Campaign Management under Approval-Driven Voting Rules

Ildikó Schlotter
Department of Computer Science and Information Theory
Budapest University of Technology and Economics, Hungary

Piotr Faliszewski
Department of Computer Science
AGH Univ. of Science and Technology
Poland

Edith Elkind
School of Physical and Mathematical Sciences
Nanyang Technological University
Singapore

Abstract

Approval-like voting rules, such as Sincere-Strategy Preference-Based Approval voting (SP-AV), the Bucklin rule (an adaptive variant of k-Approval voting), and the Fallback rule (an adaptive variant of SP-AV) have many desirable properties: for example, they are easy to understand and encourage the candidates to choose electoral platforms that have a broad appeal. In this paper, we investigate both classic and parameterized computational complexity of electoral campaign management under such rules. We focus on two methods that can be used to promote a given candidate: asking voters to move this candidate upwards in their preference order or asking them to change the number of candidates they approve of. We show that finding an optimal campaign management strategy of the first type is easy for both Bucklin and Fallback. In contrast, the second method is computationally hard even if the degree to which we need to affect the votes is small. Nevertheless, we identify a large class of scenarios that admit a fixed-parameter tractable algorithm.

Introduction

Approval voting—a voting rule that asks each voter to report which candidates she approves of and outputs the candidates with the largest number of approvals—is one of the very few election systems that have a real chance of replacing Plurality voting in political elections. Some professional organizations, such as, e.g., the Mathematical Association of America or IEEE, already employ Approval voting, and recently New Hampshire state representatives sponsored a bill that replaces first-past-the-post voting with Approval voting.[1] Irrespective of the success of this initiative, it is a clear indication that Approval voting is attracting the attention of political decision-makers. One of the reasons for this is that, in contrast to the more standard Plurality voting, under Approval voting the candidates can benefit from running their campaigns in a consensus-building fashion, i.e., by choosing a platform that appeals to a large number of voters.

Nonetheless, Approval voting has certain disadvantages as well. Perhaps the most significant of them is its limited expressivity. Indeed, even a voter that approves of several candidates may like some of them more than others; however, Approval voting does not allow her to express this. Therefore, it is desirable to have a voting rule that operates similarly to Approval, yet takes voters' preference orders into account.

Several such voting rules have been proposed. For instance, the Bucklin rule (also known as the majoritarian compromise) asks the voters to gradually increase the number of candidates they approve of, until some candidate is approved by a majority of the voters. The winners are the candidates that receive the largest number of approvals at this point. In a simplified version of this rule, which is popular in the computational social choice literature (Xia et al. 2009; Xia and Conitzer 2008; Elkind, Faliszewski, and Slinko 2010), the winners are all candidates that are approved by a majority of the voters in the last round. Under both variants of the Bucklin rule, the common approval threshold is lowered gradually, thus reflecting the voters' preferences. However, this common threshold may move past an individual voter's personal approval threshold, forcing this voter to grant approval to a candidate that she does not approve of. To alleviate this problem, Brams and Sanver (2009) have recently introduced a new election system, which they call Fallback voting. This system works similarly to the Bucklin rule, but allows each voter to only approve of a limited number of candidates; its simplified version can be defined similarly to the simplified Bucklin voting.

With variants of Approval voting gaining wider acceptance, it becomes important to understand whether various activities associated with running an approval-based electoral campaign are computationally tractable. Such activities can be roughly classified into benign, such as winner determination, and malicious, such as manipulation and control; an ideal voting rule admits polynomial-time algorithms for the benign activities, but not for the malicious ones. However, there is an election-related activity that defies such classification, namely, bribery, or campaign management (Faliszewski, Hemaspaandra, and Hemaspaandra 2009; Elkind, Faliszewski, and Slinko 2009; Elkind and Faliszewski 2010). Both of these terms are used for actions that aim to make a given candidate an election winner by means of spending money on individual voters so as to change their preference rankings; these actions can be benign if the money is spent on legitimate activities, such as advertising, or malicious, if the voters are paid to vote non-truthfully.

[1] See http://freekeene.com/2011/01/28/will-nh-adopt-approval-voting/.

Now, winner determination for all approval-based rules listed above is clearly easy, and the complexity of manipulation and especially control under such rules is well understood (Baumeister et al. 2010; Erdélyi and Rothe 2010; Erdélyi and Fellows 2010; Erdélyi, Piras, and Rothe 2011). Thus, in this paper we focus on algorithmic aspects of electoral campaign management. Following (Elkind, Faliszewski, and Slinko 2009; Elkind and Faliszewski 2010) (see also (Dorn and Schlotter 2010)) who study this problem for a variety of preference-based voting rules, we model the campaign management setting using the framework of *shift bribery*. Under this framework, each voter v is associated with a cost function π, which indicates, for each $k > 0$, how much it would cost to convince v to promote the target candidate p by k positions in his vote. The briber (campaign manager) wants to make p a winner by spending as little as possible. This framework can be used to model a wide variety of campaign management activities, ranging from one-on-one meetings to phon-a-thons to direct mailing, each of which has a per-voter cost that may vary from one voter to another.

Note, however, that in the context of approval-based voting rules, we can campaign in favor of a candidate p even without changing the preference order of any voter. Specifically, if some voter v ranks p in position k and currently approves of $k-1$ candidates, we can try to convince v to lower her approval threshold so that she approves of p as well. Similarly, we can try to convince a voter to be more stringent and withdraw her approval from her least preferred approved candidate; this may be useful if that candidate is p's direct competitor. Arguably, a voter may be more willing to change her approval threshold than to alter her ranking of the candidates. Therefore, such campaign management tactics may be within the campaign manager's budget, even when she cannot afford the more direct approach discussed in the previous paragraph. We will refer to this campaign management technique as "support bribery"; a variant of this model has been considered by Elkind, Faliszewski, and Slinko (2009).

In this paper, we investigate the algorithmic aspects of both campaign management activities discussed above, i.e., shift bribery and support bribery. We consider five approval-based voting rules, namely, SP-AV (as formalized by Brams and Sanver (2006)), Bucklin (both classic and simplified), and Fallback (both classic and simplified). We show that shift bribery is easy with respect to both variants of the Bucklin rule, as well as both variants of the Fallback rule. The argument for the simplified version of both rules relies on dynamic programming, while for the classic version of these rules we use a more involved flow-based approach. In contrast, support bribery tends to be hard; this holds even if we parameterize this problem by the number of voters to be bribed or the total change in the approval counts, and use very simple bribery cost functions. Nevertheless, we identify a natural class of bribery cost functions for which support bribery is fixed-parameter tractable with respect to the latter parameter.

The rest of this paper is organized as follows. In the next section we formally define our model of elections, the voting systems we study, and provide the necessary background on (parameterized) computational complexity. We then present our algorithms for shift bribery, followed by complexity results and a fixed-parameter tractable algorithm for support bribery. We conclude the paper by presenting directions for future research. We omit most proofs due to page limit.

Preliminaries

An *election* is a pair $E = (C, V)$, where $C = \{c_1, \ldots, c_m\}$ is the set of *candidates* and $V = (v^1, \ldots, v^n)$ is the list of *voters*. Each voter v^i is associated with a *preference order* $\succ^i$, which is a total order over C, and an *approval count* $\ell^i \in [0, |C|]$; voter v^i is said to *approve* of the top ℓ^i candidates in her preference order. We denote by $rank(c, v)$ the position of candidate c in the preference order of voter v: v's most preferred candidate has rank 1 and her least preferred candidate has rank $|C|$. A *voting rule* is a mapping that given an election $E = (C, V)$ outputs a set $W \subseteq C$ of *election winners*.

Voting rules Most voting rules commonly considered in the literature do not make use of the approval counts. For instance, under *k-Approval* each candidate gets one point from each voter that ranks her in top k positions. The k-Approval score $s_k(c)$ of a candidate $c \in C$ is the total number of points that she gets, and the winners are the candidates with the highest score. The *Bucklin rule*, which can be thought of as an adaptive version of k-Approval, is defined as follows. Given a candidate $c \in C$, let $s_B(c)$ denote the smallest value of k such that at least $\lfloor \frac{n}{2} \rfloor + 1$ voters rank c in the top k positions, where n is the number of voters; we say that c *wins in round* $s_B(c)$. The quantity $k_B = \min_{c \in C} s_B(c)$ is called the *Bucklin winning round*. Observe that no candidate wins in any of the rounds $\ell < k_B$ and at least one candidate wins in round k_B. The Bucklin winners are the candidates with the highest k_B-Approval score. Under the simplified Bucklin rule, the winners are the candidates whose k_B-Approval score is at least $\lfloor \frac{n}{2} \rfloor + 1$; all Bucklin winners are simplified Bucklin winners, but the converse is not necessarily true.

We observe that k-Approval, despite its name, ignores the approval counts entirely: a candidate c may fail to get a point from a voter v^i who approves of her (if $\ell^i \geq rank(c, v^i) > k$), or obtain a point from a voter v^j who does not approve of her (if $\ell^j < rank(c, v^j) \leq k$). Similarly, neither version of the Bucklin rule uses the information provided by the approval counts. In contrast, the SP-AV rule (Brams and Sanver 2006) relies heavily on the approval counts: each candidate gets one point from each voter that approves of her, and the winners are the candidates with the highest number of points. Finally, *Fallback voting* (Brams and Sanver 2009) makes use of both the preference orders and the approval counts. Specifically, under this rule we apply the Bucklin rule to the election obtained by deleting each voter's non-approved candidates from her preference ranking. Since the preference orders are truncated, it may happen that no candidate is ranked by more than half of the voters, in which case the candidates approved by the largest number of voters are elected. We can replace the Bucklin rule with the simplified Bucklin rule in this construction; we will refer to the result-

ing rule as the *simplified Fallback rule*.

Parameterized complexity The framework of parameterized complexity deals with computationally hard problems. In a parameterized problem, each input instance I comes together with an integer k called the *parameter*, and the aim is to design algorithms that are efficient if the value of the parameter is small. Formally, a problem is said to be *fixed-parameter tractable (FPT)* with respect to parameter k if it admits an algorithm whose running time on an input (I, k) is $f(k)|I|^{O(1)}$ for some computable function f; note that the exponent of $|I|$ does not depend on k. Though f is typically an exponential function, such an algorithm is usually more efficient than one running in time $O(|I|^k)$.

To capture problems that are not fixed-parameter tractable, researchers typically use the W-hierarchy, of which the first two levels are W[1] and W[2] (P $\subseteq$ FPT $\subseteq$ W[1] $\subseteq$ W[2] $\subseteq$ ...). Intuitively, W[1] is a parameterized analog of NP. W[1]-hardness and W[2]-hardness are defined in a standard way, on the basis of parameterized reductions.

W[1]-hardness (or, worse yet, W[2]-hardness) yields strong evidence that we cannot expect an FPT algorithm for the problem with the given parameterization. For a more extensive treatment of parameterized complexity, we refer the reader to e.g., (Downey and Fellows 1999).

Campaign Management The following definition is adapted from (Elkind and Faliszewski 2010), which itself is based on the one in (Elkind, Faliszewski, and Slinko 2009).

Definition 1. Let $\mathcal{R}$ be a voting rule. An instance of $\mathcal{R}$-SHIFT BRIBERY problem is a tuple $I = (C, V, \Pi, p)$, where $C = \{p, c_1, \ldots, c_{m-1}\}$, $V = (v^1, \ldots, v^n)$ is a list of voters together with their preference orders over C (and approval counts, if $\mathcal{R}$ uses them), $\Pi = (\pi^1, \ldots, \pi^n)$ is a family of cost functions, where each π^i is a non-decreasing function from $[0, |C|]$ to $\mathbb{Z}^+ \cup \{+\infty\}$ that satisfies $\pi^i(0) = 0$, and $p \in C$ is a designated candidate.[2] The goal is to find a minimal value b for which there is a vector $\mathbf{t} = (t_1, \ldots, t_n) \in (\mathbb{Z}^+)^n$ such that (a) $b = \sum_{i=1}^{n} \pi^i(t_i)$, and (b) if for each $i = 1, \ldots, n$ we shift p upwards in the i-th vote by t_i positions, then p becomes an $\mathcal{R}$-winner of E. We denote this value of b by $\mathrm{opt}(I)$.

In words, $\pi^i(k)$ is the cost of shifting the preferred candidate p forward by k positions in the preferences of the i-th voter. We will refer to the vector $\mathbf{t} = (t_1, \ldots, t_n)$ as a *shift action*, and denote by $shf(C, V, \mathbf{t})$ the election obtained from (C, V) by shifting p forward by t_i positions in each vote. Also, we write $\Pi(\mathbf{t}) = \sum_{i=1}^{n} \pi^i(t_i)$. If $rank(p, v_i) = k$, but a shift action prescribes shifting p by $k' > k$ positions in v_i's ranking, we simply place p on top of the vote.

Shift bribery does not change the voters' approval counts. A more general notion of bribery, which is relevant for SP-AV and (simplified) Fallback voting, was proposed by Elkind, Faliszewski, and Slinko (2009) in the technical report version of their paper. Specifically, they defined *mixed bribery* for SP-AV, where the briber can both shift the preferred candidate and change the voters' approval counts. In this work, we find it more convenient to separate these two types of bribery. Thus, we will now define *support bribery*, which focuses on changing the number of approved candidates.

First, we need to introduce another family of cost functions, which provide information about the costs of increasing/decreasing the number of candidates approved by each voter. Specifically, we assume that each voter v^i also has a *support bribery cost function* $\sigma^i \colon \mathbb{Z} \to \mathbb{Z}^+ \cup \{+\infty\}$, which satisfies (a) $\sigma^i(0) = 0$ (b) for each $k > 0$, $\sigma^i(k) \leq \sigma^i(k+1)$ and $\sigma^i(-k) \leq \sigma^i(-k-1)$. For a given $k \in \mathbb{Z}$, we interpret $\sigma^i(k)$ as the cost of convincing v^i to approve of $\ell^i + k$ candidates. Clearly, it suffices to define σ^i on $[-\ell^i, |C| - \ell^i]$, where ℓ^i is the approval count of v^i. We are now ready to define the support bribery problem.

Definition 2. Let $\mathcal{R}$ be a voting rule. An instance of $\mathcal{R}$-SUPPORT BRIBERY problem is a tuple $I = (C, V, \Sigma, p)$, where $C = \{p, c_1, \ldots, c_{m-1}\}$ is a set of candidates, $V = (v^1, \ldots, v^n)$ is a list of voters, where each voter v^i is represented by her preference order $\succ^i$ and her approval count ℓ^i, and $\Sigma = (\sigma^1, \ldots, \sigma^n)$ is a family of support bribery cost functions (each represented by listing its values for all appropriate arguments). The goal is to find a minimal value b such that there is a vector $\mathbf{t} = (t_1, \ldots, t_n) \in \mathbb{Z}^n$ with the following properties: (a) $b = \sum_{i=1}^{n} \sigma^i(t_i)$, and (b) if for each $i = 1, \ldots, n$ voter v^i changes her approval count from ℓ^i to $\ell^i + t_i$, then p is an $\mathcal{R}$-winner of the resulting election.

When discussing NP-completeness, we consider a decision version of this problem, where we ask if there exists a bribery whose cost does not exceed a given value b.

There are two interesting special cases of support bribery that can be derived from the general model by setting the bribery costs so that decreasing/increasing the approval counts is prohibitively expensive. Specifically, we will say that a support bribery cost function σ is *positive* if $\sigma(k) = +\infty$ for any $k < 0$ and *negative* if $\sigma(k) = +\infty$ for any $k > 0$. The support bribery with positive cost functions corresponds to the setting where the campaign manager can only increase the voter's approval counts, and can be viewed as a fine-grained version of control by adding voters; similarly, the support bribery with negative cost functions can be viewed as a refinement of control by deleting voters.

Note also that, just as in the case of control problems, we can consider destructive support bribery, where the goal is not to ensure that the preferred candidate p wins the election, but rather that some despised candidate d does not. In the context of control, this problem was studied by Hemaspaandra, Hemaspaandra, and Rothe (2007).

Shift Bribery

In this section, we present our results for SHIFT BRIBERY under the Bucklin rule and the Fallback rule. We start by describing our algorithm for the simplified version of the Bucklin rule; this algorithm can be modified to work for the simplified version of the Fallback rule.

Theorem 3. *Simplified Bucklin-*SHIFT BRIBERY *is in* P.

[2] Each of our cost functions π^i is specified by providing its values $\pi^i(0), \pi^i(1), \ldots, \pi^i(|C|)$.

Proof. Given an instance $I = (C, V, \Pi, p)$ of Simplified Bucklin-SHIFT BRIBERY, let $m = |C|$, $n = |V|$, and let k be the Bucklin winning round for (C, V). Let $W \subseteq C \setminus \{p\}$ be the set of the simplified Bucklin winners in (C, V).

Let $\mathbf{t} = (t_1, \ldots, t_n)$ be a minimal optimal shift action for I, i.e., $\Pi(\mathbf{t}) = \mathrm{opt}(I)$, p is a winner in $shf(C, V, \mathbf{t})$, but p is not a winner in $shf(C, V, \mathbf{s})$ for any $\mathbf{s} \neq \mathbf{t}$ with $s_i \leq t_i$ for all $i = 1, \ldots, n$ (note that an optimal shift action is not necessarily minimal, as it may include some shifts of cost 0 that are not needed to make p a winner).

Let ℓ be the Bucklin winning round in $shf(C, V, \mathbf{t})$. We have $\ell \in \{k, k+1\}$. Indeed, any shift action moves any candidate in W by at most one position downwards. Therefore, in $shf(C, V, \mathbf{t})$ all candidates in W win in round $k+1$, and hence $\ell \leq k+1$. Now, suppose that $\ell < k$. In (C, V) the $(k-1)$-Approval score of any candidate is at most $\lfloor \frac{n}{2} \rfloor$, so the only candidate that can win in round $\ell < k$ in $shf(C, V, \mathbf{t})$ is p, and for that she has to be moved into position ℓ in at least some voters' preferences. However, moving p into position k in those voters' preferences suffices to make p a winner in round k (and thus an election winner), and we have assumed that $\mathbf{t}$ is minimal. This contradiction shows that $\ell \geq k$. Hence, to find an optimal shift bribery, it suffices to compute the cheapest shift action that makes p a winner in round k, as well as the cheapest shift action that makes p a winner in round $k+1$ and ensures that no other candidate wins in round k, and output the cheaper of the two.

To win in round k, p needs to obtain $\lfloor \frac{n}{2} \rfloor + 1 - s_k(p)$ additional k-Approval points. Thus, to find the cheapest shift bribery that makes p win in round k, we consider all votes in which p is not ranked in the top k positions, order them by the cost of moving p into the k-th position (from lowest to highest), and pick the first $\lfloor \frac{n}{2} \rfloor + 1 - s_k(p)$ of these votes. Let $\mathbf{s}$ denote the shift action that moves p into position k in each of those votes.

Computing a shift action that ensures p's victory in the $(k+1)$-st round is somewhat more difficult. In this case we need to ensure that (a) each candidate in W is demoted from position k to position $k+1$ enough times not to win in round k and (b) p's $(k+1)$-Approval score is at least $\lfloor \frac{n}{2} \rfloor$. Thus, we need to find an optimal balance between bribing several groups of voters.

For each $c \in C \setminus \{p\}$, let V_c denote the set of all voters that rank c in the k-th position and rank p below c; note that $c \neq c'$ implies $V_c \cap V_{c'} = \emptyset$. Let us fix a candidate c in $C \setminus \{p\}$. To ensure that c does not win in round k, we need to shift p into position k in at least $n(c) = \max(0, s_k(c) - \lfloor \frac{n}{2} \rfloor)$ votes in V_c. Note that $n(c) > 0$ if and only if $c \in W$. Thus, if for some $c \in W$ we have $|V_c| < n(c)$, there is no way to ensure that no candidate in C wins in round k, so in this case we output $\mathbf{s}$ and stop.

Otherwise, we proceed as follows. Let A_c be the set of all voters in V_c that rank p in position $k+1$, and let $B_c = V_c \setminus A_c$. Note that for each vote in A_c, shifting p into the k-th position does not change the $(k+1)$-Approval score of p, while doing the same for a vote in B_c increases the $(k+1)$-Approval score of p by one. For each $i = 0, \ldots, |B_c|$, let $b(c, i)$ be the minimum cost of a shift action that (a) shifts p into position $k+1$ or above in i votes from B_c, and (b) shifts p into position k in at least $n(c)$ votes from $A_c \cup B_c$. We can compute each $b(c, i)$ in polynomial time using dynamic programming. To do so, for each i and j, $0 \leq i \leq j \leq |B_c|$, and each $h = 0, \ldots, n(c)$, we define $b(c, i, j, h)$ to be the cost of a minimum-cost shift action that only involves the voters in A_c and the first j voters in B_c and that (a) shifts p into position $k+1$ or above in i votes from B_c, and (b) shifts p into position k in at least h votes from $A_c \cup B_c$. If there is no such shift action, we set $b(c, i, j, h) = +\infty$.

Clearly, $b(c, 0, j, h)$ can be computed by ordering the voters in A_c according to their cost of moving p into the k-th position (from lowest to highest), and then bribing the first h voters among them. We can similarly compute $b(c, i, j, 0)$, focusing on the first j voters in B_c and on shifting to position $k+1$. For all the remaining cases, we compute $b(c, i, j, h)$ using the following formula. Abusing notation, we write v^j to denote the j-th voter in B_c.

$$b(c, i, j, h) = \min \begin{cases} b(c, i-1, j-1, h) + C_1, \\ b(c, i-1, j-1, h-1) + C_2, \\ b(c, i, j-1, h). \end{cases} \quad (1)$$

where $C_1 = \pi^j(rank(p, v^j) - (k+1))$ and $C_2 = \pi^j(rank(p, v^j) - k)$.

The first and the second line of this formula correspond to the case where p is shifted into position $k+1$ and into position k, respectively, in the j-th vote of B_c. The third line deals with the case where p is not shifted in this vote. It is straightforward to verify that this method indeed computes the desired values. By definition, we have $b(c, i) = b(c, i, |B_c|, n(c))$. For each candidate $c \in C \setminus \{p\}$ and each $i = 0, \ldots, |B_c|$, we define $\mathbf{r}(c, i)$ to be the shift action corresponding to the value $b(c, i)$, read off the dynamic programming computation of $b(c, i)$ using standard techniques.

Observe that a shift action increases the $(k+1)$-Approval score of p by exactly the number of those votes in $\bigcup_{c \in C \setminus \{p\}} B_c$ where it moves p to position $k+1$ or above. Thus, implementing each shift action of the form

$$\sum_{c \in C \setminus \{p\}} \mathbf{r}(c, i_c), \quad (2)$$

where $H = \{i_c \mid c \in C \setminus \{p\}\}$ is a set of non-negative integers whose sum is at least $\lfloor \frac{n}{2} \rfloor + 1 - s_{k+1}(p)$, ensures that (a) p wins in round $k+1$, and (b) no other candidate wins in round k. Condition (a) is guaranteed by the requirement on the sum of H and, for each candidate $c \in C \setminus \{p\}$, condition (b) is guaranteed by the definition of $\mathbf{r}(c, i_c)$. In addition, it is not too hard to see that a minimum-cost shift action ensuring that conditions (a) and (b) are satisfied must be a minimum-cost shift action of the form (2).

Now, given shift actions $\mathbf{r}(c, i)$ for each $c \in C \setminus \{p\}$ and each $i = 0, \ldots, |B_c|$, we can compute a minimum-cost shift action $\mathbf{r}$ of the form (2), where $H = \{i_c \mid c \in C \setminus \{p\}\}$ is a set of non-negative integers whose sum is at least $\lfloor \frac{n}{2} \rfloor + 1 - s_{k+1}(p)$, using standard dynamic programming (e.g., by considering the candidates in $C \setminus \{p\}$ one by one).

We output the cheaper of $\mathbf{s}$ and $\mathbf{r}$. This algorithm clearly runs in polynomial time, and our argument shows that it produces an optimal shift action for I. □

A similar argument works for the simplified Fallback rule.

Theorem 4. *Simplified Fallback*-SHIFT BRIBERY *is in* P.

A harder proof resolves the issue for regular Bucklin.

Theorem 5. *Bucklin*-SHIFT BRIBERY *is in* P.

Briefly, the argument proceeds as follows. We observe that if k is the Bucklin winning round in the original instance, then after the bribery the Bucklin winning round k' satisfies $k' \in \{k-1, k, k+1\}$. We then find the optimal bribery for each of these values of k'. For $k' = k-1$, a simple greedy algorithm works. For $k' = k$, for each $i = 1, \ldots, n$ we find the cheapest shift action $\mathbf{r}^i$ that ensures that p's score is i, and the score of any other candidate is at most i; we save the best of these actions. For $k' = k+1$, we need to ensure that p's $(k+1)$-Approval score is sufficiently high, while both the k-Approval score and the $(k+1)$-Approval score of any other candidate is sufficiently low; these goals are interrelated. This case is handled by a network flow argument, where the optimal shift action corresponds to a min-cost flow in a certain carefully constructed network. A similar approach works for the Fallback rule.

Theorem 6. *Fallback*-SHIFT BRIBERY *is in* P.

Support Bribery

The technical report version of (Elkind, Faliszewski, and Slinko 2009) gives an NP-completeness result for mixed bribery under SP-AV. However, their proof does not rely on shifting the preferred candidate in the voters' preferences, and therefore applies to support bribery as well, showing that the decision version of SP-AV-SUPPORT BRIBERY is NP-complete. In this section we extend this result to Fallback voting, and explore the parameterized complexity of support bribery under both the simplified and the classic variant of this rule.

Any instance I of support bribery can be associated with the following parameters. First, let $\alpha(I)$ denote the maximum number of voters that are bribed in any bribery that solves I optimally. Second, let $\beta(I)$ and $\beta'(I)$ denote, respectively, the maximum and the minimum of $\sum_{i=1}^{n} |t_i|$ for any bribery $(t_1, \ldots, t_n)$ that solves I optimally; these parameters describe how much we have to modify the approval counts in total. Observe that $\beta(I) \geq \beta'(I)$ and $\beta(I) \geq \alpha(I)$ for every instance I.

We will now demonstrate that support bribery under Fallback is computationally hard, even in very special cases. These results, while somewhat disappointing from the campaign management perspective, are hardly surprising. Indeed, we have argued that support bribery can be viewed as a fine-grained version of control by adding/deleting voters, and both of these control problems are NP-hard for Fallback voting (Erdélyi and Rothe 2010). In fact, since Fallback defaults to approval voting if no candidate is approved by a majority of voters, by introducing sufficiently many dummy candidates we can easily reduce the problem of control by adding voters under approval to the problem of support bribery under Fallback voting.

Our next result shows that both variants of Fallback-SUPPORT BRIBERY are NP-hard under very strong restrictions on the cost function; moreover, these problems remain intractable even for instances with a small value of α. Thus, bribing even a few voters can be a hard task.

Theorem 7. *Both Fallback*-SUPPORT BRIBERY *and simplified Fallback*-SUPPORT BRIBERY *are* NP*-complete, and also W[2]-hard with parameter α, even in the special case where each cost is either $+\infty$ or 0, and either all cost functions are positive or all cost functions are negative.*

To prove Theorem 7, we consider positive cost functions and negative cost functions separately. In each case, we give a polynomial-time computable parameterized reduction from the W[2]-hard DOMINATING SET problem. These reductions are inspired by those given by Erdélyi and Fellows (2010) in their proof that control by adding/deleting voters under Fallback is W[2]-hard.

Since the hardness result for Fallback-SUPPORT BRIBERY holds even if all bribery costs are either 0 or $+\infty$, it follows that this problem does not admit an approximation algorithm with a bounded approximation ratio.

Now, Theorem 7 shows that Fallback-SUPPORT BRIBERY is W[2]-hard with respect to the parameter α. Given that we have $\beta(I) \geq \alpha(I)$ for any instance I, it is natural to ask whether Fallback-SUPPORT BRIBERY remains hard if even β is small, i.e., every optimal bribery only makes small changes to the approval counts. It turns out that this problem is still hard, even under the assumption of *unit costs*, i.e., $\sigma^i(k) = |k|$ for each k and each $i = 1, \ldots, n$.

Theorem 8. *Both Fallback*-SUPPORT BRIBERY *and simplified Fallback*-SUPPORT BRIBERY *are W[1]-hard with parameter β, even if $\sigma^i(k) = |k|$ for each k and each $i = 1, \ldots, n$.*

The proof proceeds by a parameterized reduction from the W[1]-hard MULTICOLORED CLIQUE problem (Fellows et al. 2009), and is omitted due to space constraints.

The hardness proof in Theorem 8 makes use of unit cost functions. In contrast, for positive or negative cost functions (simplified) Fallback-SUPPORT BRIBERY is fixed-parameter tractable with respect to β'.

Theorem 9. *Both Fallback*-SUPPORT BRIBERY *and simplified Fallback*-SUPPORT BRIBERY *are FPT with respect to β', as long as either all bribery cost functions are positive or all bribery cost functions are negative.*

In both cases, the algorithm starts by guessing the round where p wins, together with p's score in that round. The main idea in the negative case is to identify a small set of relevant candidates whose score must be decreased in order to prevent them from beating p, and then partition the votes into equivalence classes, according to their effect on the relevant candidates. As the number of equivalence classes can be bounded by a function of β', this approach leads to a bounded search tree algorithm running in FPT time.

When all cost functions are positive, the number of candidates who might beat p via gaining a few extra points can be large, hence applying a bounded search tree approach is not straightforward. To overcome this difficulty, we apply the technique of color-coding (Alon, Yuster, and Zwick 1995), where a random coloring of the candidates is used to guide us in choosing a set of voters that can be bribed safely. This

gives us a randomized FPT algorithm with one-sided error, producing a correct output with probability at least $2^{-\beta'^2}$, which can be derandomized by standard methods.

We remark that our hardness result applies to the larger parameter β, while our algorithm works for the smaller parameter β'. This is good news, since, in general, it is easier to design algorithms for larger parameters (and, conversely, prove hardness results for smaller parameters).

In contrast to our hardness results for constructive support bribery, we can show that destructive support bribery is easy for SP-AV, simplified Fallback voting, and Fallback voting.

Theorem 10. *Destructive support bribery is in* P *for each of SP-AV, simplified Fallback voting, and Fallback voting.*

Conclusions and Future Work

Our results show that shift bribery tends to be computationally easier than support bribery. However, in general, the power of these campaign management strategies is incomparable: one can construct examples of, e.g., Fallback elections where it is impossible to make someone a winner within a finite budget by shift bribery, but it is possible to do so by support bribery, or vice versa. Thus, both shift bribery and support bribery deserve to be studied in more detail.

Our algorithmic techniques highlight the difference between the Bucklin rule and its simplified version, and suggest that one should exercise caution when using the results for simplified Bucklin to derive conclusions for the classic Bucklin. Another contribution of this paper is a natural parameterization that leads to FPT algorithms for support bribery under two variants of the Fallback rule, for a large class of bribery cost functions. Finding other tractable parameterizations is an interesting direction for future research. Another way to circumvent the hardness results is to study the complexity of support bribery under restricted preferences. For instance, recent work (Faliszewski et al. 2011; Brandt et al. 2010) shows that many hard problems in computational social choice become easy if the voters' preferences can be assumed to be single-peaked; it would be interesting to determine if this is the case for support bribery.

Acknowledgements. We thank the AAAI reviewers for their comments. Ildikó Schlotter was supported by the Hungarian National Research Fund (grant OTKA 67651), and by the European Union and the European Social Fund (grant TÁMOP 4.2.1./B-09/1/KMR-2010-0003). Edith Elkind was supported by NRF (Singapore) under Research Fellowship NRF RF2009-08. Piotr Faliszewski was Supported by AGH University of Technology Grant no. 11.11.120.865, by Polish Ministry of Science and Higher Education grant N-N206-378637, and by Foundation for Polish Science's program Homing/Powroty.

References

Alon, N.; Yuster, R.; and Zwick, U. 1995. Color-coding. *J. ACM* 42(4):844–856.

Baumeister, D.; Erdélyi, G.; Hemaspaandra, E.; Hemaspaandra, L.; and Rothe, J. 2010. Computational aspects of approval voting. In Laslier, J., and Sanver, R., eds., *Handbook of Approval Voting*. Springer. 199–251.

Brams, S., and Sanver, R. 2006. Critical strategies under approval voting: Who gets ruled in and ruled out. *Electoral Studies* 25(2):287–305.

Brams, S., and Sanver, R. 2009. Voting systems that combine approval and preference. In *The Mathematics of Preference, Choice, and Order: Essays in Honor of Peter C. Fishburn*. Springer. 215–237.

Brandt, F.; Brill, M.; Hemaspaandra, E.; and Hemaspaandra, L. 2010. Bypassing combinatorial protections: Polynomial-time algorithms for single-peaked electorates. In *Proceedings of AAAI-10*, 715–722.

Dorn, B., and Schlotter, I. 2010. Multivariate complexity analysis of swap bribery. In *Proceedings of IPEC-10*, 107–122.

Downey, R., and Fellows, M. 1999. *Parameterized Complexity*. Springer-Verlag.

Elkind, E., and Faliszewski, P. 2010. Approximation algorithms for campaign management. In *Proceedings of WINE-10*, 473–482.

Elkind, E.; Faliszewski, P.; and Slinko, A. 2009. Swap bribery. In *Proceedings of SAGT-09*, 299–310.

Elkind, E.; Faliszewski, P.; and Slinko, A. 2010. On the role of distances in defining voting rules. In *Proceedings of AAMAS-10*, 375–382.

Erdélyi, G., and Fellows, M. R. 2010. Parameterized control complexity in Bucklin voting and in Fallback voting. In *Proceedings of COMSOC-10*.

Erdélyi, G., and Rothe, J. 2010. Control complexity in fallback voting. In *CATS'10*, pp. 39–48.

Erdélyi, G.; Piras, L.; and Rothe, J. 2011. The complexity of voter partition in bucklin and fallback voting: Solving three open problems. In *Proceedings of AAMAS-11*. To appear.

Faliszewski, P.; Hemaspaandra, E.; Hemaspaandra, L.; and Rothe, J. 2011. The shield that never was: Societies with single-peaked preferences are more open to manipulation and control. *Information and Computation* 209(2):89–107.

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

Fellows, M. R.; Hermelin, D.; Rosamond, F. A.; and Vialette, S. 2009. On the parameterized complexity of multiple-interval graph problems. *Theoretical Computer Science* 410:53–61.

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

Xia, L., and Conitzer, V. 2008. Determining possible and necessary winners under common voting rules given partial orders. In *Proceedings of AAAI-08*, 196–201.

Xia, L.; Zuckerman, M.; Procaccia, A.; Conitzer, V.; and Rosenschein, J. 2009. Complexity of unweighted manipulation under some common voting rules. In *Proceedings of IJCAI-09*, 348–353.