following notation. For each two candidates $c_1, c_2$, we write $\mathit{Pref}(c_1, c_2)$ to denote the set of voters (from $V$) that prefer $c_1$ over $c_2$. Let us $\mathit{Pr}(v, c)$ be the cost of shifting candidate $d$ just behind $c$ in $v$'s preference order.

Since we are looking for a tight solution, we focus on voters in the set $\mathit{Pref}(d, w) \cup \mathit{Pref}(d, t) = \{v''_1, \cdots, v''_\ell\}$ (i.e., voters who prefer $d$ to at least one of $w$ and $t$). For given candidates $w, t$, $w \neq t$, and integers $j, x, y \in \{0, \dots, \ell\}$, we define function $f_{w,t}$, so that $f_{w,t}(j, x, y)$ is the lowest cost for shifting $d$ backward in the preference orders of the voters from $\{v''_1, \dots, v''_j\}$ so that $d$ moves $x$ times below $w$ and $y$ times below $t$. Given function $f_{w,t}$, we will be able to check for an existence of a tight solution for $w$ and $t$. Before we describe how to do it, let us argue that we can compute the values of function $f_{w,t}$ in polynomial time. To this end, we use dynamic programming.

First, we note that for each $j, x, y \in \{0, \dots, \ell\}$, we have $f_{w,t}(j, 0, 0) = 0$ and, if $x > 0$ or $y > 0$, $f_{w,t}(0, x, y) = \infty$ (to indicate the impossibility). For other values of the arguments, we express the value $f_{w,t}(j, x, y)$ recursively as follows. Let us fix some values of $j \in [\ell]$, and $x, y \in \{0, \dots, \ell\}$. There are three cases to consider:

**Case 1:** If $v''_j \in \mathit{Pref}(d, w) \setminus \mathit{Pref}(d, t)$, i.e., if $v''_j$ prefers $d$ to $w$, but not to $t$, then we have:

$$f_{w,t}(j, x, y) = \min \left\{ \begin{array}{l} f_{w,t}(j-1, x, y), \\ f_{w,t}(j-1, x-1, y) + \mathit{Pr}(w) \end{array} \right\}.$$

To see the correctness of the formula, note that to ensure that $d$ moves $x$ times below $w$ and $y$ times below $t$ for the voters $v''_1, \dots, v''_j$, we either ensure that this happens already for the voters $v''_1, \dots, v''_{j-1}$ and leave $v''_j$ intact, or we ensure that $d$ moves $x - 1$ times below $w$ and $y$ times below $t$ for voters $v''_1, \dots, v''_{j-1}$, and moves once below $w$ in the the preference order of $v''_j$ (from now on we will omit such detailed description, but the general idea for each of the cases is the same).

**Case 2:** If $v''_j$ is in $\mathit{Pref}(d, t) \setminus \mathit{Pref}(d, w)$, then we have:

$$f_{w,t}(j, x, y) = \min \left\{ \begin{array}{l} f_{w,t}(j-1, x, y), \\ f_{w,t}(j-1, x, y-1) + \mathit{Pr}(t) \end{array} \right\}.$$

**Case 3:** If $v''_j$ is in $\mathit{Pref}(d, w) \cap \mathit{Pref}(d, t)$ and $v''_j$ prefers $d$ to $w$ and $w$ to $t$, then we have:

$$f_{w,t}(j, x, y) = \min \left\{ \begin{array}{l} f_{w,t}(j-1, x, y), \\ f_{w,t}(j-1, x-1, y) + \mathit{Pr}(w), \\ f_{w,t}(j-1, x-1, y-1) + \mathit{Pr}(t) \end{array} \right\}.$$

On the other hand, if $v''_j$ prefers $d$ to $t$ and $t$ to $w$ then we have:

$$f_{w,t}(j, x, y) = \min \left\{ \begin{array}{l} f_{w,t}(j-1, x, y), \\ f_{w,t}(j-1, x, y-1) + \mathit{Pr}(t), \\ f_{w,t}(j-1, x-1, y-1) + \mathit{Pr}(w) \end{array} \right\}.$$

Given this discussion, it is clear that using standard dynamic-programming approach it is possible to compute function $f_{w,t}$ in polynomial time.

Finally, using function $f_{w,t}$, we can compute the cost of the cheapest tight solution for candidates $w$ and $t$. It suffices to try all values $x, y \in \{0, \dots, \ell\}$. For each such pair it is immediate to compute the score that candidates $d$ and $w$ would have after $d$ shifted $x$ times below $w$ and $y$ times below $t$. We would accept if the score of $w$ were at least as high as that of $d$ and we had $f_{w,t}(\ell, x, y) \leq B$. We reject if there is no choice of $w$, $t$, $x$, and $y$ that leads to acceptance. (Technically, we should also consider the case where $w = t$. It is straightforward to adapt the description of function $f_{w,t}$ to this case.) This concludes our proof. ☐

It is interesting to ask what feature of the Maximin rule—as opposed to the Copeland rule—lead to the fact that DESTRUCTIVE SHIFT BRIBERY is polynomial-time solvable. We believe that the reason is that it is safe to focus on a small number of candidates (the candidates $w$ and $t$). For Copeland elections, on the other hand, one has to keep track of all the candidates that the despised candidate passes, because each such pass could, in effect, decrease the despised candidate's score. (Interestingly, the same is true for the Borda rule—each time the despised candidate passes a candidate, the despised candidate's score decreases. However, in the case of Borda this process is unconditional, whereas in the case of Copeland's rule, the decrease may or may not happen, depending on shifts in other preference orders).

| Election rule | Constructive Shift Bribery | Destructive Shift Bribery |
|---|---|---|
| Plurality | P | P |
| $k$-Approval | P | P |
| Borda | NP-com. | P |
| Maximin | NP-com. | P |
| Copeland$^\alpha$ | NP-com. | NP-com. |

**Table 3: The complexity of Shift Bribery for various election rules. The results for the constructive case are due to Elkind et al. [14] and the results regarding the destructive case are due to this paper.**

## 4. CONCLUSIONS

We have introduced and studied a destructive variant of the Shift Bribery problem. In our problem, we ask if it is possible to preclude a given candidate from being the winner of an election by shifting this candidate backward in some of the votes (at a given cost, within a given budget), whereas in the constructive variant of the problem one asks if it is possible to ensure a given candidate's victory by shifting him or her forward. Thus, our approach to defining Destructive Shift Bribery is somewhat nonstandard: Not only we change the goal (from the constructive to the destructive one), but also we modify the set of available actions. We do so, because otherwise our problem would be meaningful only for non-monotone rules and would be quite unnatural (increasing the support for a given candidate to preclude him or her from winning, while possible and indeed occurring in real life, e.g., in STV elections, is quite a tricky, and perhaps controversial, issue).

We have shown that Destructive Shift Bribery is polynomial-time solvable for the $k$-Approval family of rules (in effect, including the Plurality rule), the Borda rule, all scoring protocols (as long as either the protocol or the price functions can be assumed to be encoded in unary), and the Maximin rule. On the other hand, we have shown that for each rational value of $\alpha$, the problem is NP-complete for Copeland$^\alpha$ (and we have drawn initial conclusions regarding the problem's parameterized complexity in this case). We summarize our results in Table 3.

Our work leads to several open questions. First, one could always study more elections rules. Second, at a technical level, it is tempting to study the parameterized complexity of our problem for Copeland in more depth. Third, it would be interesting to perform an empirical test to measure, for example, by how much we need to shift back the election winner to change the result under various assumptions regarding the voters' preference orders (and in real-life elections, such as those collected in PrefLib [24]).

## Acknowledgments

We are grateful to AAMAS reviewers for their helpful comments. The authors were supported by the AGH University grant 11.11.230.124 (statutory research).

## REFERENCES

[1] D. Baumeister, G. Erdélyi, and J. Rothe. How hard is it to bribe judges? A study of the complexity of bribery in judgement aggregation. In *Proceedings of the 2nd International Conference on Algorithmic Decision Theory*, pages 1–15. Springer-Verlag *Lecture Notes in Computer Science #6992*, Oct. 2011.

[2] D. Baumeister, P. Faliszewski, J. Lang, and J. Rothe. Campaigns for lazy voters: Truncated ballots. In *Proceedings of AAMAS-12*, pages 577–584. International Foundation for Autonomous Agents and Multiagent Systems, June 2012.

[3] D. Binkele-Raible, G. Erdélyi, H. Fernau, J. Goldsmith, N. Mattei, and J. Rothe. The complexity of probabilistic lobbying. *Discrete Optimization*, 11:1–21, 2014.

[4] R. Bredereck, J. Chen, P. Faliszewski, A. Nichterlein, and R. Niedermeier. Prices matter for the parameterized complexity of shift bribery. In *Proceedings of AAAI-14*, pages 1398–1404. AAAI Press, 2014.

[5] R. Bredereck, P. Faliszewski, R. Niedermeier, and N. Talmon. Large-scale election campaigns: Combinatorial shift bribery. In *Proceedings of AAMAS-15*, pages 67–75, 2015.

[6] D. Cary. Estimating the margin of victory for instant-runoff voting. Presented at the 2011 Electronic Voting Technology Workshop/Workshop on Trustworthy Elections, 2011.

[7] R. Christian, M. Fellows, F. Rosamond, and A. Slinko. On complexity of lobbying in multiple referenda. *Review of Economic Design*, 11(3):217–224, 2007.

[8] V. Conitzer, T. Sandholm, and J. Lang. When are elections with few candidates hard to manipulate? *Journal of the ACM*, 54(3):Article 14, 2007.

[9] M. Cygan, F. Fomin, Ł. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Sprigner, 2015.

[10] P. Dey and Y. Narahari. Estimating the margin of victory of an election using sampling. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence*, pages 1120–1126, July 2015.

[11] B. Dorn and D. Krüger. On the hardness of bribery variants in voting with CP-nets. *Annals of Mathematics and Artificial Intelligence*, To appear.

[12] B. Dorn and I. Schlotter. Multivariate complexity analysis of swap bribery. *Algorithmica*, 64(1):126–151, 2012.

[13] E. Elkind and P. Faliszewski. Approximation algorithms for campaign management. In *Proceedings of the 6th International Workshop On Internet And Network Economics*, pages 473–482. Springer-Verlag *Lecture Notes in Computer Science #6484*, Dec. 2010.

[14] E. Elkind, P. Faliszewski, and A. Slinko. Swap bribery. In *Proceedings of the 2nd International Symposium on Algorithmic Game Theory*, pages 299–310. Springer-Verlag *Lecture Notes in Computer Science #5814*, Oct. 2009.

[15] P. Faliszewski, E. Hemaspaandra, and L. Hemaspaandra. How hard is bribery in elections? *Journal of Artificial Intelligence Research*, 35:485–532, 2009.

[16] P. Faliszewski, E. Hemaspaandra, L. Hemaspaandra, and J. Rothe. Llull and Copeland voting computationally resist bribery and constructive

control. *Journal of Artificial Intelligence Research*, 35:275–341, 2009.

[17] P. Faliszewski, Y. Reisch, J. Rothe, and L. Schend. Complexity of manipulation, bribery, and campaign management in Bucklin and fallback voting. *Autonomous Agents and Multiagent Systems*, 29(6):1091–1124, 2015.

[18] P. Faliszewski and J. Rothe. Control and bribery in voting. In F. Brandt, V. Conitzer, U. Endriss, J. Lang, and A. D. Procaccia, editors, *Handbook of Computational Social Choice*, chapter 7. Cambridge University Press, To appear.

[19] N. Hazon, R. Lin, and S. Kraus. How to change a group's collective decision? In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence*, pages 198–205. AAAI Press, 2013.

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

[21] T. Magrino, R. Rivest, E. Shen, and D. Wagner. Computing the margin of victory in IRV elections. Presented at 2011 Electronic Voting Technology Workshop/Workshop on Trushworthy Elections, Aug. 2011.

[22] A. Maran, N. Maudet, M. Pini, F. Rossi, and K. Venable. A framework for aggregating influenced cp-nets and its resistance to bribery. In *Proceedings of AAAI-13*, pages 668–674. AAAI Press, 2013.

[23] N. Mattei, M. Pini, F. Rossi, and K. Venable. Bribery in voting over combinatorial domains is easy. In *Proceedings of AAMAS-12*, pages 1407–1408. International Foundation for Autonomous Agents and Multiagent Systems, June 2012.

[24] N. Mattei and T. Walsh. Preflib: A library for preferences. In *Proceedings of the 3nd International Conference on Algorithmic Decision Theory*, pages 259–270, 2013.

[25] I. Nehama. Complexity of optimal lobbying in threshold aggregation. In *Proceedings of the 4th International Conference on Algorithmic Decision Theory*, pages 379–395. Springer-Verlag *Lecture Notes in Computer Science #9346*, Sept. 2015.

[26] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006.

[27] S. Obraztsova and E. Elkind. On the complexity of voting manipulation under randomized tie-breaking. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, pages 319–324, July 2011.

[28] S. Obraztsova, E. Elkind, and N. Hazon. Ties matter: Complexity of voting manipulation revisited. In *Proceedings of AAMAS-11*, pages 71–78, May 2011.

[29] C. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[30] I. Schlotter, P. Faliszewski, and E. Elkind. Campaign management under approval-driven voting rules. *Algorithmica*, To appear.

[31] D. Shiryaev, L. Yu, and E. Elkind. On elections with robust winners. In *Proceedings of AAMAS-13*, pages 415–422. International Foundation for Autonomous Agents and Multiagent Systems, 2013.

[32] L. Xia. Computing the margin of victory for various voting rules. In *Proceedings of the 13th ACM Conference on Electronic Commerce*, pages 982–999. ACM Press, June 2012.