Elections with Few Candidates: Prices, Weights, and Covering Problems

Abstract. We show that a number of election-related problems with prices (such as, for example, bribery) are fixed-parameter tractable (in FPT) when parameterized by the number of candidates. For bribery, this resolves a nearly 10-year old family of open problems. Our results follow by a general technique that formulates voting problems as covering problems and extends the classic approach of using integer linear programming and the algorithm of Lenstra [19]. In this context, our central result is that WEIGHTED SET MULTICOVER parameterized by the universe size is fixed-parameter tractable. Our approach is also applicable to weighted electoral control for Approval voting. We improve previously known XP-memberships to FPT-memberships. Our preliminary experiments on real-world-based data show the practical usefulness of our approach for instances with few candidates.

1 Introduction

We resolve the computational complexity status of a number of election problems parameterized by the number of candidates, for the case where voters are unweighted but have prices. These include, for example, various bribery problems [10, 12] and priced control problems [21] that were known to be in XP since nearly 10 years ago, but were neither known to be fixed-parameter tractable (in FPT), nor to be W[1]-hard. We develop a general technique for showing their membership in FPT, which also applies to weighted voter control for Approval voting, improving results of Faliszewski et al. [14]. We test the running times of our algorithms empirically.

Algorithmic problems that model manipulating elections include, among others, strategic voting problems [1, 6] (where we are given an election with honest voters and we ask whether a group of manipulators can cast votes to ensure their preferred candidate's victory), election control problems [2, 17] (where we are given an election and ask if we can ensure a given candidate's victory by adding/deleting candidates/voters), or bribery [10, 12, 24] and campaign management problems [3, 5, 8, 23] (where we want to ensure a given candidate's victory by changing some of the votes, but where each vote change comes at a price and we are bound by a budget). We focus on the case where we have a few candidates but (possibly) many voters. As pointed out by Conitzer et al. [6], this is a very natural setting and it models many real-life scenarios such as political elections or elections among company stockholders.

The complexity of manipulating elections with few candidates is, by now, very well understood. On the one hand, if the elections are weighted (as is the case for the elections held by company stockholders), then our problems are typically NP-hard even if the number of candidates is a small fixed constant [6, 12, 14]; these results typically follow by reductions from the well-known NP-hard problem PARTITION. One particular example where we do not have NP-hardness is control by adding/deleting voters

under the Approval and k-Approval voting rules. Faliszewski at al. [14] have shown that these problems are in XP, that is, that they can be solved in polynomial time if the number of candidates is assumed to be a constant. On the other hand, if the elections are unweighted (as is the case for political elections) and no prices are involved, then we typically get FPT results. These results are often obtained by expressing the respective problems as integer linear programs (ILPs) and by applying the famous algorithm of Lenstra [19]. For example, for control by adding voters we can have a program with a separate integer variable for each possible preference, counting how many voters with each preference we add [13] (the constraints ensure that we do not add more voters with a given preference than are available and that the desired candidate becomes a winner). Since the number of different preferences is a function depending only on the number of candidates, we can solve such an ILP using Lenstra's algorithm in FPT time (Lenstra's algorithm solves ILPs in FPT time with respect to the number of integer variables). Typically, this approach does not work for weighted elections as weights give voters a form of "identity." In the control example, it is no longer suffices to specify how many voters to add; we need to know exactly which ones to add (the trick in showing XP-membership for weighted voter control under Approval is to see that for each possible voter preference, we add only the heaviest voters with this preference [14]).

The main missing piece in our understanding of the complexity of manipulating elections with few candidates regards those unweighted-election problems where each voter has some sort of price (for example, as in the bribery problems). In this paper we almost completely fill this gap by showing a general approach for proving FPT membership for a class of bribery-like problems parameterized by the number of candidates, for unweighted elections[1] (as a side effect, we also get FPT membership for weighted control under the Approval and k-Approval rules). The main idea of our solution is to use mixed integer linear programming (MILP) formulations of our problems, divided into two parts. The first part is the same as in the standard ILP solutions for the non-priced variants of our problems, modeling how many voters with each possible preferences are "affected" ("bought" or "convinced" for bribery and campaign managements, "added" or "deleted" for control problems). The second part, and a contribution of this paper, uses non-integer variables to compute the cost of the solution from the first part. The critical insight of our approach is to use the fact that we "affect" the voters in the order of their increasing prices to force all our variables to be integer-valued in the optimal solutions. This way we can compute the cost of each of our solutions using rational-valued variables and solve our MILPs using Lenstra's algorithm (it maintains its FPT running time for MILPs parameterized by the number of integer variables).

Unfortunately, while Lenstra's algorithm is a very powerful tool for proving FPT membership, it might be too slow in practice. Further, as pointed out by Bredereck et al. [4], each time an FPT result is achieved through an application of Lenstra's result, it is natural to ask whether one can derive the same result through a direct, combinatorial algorithm. Coming up with such a direct algorithm seems very difficult. Thus,

[1] One problem for which our technique does not apply is SWAP BRIBERY [10]; even though Dorn and Schlotter [8] claim that it is in FPT when parameterized by the number of candidates, their proof applies only to a restricted setting. The complexity of SWAP BRIBERY parameterized by the number of candidates remains open.

instead, for our problems we show a combinatorial algorithm obtaining solutions arbitrarily close to the optimal ones in FPT time (formally, we show an FPT approximation scheme). Nonetheless, in practice, one would probably not use Lenstra's algorithm for solving MILPs, but instead, one would use an off-the-shelf optimized heuristic. We provide a preliminary empirical comparison of the running times of the MILP-based algorithm (using an off-the-shelf MILP solver instead of Lenstra's algorithm) and an ILP-based algorithm that reduces our problems directly to integer linear programming (basically without "exploiting" the parameter number of candidates). Our results suggest that FPT algorithms based on solving MILPs can be very efficient in practice.

Our results can be applied to a large class of voting rules and to many election problems. Thus, to better illustrate technical details, we focus on the simplest setting possible. Specifically, we present most of our techniques through a family of classic covering problems. The motivation is threefold: (a) this focus allows us to present our results most clearly, (b) our variants of the covering problem apply directly to a number of election problems for the Approval rule, and (c) various covering problems appear in the analysis of various voting problems, thus our results should translate (more or less directly) to those problems as well. Due to lack of space, we omit some proof details.

2 Preliminaries

We model an election as 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 a collection of voters. Each voter is represented through his or her preferences. For the case of Approval voting, each voter's preferences take the form of a set of candidates approved by this voter. The candidate(s) receiving the most approvals are the winner(s), that is, we assume the nonunique-winner model (if several candidates have the same number of approvals then we view each of them as winning). We write $\mathrm{score}_E(c_i)$ to denote the number of voters approving c_i in election E. We refer to elections that use Approval voting and represent voter preferences in this way as approval elections. In a weighted election, in addition to their preferences, voters also have integer weights. A voter v with weight $\omega(v)$ counts as $\omega(v)$ copies of an unweighted voter.

Let us review the most essential notions of parameterized complexity theory. A parameterized problem is a problem with a certain feature of the input distinguished as the parameter. For example, for our election problems, the parameter will always be the number m of candidates in the election. A problem is fixed-parameter tractable (in FPT) if there exists an algorithm that, given an instance I with parameter k, can compute the answer to this problem in time $f(k) \cdot |I|^{O(1)}$, where f is a computable function and $|I|$ is the length of the encoding of I. A parameterized problem is in XP if there exists an algorithm that, given an instance I with parameter k, can compute the answer to this problem in time $|I|^{f(k)}$, where f is some computable function. In other words, XP is the class of those problems that can be solved in polynomial time under the assumption that the parameter is a constant. In contrast, problems which are NP-hard even for constant values of the parameter are said to be Para-NP-hard with respect to the parameter. For further information, we point the readers to textbooks on parameterized complexity theory [9, 15, 22]).

3 Covering and Voting: Technique Showcase

In this section we present our main results and techniques. We start by showing a relation between election problems for the Approval rule and several covering problems. Then we present a technique for obtaining FPT results for these problems, and finally we evaluate our algorithms empirically.

3.1 From Approval Voting to Covering Problems

We are interested in the following three problems.

Definition 1 (Bartholdi et al. [2], Faliszewski et al. [12, 21]). In each of the problems Approval-$BRIBERY (priced bribery), Approval-$CCAV (priced control by adding voters), and Approval-$CCDV (priced control by deleting voters), we are given an approval election $E = (C, V)$ with $C = \{p, c_1, \ldots, c_m\}$ and $V = (v_1, \ldots, v_n)$, and an integer budget B. In each of the problems the goal is to decide whether it is possible to ensure that p is a winner, at a cost of at most B. The problems differ in the allowed actions and possibly in some additional parts of the input:

1. In Approval-$BRIBERY, for each voter v_i, $1 \leq i \leq n$, we are given a nonnegative integer price π_i; for this price we can change the voter's approval set in any way we choose.
2. In Approval-$CCAV (CCAV stands for "Constructive Control by Adding Voters") we are given a collection $W = (w_1, \ldots, w_{n'})$ of additional voters. For each additional voter w_i, $1 \leq i \leq n'$, we also have a nonnegative integer price π_i for adding w_i to the original election.
3. In Approval-$CCDV (CCDV stands for "Constructive Control by Deleting Voters"), we have a nonnegative integer price π_i for removing each voter v_i from the election.

In the weighted variants of these problems (which we denote by putting "WEIGHTED" after "Approval"), the input elections (and all the voters) are weighted; in particular, each voter v has an integer weight $\omega(v)$.

The unpriced variants of these problems (denoted by omitting the dollar sign from their names) are defined identically, except that all prices have the same unit value.

The above problems are, in essence, equivalent to certain covering problems, thus in this paper we focus on the complexity of these covering problems. As many of these covering problems consider multisets, the following notation will be useful. If A is a multiset and x is some element, then we write $A(x)$ to denote the number of times x occurs in A (that is, $A(x)$ is x's multiplicity in A). If x is not a member of A, then $A(x) = 0$.

Definition 2. In the WEIGHTED MULTISET MULTICOVER (WMM) problem we are given a family $\mathcal{S} = \{S_1, \ldots, S_n\}$ of multisets over the universe $U = \{x_1, \ldots, x_m\}$, integer weights $w_1, \ldots, w_n$ for the multisets,[2] integer covering requirements $r_1, \ldots, r_m$

[2] There is a name clash between the literature on covering problems and that on elections. In the former, "weights" refer to what voting literature would call "prices." Weights of the voters are modeled as multiplicities of the elements in the multisets. We kept the naming conventions from respective parts of the literature to make our results more accessible to researchers from both communities.

for the elements of the universe, and an integer budget B. We ask whether there is a subfamily $\mathcal{S}' \subseteq \mathcal{S}$ of multisets from $\mathcal{S}$ such that: (a) for each $x_i \in U$ it holds that $\sum_{S_j \in \mathcal{S}'} S_j(x_i) \geq r_i$ (that is, each element x_i is covered by at least the required number of times), and (b) $\sum_{S_j \in \mathcal{S}'} w_j \leq B$ (the budget is not exceeded).

Briefly put, the relation between WMM and various election problems (as those defined above) is that the universe corresponds to the candidates in the election, the multisets correspond to the voters, and the covering requirements depend on particular actions that we are allowed to perform.

Example 1. Consider an instance of Approval-$CCDV with election $E = (C, V)$, where $C = \{p, c_1, \ldots, c_m\}$ and $V = (v_1, \ldots, v_n)$, with prices $\pi_1, \ldots, \pi_n$ for voters not to participate in the election, and with budget B. We can express this instance as an instance of WEIGHTED MULTISET MULTICOVER as follows. For each voter v_i not approving p, we form a multiset S_i with weight π_i that includes exactly the candidates approved by v_i, each with multiplicity exactly one. For each candidate c_i, $1 \leq i \leq m$, we set its covering requirement to be $\max(\mathrm{score}_E(c_i) - \mathrm{score}_E(p), 0)$. It is easy to see that there is a way to ensure p's victory by deleting voters of total cost at most B if and only if it is possible to solve the presented instance of WEIGHTED MULTISET MULTICOVER with budget B.

It is clear that we do not need the full flexibility of WMM in Example 1; it suffices to use WEIGHTED SET MULTICOVER where each input multiset has multiplicities from the set $\{0, 1\}$ (in other words, the family $\mathcal{S}$ contains sets without multiplicities, but the union operation takes multiplicities into account). This is quite important since unrestricted WMM is NP-hard even for the case of a single-element universe, by a polynomial-time reduction from PARTITION.

Proposition 1. WMM *is* NP*-hard even for the case of a single element in the universe.*

Another variant of WMM is MULTISET MULTICOVER, where we assume each set to have unit weight. By generalizing the proof for Proposition 1, we show that this problem is NP-hard already for two-element universes.

Proposition 2. MULTISET MULTICOVER *is* NP*-hard even for universes of size two.*

From the viewpoint of voting theory, it is more interesting to consider an even more restricted variant of MULTISET MULTICOVER, where for each multiset S_i in the input instance there is a number t_i such that for each element x we have $S_i(x) \in \{0, t_i\}$ (in other words, elements within a single multiset have the same multiplicity). We refer to this variant of the problem as UNIFORM MULTISET MULTICOVER. Using an argument similar to that used in Example 1, it is easy to show that UNIFORM MULTISET MULTICOVER is, in essence, equivalent to Approval-WEIGHTED-CCDV.

In Example 1 we have considered Approval-$CCDV because, among our problems, it is the most straightforward one to model via a covering problem. Nonetheless, constructions with similar flavor are possible both for Approval-$CCAC (by, in effect, counting how many times each candidate is not approved) and for Approval-$BRIBERY (by slightly more complicated tricks). Formally, we have the following result.

Proposition 3. *If* WEIGHTED SET MULTICOVER *parameterized by the universe size is in* FPT*, then, parameterized by the number of candidates, each of Approval-*$\$$BRIBERY*, Approval-*$\$$CCAV*, and Approval-*$\$$CCDV *is in* FPT.

The same applies to UNIFORM MULTISET MULTICOVER and the weighted-but-unpriced variants of our problems.

3.2 The Main Theoretical Results

Our main theoretical results are FPT algorithms for WEIGHTED SET MULTICOVER and UNIFORM MULTISET MULTICOVER parameterized by the universe size.[3]

Theorem 1. WEIGHTED SET MULTICOVER *is in* FPT *when parameterized by the universe size.*

Proof. Consider an instance of WEIGHTED SET MULTICOVER with universe $U = \{x_1, \ldots, x_m\}$, family $\mathcal{S} = \{S_1, \ldots, S_n\}$ of subsets, weights $w_1, \ldots, w_n$ for the sets, covering requirements $r_1, \ldots, r_m$ for the elements, and budget B. Our algorithm proceeds by solving an appropriate mixed integer linear program.

First, we form a family $U_1, \ldots, U_{2^m}$ of all subsets of U. For each i, $1 \leq i \leq 2^m$, let $\mathcal{S}(U_i) := \{S_j \in \mathcal{S} \mid S_j = U_i\}$. For each i and j, $1 \leq i \leq 2^m$ and $1 \leq j \leq |\mathcal{S}(U_i)|$, we write $\mathcal{S}(U_i, j)$ to denote the set from $\mathcal{S}(U_i)$ with the jth lowest weight (breaking ties in a fixed arbitrary way). Similarly, we write $w(U_i, j)$ to mean the weight of $\mathcal{S}(U_i, j)$ and we define $w(U_i, 0) = 0$ (in other words, we group the sets from $\mathcal{S}$ based on their content and sort them with respect to their weights). Given this setup, we form our mixed integer linear program.

We have 2^m integer variables z_i, $1 \leq i \leq 2^m$. Intuitively, these variables describe how many sets we take from each type. We also have $2^m n$ regular (rational) variables $y_{i,j}$, $1 \leq i \leq 2^m$, $0 \leq j \leq n-1$, which are used to model the total weight of the selected sets. We introduce the following constraints:

***Part 1* constraints.** For each i, $1 \leq i \leq 2^m$, we have constraints $z_i \geq 0$ and $z_i \leq |S(U_i)|$. For each element x_ℓ of the universe, we also have constraint $\sum_{U_i :\, x_\ell \in U_i} z_i \geq r_\ell$. These constraints ensure that the variables z_i describe a possible solution for the problem (disregarding the budget).

***Part 2* constraints.** For each i and j, $1 \leq i \leq 2^m$, $0 \leq j \leq n-1$, we have constraints: $y_{i,j} \geq 0$ and $y_{i,j} \geq z_i - j$. The intended meaning of variable $y_{i,j}$ is as follows. If the solution described by variables $z_1, \ldots, z_{2^m}$ includes fewer than j sets from $\mathcal{S}(U_i)$, then $y_{i,j} = 0$; otherwise, $y_{i,j}$ says that after we added the j lowest-weight sets from family $\mathcal{S}(U_i)$, we still need to add $y_{i,j}$ more sets from this family (however, note that these variables are not required to be integers, thus the following constraint is designed in such a way that inaccurate—too large—values of these variables do not affect correctness).

[3] Remarkably, under reasonable complexity-theoretic assumptions, Dom et al. [7] have shown that no polynomial-size kernels exist for the special case of SET COVER, parameterized by the universe size and the solution size.

Our final constraint uses variables $y_{i,j}$ to express the requirement that the solution has cost at most B:

$$\sum_{i=1}^{2^m} \sum_{j=0}^{n-1} y_{i,j}(w(U_i, j+1) - w(U_i, j)) \leq B.$$

To understand this constraint, let us first replace each $y_{i,j}$ with $\max(0, z_i - j)$. Now, note that for each fixed value of i, the "internal sum" over j gives the weight of the cheapest z_i sets from $\mathcal{S}(U_i)$ (specifically, we first take z_i times $w(U_i, 1)$ because we know that each set costs at least this much, then to that we add $z_i - 1$ times $w(U_i, 2) - w(U_i, 1)$, because short of the first set from $U(S_i)$, each set costs at least $w(U_i, 2)$, and so on). To see that the constraint is correct, note that, for each $y_{i,j}$, we have that $y_{i,j} \geq \max(0, z_i - j)$ and the smaller the values $y_{i,j}$, the smaller the sum computed in this constraint.

Finally, to solve this mixed integer linear program we invoke Lenstra's famous result in its variant for mixed integer programming (see [19, Section 5]). □

The mixed integer linear program that we construct has two main parts. Part 1 simply specifies what it means to solve the problem at hand, without worrying about the budget. Part 2 uses the fact that we pick the sets that implement the solution expressed in Part 1 in the increasing order of weights, to compute the total cost of the solution through rational-valued variables. The key observation is that the closer the required budget B is to the cost of the optimal solution, the closer are variables $y_{i,j}$ to integer values. Using the same technique we can show that UNIFORM MULTISET MULTICOVER is in FPT (Part 2 of our program is slightly different in this case to account for the fact that now we pick sets with particular content in the order of decreasing multiplicities).

Theorem 2. UNIFORM MULTISET MULTICOVER *is in* FPT *when parameterized by the universe size.*

Unfortunately, it is impossible to apply our approach to the more general MULTISET MULTICOVER problem; by Proposition 2, the problem is already NP-hard for two-element universes. It is, however, possible to obtain a certain form of an FPT approximation scheme.[4]

Definition 3. Let ϵ be a real number, $\epsilon > 0$. We say that algorithm $\mathcal{A}$ is an *ϵ-almost-cover algorithm* for MULTISET MULTICOVER if, given an input instance I with universe $U = \{x_1, \ldots, x_m\}$ and covering requirements $r_1, \ldots, r_m$, it outputs a solution that covers each element x_i with multiplicity r'_i such that $\sum_i \max(0, r_i - r'_i) < \epsilon \sum_i r_i$.

In other words, on the average an ϵ-almost-cover algorithm can miss each element of the universe by an ϵ-fraction of its covering requirement. For the case where we really need to cover all the elements perfectly, we might first run an ϵ-almost-cover algorithm and then complement its solution, for example, in some greedy way. Indeed, the remaining instance might be much easier to solve.

The key idea regarding computing an ϵ-almost-cover is that it suffices to replace each input multiset by several submultisets, each with a particular "precision level," so

[4] While this result does not, as of yet, have direct application to voting, we believe it is quite interesting in itself.

that multiplicities of the elements in each submultiset are of a similar order of magnitude.

Theorem 3. *For every rational $\epsilon > 0$, there is an* FPT *ϵ-almost-cover algorithm for* MULTISET MULTICOVER *parameterized by the universe size.*

Proof. Throughout this proof we describe our ϵ-almost-cover algorithm for MULTISET MULTICOVER. We consider an instance I of MULTISET MULTICOVER with a family $\mathcal{S} = \{S_1, \ldots, S_n\}$ of multisets over the universe $U = \{x_1, \ldots, x_m\}$, where the covering requirements for the elements of the universe are $r_1, \ldots, r_m$. We associate each set S from the family $\mathcal{S}$ with the vector $v_S = \langle S(x_1), S(x_2), \ldots, S(x_m)\rangle$ of element multiplicities.

Let $\epsilon > 0$ be the desired approximation ratio. We fix $Z = \lceil \frac{4m}{\epsilon} \rceil$ and $Y = Z + \lceil \frac{4Zm^3}{\epsilon} \rceil$. Notice that $\frac{m}{Z} \leq \frac{\epsilon}{4}$ and $\frac{Zm^3}{Y-Z} \leq \frac{\epsilon}{4}$. Let $X = \left(\frac{2Y^m}{\epsilon} + 1\right)^m$ and let $V_1, \ldots, V_X$ be a sequence of all m-dimensional vectors whose entries come from the $\left(\frac{2Y^m}{\epsilon} + 1\right)$-element set $\{0, \frac{\epsilon}{2}, \epsilon, \frac{3\epsilon}{2}, 2\epsilon, \ldots, Y^m\}$. For each j, $1 \leq j \leq X$, we write $V_j = \langle V_j(x_1), V_j(x_2), \ldots, V_j(x_m)\rangle$. Intuitively, these vectors describe some subset of "shapes" of all possible multisets—interpreted as vectors of multiplicities—over our m-element universe. For each number β, we write βV_i to mean the vector $\langle \lfloor \beta V_{i,1} \rfloor, \lfloor \beta V_{i,2} \rfloor, \ldots, \lfloor \beta V_{i,m} \rfloor\rangle$.

Intuitively, vectors of the form βV_i are approximations of those multisets for which the positive multiplicities of the elements do not differ between each other too much (formally, for those multisets for which the positive multiplicities differ by at most a factor of $\frac{2Y^m}{\epsilon}$). Indeed, for each such set S, we can find a value β and a vector V_j such that for each element x_i it holds that $S(x_i) \geq V_j(x_i) \geq \left(1 - \frac{\epsilon}{2}\right) S(x_i)$. However, this way we cannot easily approximate those sets for which multiplicities differ by a large factor. For example, consider a set S represented through the vector $\langle 0, \ldots, 0, 1, Q\rangle$, where $Q \gg \frac{2Y^m}{\epsilon}$. For each value β and each vector V_j, the vector βV_j will be inaccurate with respect to the multiplicity of element x_{m-1} or inaccurate with respect to the multiplicity of element x_m (or inaccurate with respect to both these multiplicities).

The main step of our algorithm is to modify the instance I so that we replace each set S from the family $\mathcal{S}$ with a sequence of vectors of the form βV_j that altogether add to at most the set S (each such sequence can contain multiple vectors of different "shapes" V_j and of different scaling factors β). The goal is to obtain an instance that on the one hand consists of "nicely-structured" sets (vectors) only, and on the other hand has the following property: If in the initial instance I there exist K sets that cover elements $x_1, \ldots x_m$ with multiplicities $r_1, \ldots, r_m$, then in the new instance there exist K sets that cover elements $x_1, \ldots, x_m$ with multiplicities $r'_1, \ldots, r'_m$, such that $\sum_i \max(0, r_i - r'_i) < \epsilon \sum_i r_i$. We refer to this as the *almost-cover approximation property*.

The procedure for replacing a given set S is presented as Algorithm 1. This algorithm calls the `Emit` function with arguments (β, V) for each vector βV that it wants to output (V is always one of the vectors $V_1, \ldots, V_X$). The emitted sets replace the set S from the input. Below we show that if we apply Algorithm 1 to each set from $\mathcal{S}$, then the resulting instance I' has our almost-cover approximation property.

Let us consider how Algorithm 1 proceeds on a given set S. For the sake of clarity, let us assume there is no rounding performed by Algorithm 1 in function `Round_And_Emit` (the loop in line 29). We will go back to this issue later.

Algorithm 1: The transformation algorithm used in the proof of Theorem 3—the algorithm replaces a given set S with a sequence of vectors of the form βV_j.

```
1  Main(S):
2      multip ← ⟨(1, S(x_1)), (2, S(x_2)), ..., (m, S(x_m))⟩;
3      sorted ← sort(multip) ;                    // sort in ascending order of multiplicities
4      i ← 0 ;
       //   sorted[i].first refers to the i'th item's number
       //   sorted[i].second refers to its multiplicity
5      while sorted[i].second = 0 do
6          i ← i + 1 ;
7      Main_Rec(i, sorted) ;
8
9  Main_Rec(i, multip):
10     V ← ⟨0, 0, ..., 0⟩ (vector of m zeros). ;
11     β ← multip[i].second ;
12     V[multip[i].first] ← 1 ;
13     i ← i + 1 ;
14     while i ≤ m do
15         if multip[i].second < Y · multip[i−1].second then
16             V[multip[i].first] ← multip[i].second / β ;
17             i ← i + 1 ;
18         else
19             for j ← i to m do
20                 V[multip[j].first] ← Z·multip[i−1].second / β ;
21             Round_And_Emit(β, V) ;
22             for j ← 1 to m do
23                 multip[i].second ← multip[i].second − βV[multip[i].first] ;
24             Main_Rec(i, multip) ;
25             return
26     Round_And_Emit(β, V);
27
28 Round_And_Emit(β, V):
29     for ℓ ← 1 to m do
30         V[ℓ] ← ⌊2V[ℓ] / ε⌋ / (ε/2);
31     Emit(β, V);
```

The algorithm considers the elements of the universe—indexed by variable i throughout the algorithm—in the order given by the vector "sorted" (formed in line 3 of Algorithm 1). Let $\prec$ be the order in which Algorithm 1 considers the elements (so $x_{i'} \prec x_{i''}$ means that $x_{i'}$ is considered before $x_{i''}$), and let $x'_1, \ldots, x'_m$ be the elements from the universe renamed so that $x'_1 \prec x'_2 \prec \cdots \prec x'_m$. Let r be the number of sets that Algorithm 1 emits on our input set S and let these sets be $S_1, S_2, \ldots, S_r$. (This is depicted on Figure 1, where for the sake of the example we take $m = 6$ and $r = 3$.)

Consider the situation where the algorithm emits the k'th set, S_k, and let i_k be the value of variable i right before the call to `Round_And_Emit` that caused S_k to be

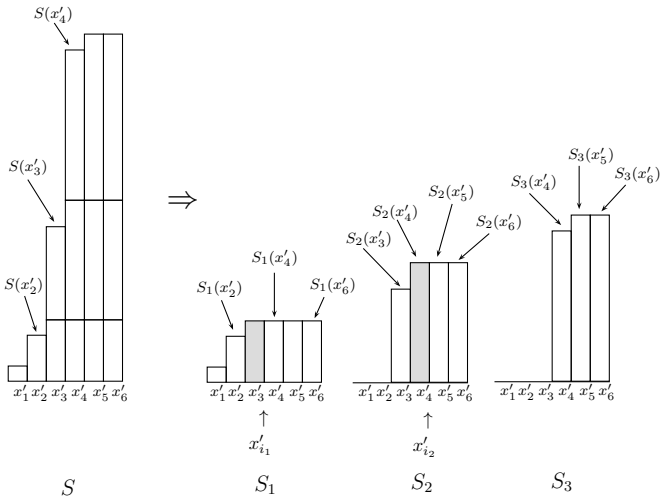

Fig. 1: An example transformation performed by Algorithm 1; the algorithm replaces some set S with three sets: S_1, S_2, and S_3.

emitted. Note that each element x from the universe such that $x_{i_k} \prec x$ has the same multiplicity in S_k as element x_{i_k} (line 19 of Algorithm 1). Let $t_k = \sum_j S_k(x'_j)$ be the sum of the multiplicities of the elements from S_k. We make the following observations:

Observation 1: $S_k(x'_{i_k}) = Z \cdot S_k(x'_{i_{k-1}})$.

Observation 2: It holds that $S_{k+1}(x'_{i_k}) \geq Y \cdot S_k(x'_{i_k-1}) - Z \cdot S_k(x'_{i_k-1}) = (Y - Z)S_k(x'_{i_k-1}) = \frac{(Y-Z)}{Z} S_k(x'_{i_k})$.

Observation 3: We have that $S_{k+1}(x'_{i_k}) \geq \frac{(Y-Z)}{Z} S_k(x'_{i_k}) \geq \frac{(Y-Z)}{Zm} t_k$. Further, we have that $S_{k+1}(x'_{i_{k+1}}) \geq S_{k+1}(x'_{i_k}) \geq \frac{(Y-Z)}{Zm^2} \sum_{j \leq k} t_j$.

Observation 4: For $i < i_k$ it holds that $\sum_{q \leq k} S_q(x'_i) = S(x'_i)$.

Now let us consider some solution for instance I that consists of K sets, $\mathcal{S}^{\text{opt}} = \{S_1^{\text{opt}}, S_2^{\text{opt}}, \ldots, S_K^{\text{opt}}\} \subseteq \mathcal{S}$. These sets, altogether, cover all the elements from the universe with required multiplicities, that is, it holds that for each i we have $\sum_{S \in \mathcal{S}^{\text{opt}}} S(x_i) \geq r_i$. For each set $S \in \mathcal{S}^{\text{opt}}$ and for each element x_i from the universe, we pick an arbitrary number $y_{S,i}$ so that altogether the following conditions hold:

1. For every set $S \in \mathcal{S}^{\text{opt}}$ and every x_i, $y_{S,i} \leq S(x_i)$.
2. For every x_i, $\sum_{S \in \mathcal{S}^{\text{opt}}} y_{S,i} = r_i$.

Intuitively, for a given set S, the values $y_{S,1}, y_{S,2}, \ldots, y_{S,m}$ describe the multiplicities of the elements from S that are *actually used* to covers the elements. Based on these numbers, we will show how to replace each set from $\mathcal{S}^{\text{opt}}$ with one of the sets emitted for it, so that the resulting family of sets has the almost-cover approximation property.

Consider a set $S \in \mathcal{S}^{\text{opt}}$ for which Algorithm 1 emits r sets, $S_1, S_2, \ldots, S_r$. As in the discussion of Algorithm 1, let $x'_1, \ldots, x'_m$ be the elements from the universe in

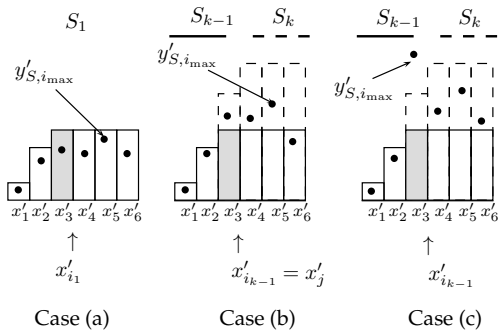

Fig. 2: Three different cases considered in the proof of Theorem 3. The bullets depict the values $y'_{S,1}, y'_{S,2}, \ldots, y'_{S,m}$.

which Algorithm 1 considers them (when emitting sets for S). We write $y'_{S,i}$ to mean the value $y_{S,j}$ such that $x_j = x'_i$. Let $\mathcal{R} = \{S_1, S_2, \ldots, S_r\}$, let $i_{\max} = \operatorname{argmax}_i y'_{S,i}$, and let S_{repl} be the set from $\mathcal{R}$ defined in the following way:

1. If for every set $S_k \in \mathcal{R}$ we have $S_k(x'_{i_{\max}}) < y'_{S,i_{\max}}$, then S_{repl} is the set $S_k \in \mathcal{R}$ with the greatest value $S_k(x'_{i_{\max}})$ (the set that covers element $x'_{i_{\max}}$ with the greatest multiplicity). This is the case denoted as "Case (c)" in Figure 2.
2. Otherwise S_{repl} is the set $S_k \in \mathcal{R}$ that has the lowest value $S_k(x'_{i_{\max}})$, yet no-lower than $y'_{S,i_{\max}}$. This is the case denoted as either "Case (a)" or "Case (b)" in Figure 2.

We now show that S_{repl} is a good candidate for replacing S, that is, that $\sum_i \max(0, y'_{S,i} - S_{\text{repl}}(x'_i)) < \epsilon \sum_i y'_{S,i}$. To this end, we consider the three cases depicted in Figure 2:

Case (a) It holds that $y'_{S,i_{\max}} < S_1(x'_{i_{\max}})$ (that is, S_1 already covers the most demanding element of the universe to the same extent as S does). This means that we have $\sum_\ell \max(0, y'_{S,\ell} - S_1(x'_\ell)) = 0$. By the criterion for choosing set S_{repl}, we have that $S_{\text{repl}} = S_1$.

Case (b) There exist sets $S_{k-1}, S_k \in \mathcal{R}$ such that $S_k(x'_{i_{\max}}) \geq y'_{S,i_{\max}} > S_{k-1}(x'_{i_{\max}})$ (and thus, $S_{\text{repl}} = S_k$). Let $x'_j = x'_{i_{k-1}}$ (recall from the discussion of Algorithm 1 that i_{k-1} is the index of the universe element which caused emitting S_{k-1}). Let us consider two subcases:

(i) $y'_{S,i_{\max}} \leq S_k(x'_j)$: We first note that for each $i \geq j$ it holds that $y'_{S,i} \leq S_k(x'_i)$. Further, for each $i < j$, we have $y'_{S,i} \leq \sum_{\ell \leq k-1} S_\ell(x'_i)$ (this follows from Observation 4 and the fact that $y'_{S,i} \leq S(x'_i)$). Based on this inequality, we get:

$$\begin{aligned}
\sum_{i<j} y'_{S,i} &\leq \sum_{i<j} \sum_{\ell \leq k-1} S_\ell(x'_i) \leq \sum_{\ell \leq k-2} t_\ell + \sum_{i<j} S_{k-1}(x'_i) \\
&\leq \frac{Zm^2}{(Y-Z)} S_{k-1}(x'_j) + \frac{m}{Z} S_{k-1}(x'_j) \qquad \text{(Observations 3 and 1)} \\
&\leq \frac{\epsilon}{2} S_{k-1}(x'_j) \leq \frac{\epsilon}{2} y_{S,i_{\max}}.
\end{aligned}$$

In consequence, it holds that $\sum_\ell \max(0, y'_{S,\ell} - S_k(x'_\ell)) < \frac{\epsilon}{2} \sum_\ell y'_{S,\ell}$.

(ii) $y'_{S,i_{\max}} > S_k(x'_j)$: We omit the proof that in this case it also holds that $\sum_\ell \max(0, y'_{S,\ell} - S_k(x'_\ell)) \leq \frac{\epsilon}{2} \sum_\ell y'_{S,\ell}$.

Case (c) Every set $S_k \in \mathcal{R}$ has $S_k(x'_{i_{\max}}) < y'_{S,i_{\max}}$. We omit the proof that in this case it holds that $\sum_\ell \max(0, y_{S,\ell} - S_k(x'_\ell)) \leq \frac{\epsilon}{2} \sum_\ell y'_{S,\ell}$.

The above case analysis almost shows that we indeed have the almost-cover approximation property. It remains to consider the issue of rounding (Line 29 of Algorithm 1). This rounding introduces inaccuracy that is bounded by factor $\frac{\epsilon}{2}$ and thus, indeed, we do have the almost-cover approximation property.

Now, given the new instance I', it suffices to find a solution for I' that satisfies the desired approximation guarantee (that is, a collection $\mathcal{S}'$ of at most K sets that form an ϵ-almost-cover). It is possible to do so through a mixed integer linear program (and an application of the Lenstra's algorithm [19]). We omit the details due to space (we mention that since in I' all the sets are represented through vectors of the form βV_j, we can bound the number of integer variables by a function of the size of the universe). The final output of our algorithm is as follows: For each set S from the original family $\mathcal{S}$, we output S if $\mathcal{S}'$ contains at least one set emitted for S. □

For the case of WEIGHTED SET MULTICOVER we show a more standard variant of an FPT approximation scheme.

Definition 4. Let $\varepsilon, 0 < \varepsilon < 1$, be a real number. A $(1+\varepsilon)$-approximation algorithm for WEIGHTED SET MULTICOVER is an algorithm that, given an instance of the problem, outputs a solution satisfying all covering requirements, but whose weight is at most $1 + \varepsilon$ times the weight of the optimal one.

As opposed to all the previously presented algorithms (including the one from Theorem 3), the next algorithm does not rely on solving (M)ILP instances. The main idea is to use a refined variant of brute-force search which considers for each type of sets only a set of promising numbers of occurrences in the solution instead of considering every possible number of occurrences.

Theorem 4. *For each ε, $0 < \varepsilon < 1$, there is a $(1 + \varepsilon)$-approximation algorithm for* WEIGHTED SET MULTICOVER *that runs in time $O(\lceil 2^m/\varepsilon + 1 \rceil^{2^m} n^2 m)$.*

We conclude this section by translating the results from the world of covering problems to the world of approval elections. We obtain the following corollary.

Corollary 1. APPROVAL-$CCAV, APPROVAL-$CCDV, APPROVAL-$BRIBERY, *as well as* APPROVAL-WEIGHTED-CCAV *and* APPROVAL-WEIGHTED-CCDV, *are in* FPT, *when parameterized by the number of candidates.*

Contrarily, it is either shown explicitly by Faliszewski et al. [12], or follows trivially, that these problems with both prices and weights are NP-hard already for two candidates (that is, Para-NP-hard with respect to the number of candidates).

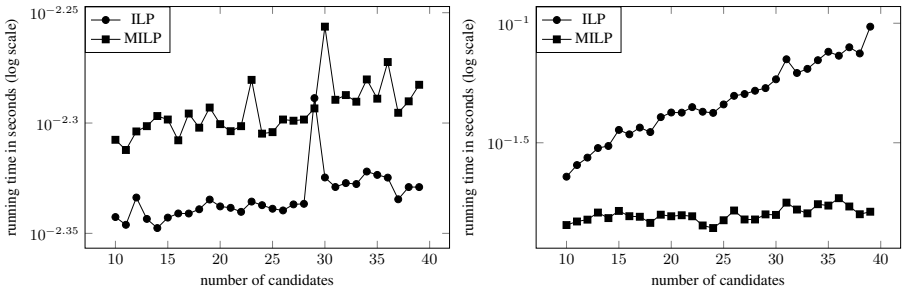

Fig. 3: Running time depending on the number of candidates. Left: plain test series, right: duplicated voters.

3.3 Preliminary Empirical Evaluation

In this section we evaluate our algorithms empirically. Specifically, we focus on the MILP-based algorithm from subsection 3.2, as applied to Approval-$CCDV, and on the standard ILP algorithm for this problem (see below). In both cases, instead of using the very slow algorithm of Lenstra (with running time being roughly $(2^m)^{(2^m)}$ [19, 16, 18]), we chose an off-the-shelf solver (CPLEX). The main purpose of the experiments is to explore whether our (M)ILP-based FPT algorithms are practical to use. Thus, we focus on evaluating the running time. (We point the reader, for example, to the work of Erdélyi et al. [11] for an example of a much more detailed experimental analysis of control problems in elections for several voting rules.)

Test Data. We use Preflib [20] as a well-known source for real-world elections. Since Preflib contains only few elections with approval preferences (provided through linear orderings with ties containing exactly two groups of tied candidates each), we used elections with strict linear-order preferences and for each voter we uniformly at random chose how many of the top candidates this voter approves of.

Test Series. We focus on Approval-$CCDV since, among our problems, it requires the least amount of information to be added to the elections to obtain full instances. Specifically, we only need to choose the preferred candidate p and the prices for deleting the voters. We performed two test series, *duplicated voters* and *plain*. In the duplicated voters test series we interpret the Preflib elections as random samples of larger elections and, for each election from PrefLib, duplicate each voter between 1 and 500 times (the multiplication factor was chosen uniformly at random for each voter separately). In the plain test series, the set of voters remains unchanged. For each voter (after the duplication) we set the price for its deletion uniformly at random to be an integer between 1 and 500. Finally, we uniformly at random select one candidate to be the preferred one and create for each Preflib election and each $m \in \{10, 12, \ldots, 40\}$, ten instances with m candidates (by first creating the full instance and removing all but m randomly chosen candidates). All in all, we obtained more than 8000 test instances for each test series.

We stress that our focus is on the running times of our algorithms, and not—for example—on modeling how frequent control might be in real-life settings. The purpose of the experiments is to be a proof-of-concept of the algorithms suggested herein.

Algorithms. We tested two algorithms, both of which transfer the Approval-$CCDV instance into a WEIGHTED SET MULTICOVER instance and apply CPLEX to solve a (mixed) integer program. The first algorithm (referred to as ILP) uses a straightforward integer linear programming formulation with one binary variable for each set (representing presence in the solution), constraints ensuring that each element is appropriately covered by the selected sets, and an objective function minimizing the total costs. The second algorithm (referred to as MILP) uses the mixed integer linear programming formulation from Theorem 1. We did not consider brute-force or approximation approaches since both (M)ILP-based algorithms turned out to be extremely fast for the Preflib data set (and always return optimal solutions).

Results. Surprisingly, both algorithms solved all instances very fast (using at most a few seconds). For the plain test series the running time slightly increases as the number of candidates increases, for both algorithms. A possible explanation is that the program description as well as the number of variables[5] also slightly increases. The ILP is faster by roughly a constant factor which might be caused by its much simpler formulation and the usage of binary variables instead of integer ones. For the duplicated voters test series, the situation changes: the running time increases only slightly with the increase of the number of candidates for the MILP, but it increases significantly for the ILP. A possible explanation is that the ILP has one variable for each voter whereas the MILP has one variable for each class of duplicated voters. See Figure 3 for an illustration.

4 Generalizations

We now consider the ordinal model of elections, where each voter's preferences are represented as an order, ranking the candidates from the most preferred one to the least preferred one. For example, for $C = \{c_1, c_2, c_3\}$, vote $c_1 \succ c_3 \succ c_2$ means that the voter likes c_1 best, then c_3, and then c_2.

There are many different voting rules for the ordinal election model. Here we concentrate only on scoring rules. A scoring rule for m candidates is a nondecreasing vector $\alpha = (\alpha_1, \ldots, \alpha_m)$ of integers. Each voter gives α_1 points to his or her most preferred candidate, α_2 points to the second most preferred candidate, and so on. Examples of scoring rules include the Plurality rule, defined through vectors of the form $(1, 0, \ldots, 0)$, k-Approval, defined through vectors with k ones followed by $m - k$ zeroes, and Borda count, defined through vectors of the form $(m - 1, m - 2, \ldots, 0)$.

For each voting rule $\mathcal{R}$ in the ordinal model, it is straightforward to define $\mathcal{R}$-$CCAV, $\mathcal{R}$-$CCDV, and $\mathcal{R}$-$BRIBERY (compared to the Approval variants of these problems, we represent voter preferences through preference orders and we compute winners using $\mathcal{R}$ instead of Approval). By our MILP technique from Theorem 1, we get the following.

Theorem 5. *For every voting rule $\mathcal{R}$ for which winner determination can be expressed through a set of integer linear inequalities over variables that indicate how many voters*

[5] By removing candidates during instance generation one also removes voters only approving removed candidates.

with each given preference order are in the election, $\mathcal{R}$-$CCAV*,* $\mathcal{R}$-$CCDV*, and* $\mathcal{R}$-$BRIBERY *are in* FPT *when parameterized by the number of candidates.*

For a more detailed description of what we mean by "expressing the winner determination problem through integer linear inequalities," we point to the discussions by Dorn and Schlotter [8] or by Faliszewski et al. [14]. As an important example, Theorem 5 applies to all scoring rules.

The proof of Theorem 5 follows the same structure as that of Theorem 1; we replace Part 1 with an integer linear program describing the appropriate bribery/control action (and winner determination), and we modify Part 2 to compute the cost of this action.

We also partially resolve an open problem posed by Bredereck et al. [5] regarding SHIFT BRIBERY. In this problem we are given an election and a preferred candidate p, and the goal is to ensure p's victory by shifting p forward in some of the votes (the cost of each shift depends on the number of positions by which we shift p). Under the sortable prices assumption, we know that voters with the same preference orders can be sorted so that if voter v' precedes voter v'', then we know that shifting p by each given number of positions i in the vote of v' costs at most as much as doing the same in the vote of v'' (due to this assumption, introduced by Bredereck et al. [5], the problem has the monotonicity property on which our MILP approach relies; all-or-nothing prices are a special case of sortable prices where we always shift p to the top of a given vote or we leave the vote unchanged).

Theorem 6. *For Borda (and for Maximin and Copeland voting rules),* SHIFT BRIBERY *for sortable price functions and for all-or-nothing price functions is in* FPT *when parameterized by the number of candidates.*

Bredereck et al. [5] gave an FPT approximation scheme for the variants of SHIFT BRIBERY from Theorem 6. Their algorithm rephrases the problem and then applies a bounded search through the solution space. We use their rephrasing but replace the search with our MILP technique.

5 Conclusions

We have studied election control and bribery for the case of few voters with prices. We also considered weighted Approval elections with few voters. By developing a very general proof technique, in these settings we have improved known XP-membership results to FPT-membership results. We have also tested our algorithms empirically and found them to be extremely fast. Our empirical results provide some evidence for the correlation between running time and number of candidates, as given by the FPT-classification, at least for Preflib-based Approval-$CCDV test instances.

Our paper leads to several possible directions for future work. First, the experiments we have presented are only preliminary and they should be extended in a number of ways. Second, it would be very interesting to further explore the relevance of FPT approximation algorithms to other voting scenarios.

Bibliography

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

[2] J. J. Bartholdi, III, C. A. Tovey, and M. A. Trick. How hard is it to control an election. *Mathematical and Computer Modelling*, 16(8–9):27–40, 1992.

[3] D. Baumeister, P. Faliszewski, J. Lang, and J. Rothe. Campaigns for lazy voters: Truncated ballots. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS '12)*, pages 577–584, June 2012.

[4] R. Bredereck, J. Chen, P. Faliszewski, J. Guo, R. Niedermeier, and G. J. Woeginger. Parameterized algorithmics for computational social choice: Nine research challenges. *Tsinghua Science and Technology*, 19(4):358–373, 2014.

[5] R. Bredereck, J. Chen, P. Faliszewski, A. Nichterlein, and R. Niedermeier. Prices matter for the parameterized complexity of shift bribery. In *Proceedings of The Twenty-Eighth AAAI Conference on Artificial Intelligence (AAAI '14)*, pages 1398–1404, 2014.

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

[7] M. Dom, D. Lokshtanov, and S. Saurabh. Kernelization lower bounds through colors and IDs. *ACM Transactions on Algorithms*, 11(2):13:1–13:20, 2014.

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

[9] R. G. Downey and M. R. Fellows. *Fundamentals of Parameterized Complexity*. Springer, 2013.

[10] E. Elkind, P. Faliszewski, and A. Slinko. Swap bribery. In *Proceedings of the 2nd International Symposium on Algorithmic Game Theory (SAGT '15)*, volume 5814 of *LNCS*, pages 299–310. Springer, Oct. 2009.

[11] G. Erdélyi, M. Fellows, J. Rothe, and L. Schend. Control complexity in Bucklin and fallback voting: An experimental analysis. *Journal of Computer and System Sciences*, 81(4):661–670, 2015.

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

[13] P. Faliszewski, E. Hemaspaandra, and L. A. Hemaspaandra. Multimode control attacks on elections. *Journal of Artificial Intelligence Research*, 40:305–351, 2011.

[14] P. Faliszewski, E. Hemaspaandra, and L. A. Hemaspaandra. Weighted electoral control. In *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS '13)*, pages 367–374, 2013.

[15] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2006.

[16] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34(3):596–615, 1987.

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

[18] R. Kannan. Minkowski's convex body theorem and integer programming. *Mathematics of Operations Research*, 12(3):415–440, 1987.

[19] H. W. Lenstra Jr. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8(4):538–548, 1983.

[20] N. Mattei and T. Walsh. Preflib: A library of preference data. In *Proceedings of Third International Conference on Algorithmic Decision Theory (ADT '13)*, volume 8176 of *LNCS*, pages 259–270. Springer, 2013.

[21] T. Miąsko and P. Faliszewski. The complexity of priced control in elections. Available at http://home.agh.edu.pl/~faliszew/priced.pdf. Manuscript, 2014.

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

[23] I. Schlotter, P. Faliszewski, and E. Elkind. Campaign management under approval-driven voting rules. In *Proceedings of the Twenty-Fifth Conference on Artificial Intelligence (AAAI '11)*, pages 726–731, Aug. 2011.

[24] L. Xia. Computing the margin of victory for various voting rules. In *Proceedings of the 13th ACM Conference on Electronic Commerce (EC '12)*, pages 982–999, June 2012.