Mixed integer programming with convex/concave constraints: Fixed-parameter tractability and applications to multicovering and voting

Robert Bredereck\textsuperscript{a}, Piotr Faliszewski\textsuperscript{b}, Rolf Niedermeier\textsuperscript{a}, Piotr Skowron\textsuperscript{c,}\textsuperscript{*}, Nimrod Talmon\textsuperscript{d}

\textsuperscript{a} TU Berlin, Berlin, Germany
\textsuperscript{b} AGH University, Krakow, Poland
\textsuperscript{c} University of Warsaw, Warsaw, Poland
\textsuperscript{d} Ben-Gurion University, Be'er Sheva, Israel

1. Introduction

The idea of parameterized complexity theory is to measure the difficulty of computational problems with respect to both the length of their encodings (as in standard complexity theory) and additional parameters (e.g., pertaining to the structure of the input). For example, in the Set MULTICOVER problem we are given a set of elements \( U = \{x_1, \ldots, x_m\} \), a multiset \( S = \{S_1, \ldots, S_n\} \) of sets over \( U \), integer covering requirements \( r_1, \ldots, r_m \) for the elements of \( U \), and a budget \( B \). The question is whether it is possible to pick a collection of at most \( B \) sets from \( S \) so that each element \( x_i \in U \) belongs to at least \( r_i \) of them. This problem is a simple extension of the Set COVER problem and, thus, is NP-hard. There exists, however, an efficient algorithm for this problem when the cardinality of \( U \) is small. In fact, Set MULTICOVER is fixed-parameter tractable (is in FPT) with respect to the number of elements to cover: As we explain next, there is an algorithm that solves it in

\textsuperscript{*} Corresponding author.
E-mail address: p.skowron@mimuw.edu.pl (P. Skowron).

https://doi.org/10.1016/j.tcs.2020.01.017
0304-3975/© 2020 Elsevier B.V. All rights reserved.
$f(|U|) \cdot |I|^{O(1)}$ time, where $|I|$ is the length of the encoding of the given instance and $f$ is a computable function (which depends only on the parameter; i.e., $|U|$ in our case).

This FPT algorithm for Set Multicover proceeds by expressing the problem as an integer linear program (ILP) feasibility problem and solving it using the classic algorithm of Lenstra [44]. For each subset $A$ of $U$, let $S(A)$ be the subfamily of $S$ comprised of sets equal to $A$; use a variable $x_A$, intended to hold the number of sets from $S(A)$ that we include in the solution; and define a constant $b_A = |S(A)|$. Then, we introduce the following constraints:

\[
\begin{align*}
x_A & \leq b_A & \text{for each } A \subseteq U, \\
\sum_{A \subseteq U} x_A & \leq B, \\
\sum_{A \subseteq U : x_A \in A} x_A & \geq r_i & \text{for each } x_i \in U.
\end{align*}
\]

Constraints of the form (1) ensure that the solution is possible (i.e., we never use more sets of a given type than there are in the input), constraint (2) ensures that we use at most $B$ sets, and constraints of the form (3) ensure that each element from $U$ is covered a required number of times. We solve this ILP using the famous algorithm of Lenstra [44], which decides feasibility of ILPs in FPT time with respect to the number of integer variables. Since the number of integer variables in our ILP formulation is upper-bounded by $2^{|U|}$, this eventually yields fixed-parameter tractability with respect to the parameter $|U|$.

Unfortunately, the above approach seems to fail for the case of Weighted Set Multicover, a variant of Set Multicover where each set from $S$ also has a weight and we can only choose sets of total weight at most $B$ (Set Multicover is the special case of Weighted Set Multicover where all weights are equal to one). The reason for this apparent failure is that, for each $A \subseteq U$, the ILP for Set Multicover treats all sets from $S(A)$ as indistinguishable, but in Weighted Set Multicover they have weights, which give them “identities.” Specifically, for the case of Weighted Set Multicover we would have to replace constraint (2) with one of the form:

\[
\sum_{A \subseteq U} \text{weight}_A(x_A) \leq B,
\]

where $\text{weight}_A(x_A)$ is a (convex) function that gives the sum of the lowest $x_A$ weights of the sets from $S(A)$. The main contribution of this paper is in showing a simple technique that allows us to include constraints of this type, where some variables are replaced by their “piecewise linear convex or concave functions”, and still solve the resulting programs in FPT time using Lenstra’s algorithm as a black box.

In Section 1.1.1 we compare our technique to similar tools known in the literature. We argue that one of the greatest benefits of our technique is that it is very simple to use. Indeed, to obtain an FPT algorithm for Weighted Set Multicover it suffices to make only very small changes to the algorithm for Set Multicover and invoke our machinery. Second, our technique proceeds by constructing a mixed integer program from an integer linear program with aforementioned convex constraints—this makes it easy to feed it into existing commercial solvers and to obtain an efficient practical algorithm with a relatively low effort. Finally, we consider that the main contribution of our work is that we illustrate the simplicity and usefulness of the discussed technique by applying it to a number of set-covering problems and by resolving the computational complexity status of a number of election-related problems parameterized by the number of candidates. These problems include, for example, various bribery problems [18,20] and priced control problems [49] that were known to be in XP for nearly ten years, but were neither known to be fixed-parameter tractable, nor to be W[1]-hard. Indeed, in all these problems the voters had prices, which gave them “identities” in the same way as weights gave “identities” to sets in the Weighted Set Multicover problem. Nonetheless, our technique is not limited to the priced variants of election problems. For example, we show that it also applies to weighted voter control for Approval voting, improving results of Faliszewski et al. [22]. Further, Faliszewski et al. [24] and Izak et al. [33] recently used our technique to design algorithms for certain multiwinner election rules, and Fluschnik et al. [26] used it for a (voting-inspired) variant of the classic Knapsack problem (in addition, Peters [51] and Caragiannis et al. [10] used technical tricks that are very similar to our approach).

We also demonstrate the usefulness of our technique in more technically demanding scenarios. In particular, we consider the Multiset Multicover problem, which generalizes Set Multicover to the case of multisets, and we show that our general technique can be used as a component in the construction of an FPT-time approximation scheme for the problem, notably, parameterized by the number of elements, there is no hope for an FPT exact algorithm as the problem is NP-hard even for two elements. In this case, our analysis combines combinatorial arguments with the technique of handling ILPs with piecewise-linear convex/concave transformations.

1.1. Related work

Our work is related to three main research lines, namely regarding algorithms for integer linear programming, regarding various types of covering problems, and regarding algorithms for and complexity of voting-related problems. Below we review the main points of intersection with the corresponding literature.
1.1.1. Integer linear programming

Solving integer linear programs is one of the major approaches for tackling NP-hard problems [55]. Indeed, modern ILP solvers such as CPLEX or Gurobi can effectively deal with ILPs with thousands of variables and constraints; this makes them practical tools to solve even fairly large instances of computationally difficult problems. Yet, from our point of view, the most important aspect of integer linear programming is that it provides a powerful tool for designing FPT algorithms. The first major breakthrough in this direction was achieved by Lenstra [44], who showed that Mixed Integer Linear Programming is fixed-parameter tractable with respect to the number $n$ of the variables. Then Frank and Tardos [27] and Kannan [35] improved the corresponding upper bounds on running time.

The algorithm of Lenstra is very useful, but for some problems its parameterization by the number of integer variables seems insufficient to obtain FPT algorithms (the NP-hard problem Swap Bribery [18] parameterized by the number of candidates is one particular example where this seems to be the case). Fortunately, one may use other approaches, such as the recently popularized $n$-fold integer programming technique ($n$-fold IP). The main idea of $n$-fold integer programming is as follows: Our goal is to find a feasible solution of an inequality of the form $E \cdot x \leq b$, where $E$ is a matrix, $b$ is a vector of constants, and $x$ is a vector of integer variables (all as in the classic ILP problem), but where $E$ is restricted to be of a very special form ($A$ and $D$ are matrices):

$$
E = \begin{pmatrix}
D & D & \ldots & D \\
A & 0 & \ldots & 0 \\
0 & A & \ldots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \ldots & A
\end{pmatrix}
$$

There is an algorithm that solves integer programs (IPs) of this form in FPT time parameterized by the absolute value of the largest of their entries in combination with the dimensions of the matrices $A$ and $D$ [30,45,46] without restricting the dimensions of matrix $E$ (i.e., the constraints have a restricted form, but we can use as many integer variables as we like). Despite of this restrictive structure, $n$-fold integer programming found a number of applications, e.g., in scheduling [38] and voting [39]. See also the works of Dvořák et al. [16], Knop et al. [40], Eisenbrand et al. [17], and Koutecký et al. [43] for very recent generalizations, extensions and advancements concerning this technique.

Yet, despite of its recent successes, the $n$-fold IP technique is incomparable to our approach. On the one hand, using $n$-fold IP it is possible to find FPT algorithms for problems for which our approach seems not to be applicable (such as Swap Bribery parameterized by the number of candidates [39]). On the other hand, there are cases where using our approach it is very easy to provide an FPT algorithm, but where the $n$-fold IP technique appears to fail (e.g., the uniform Multiset Multicover problem parameterized by the size of the universe; see Section 3). While we are convinced that our approach is easier to use than that of $n$-fold integer programming, we also point out that Knop et al. [39,40] made significant progress in making $n$-fold IP more applicable. In particular, they described how to implement a number of primitives, and showed how using them makes the task of building $n$-fold programs much easier compared to directly designing the corresponding $A$ and $D$ matrices.

Further, we note that more general techniques than the one presented in our paper are known in the literature.\(^1\) For instance, Dadush et al. [12] proved that an IP can be solved in FPT time with respect to the number of integer variables even if the constraints describe an arbitrary convex polyhedron and are given through a separation oracle. One particular aspect in which the result of Dadush et al. [12] generalizes our technique is that it allows for using convex multi-variate functions in the constraints formulations. A similar argument has also been given by Hildebrand and Köppe [31]—for the case when constraints are expressed as convex polynomials, and by Khachiyan and Porkolab [37]—for the case when the constraints are expressed as convex semialgebraic sets.\(^2\) Most related to our technique is the work of Oertel et al. [50] showing that convex integer programming with a fixed number of variables can be solved with a number of calls to a mixed ILP oracle that is polynomial in $\log B$, where $B$ is a known upper bound on the values of the integer variables. Gavenčiak et al. [28] give a comprehensive review of the advances in solving convex integer programs from the last two decades.

We believe that our approach has two advantages that make it preferable over those mentioned above (whenever it can be used). First, the technique explained by our proof is simpler, uses only elementary methods, can be easily understood without the necessity of learning quite advanced tools, and it is straightforward to verify whether it was applied correctly. Second, our technique reduces an IP with (restricted) convex constraints to one single standard mixed ILP (the technique of

---

1 We thank Martin Koutecký for pointing us to the most relevant works in this area.

2 Some of these results are not commonly known among the community centered around parameterized complexity theory and are often stated in a “non-parameterized” language. For instance, Khachiyan and Porkolab only claim to show a polynomial time algorithm for solving IPs provided that the number of integer variables is constant (which in the language of parameterized complexity theory could simply be understood as an XP algorithm) but, in fact, their algorithm achieves FPT running time. Again, we thank Martin Koutecký for fruitful discussions on clarifying these issues.
Oertel et al. [50] allows more general constraints but requires multiple calls to an ILP oracle within the algorithm.) Hence, our approach allows to use cutting edge off-the-shelf MIP solvers with piecewise linear convex or concave transformations in the constraints. Thus, due to our results one may obtain both a theoretical FPT guarantee and a practical algorithm.

### 1.1.2. Covering

The class of covering problems is of fundamental importance in theoretical computer science because, on the one hand, covering problems are abstractions of many real-life issues, and, on the other hand, they exhibit very interesting algorithmic behavior. For example, the Set Cover problem, arguably the best known representative of the class, was among the first problems shown to be NP-complete (in Karp’s seminal paper [36]), and later it was thoroughly studied from the points of view of (in)approximability [59] and parameterized complexity [15]. For example, it is known to be W[2]-hard for the parameterization by the solution size (indeed, it is one of the classic W[2]-hard problems), but it is fixed-parameter tractable with respect to the number of elements. Covering problems have various applications in domains such as software engineering (e.g., covering scenarios by few test cases), antivirus development (looking for a set of suspicious byte strings which covers all known viruses), databases (finding a set of labels which covers all data items), to name just a few.

There is a vast literature on the Set Cover problem and, thus, we only briefly point out selected results. It is known that a simple greedy algorithm gives a log(m) approximation guarantee, where m is the size of the set to be covered (this algorithm was given, e.g., by Johnson [34], but see the textbook of Vazirani [58] for further references). It is also known that unless \( P = NP \), no polynomial-time algorithm can approximate the problem with a better ratio [13,25]. The variant of the problem where each element appears in at most \( f \) sets can be approximated with the ratio \( f \) [58]. Parameterized approximation algorithms for the problem were considered by Bonnet et al. [5], Skowron and Faliszewski [57], and Skowron [56].

Set Multicover and Multiset Multicover also received some attention in the literature. For example, Berman et al. [4] considered polynomial-time approximability of the former problem and Rajagopalan and Vazirani [52] studied the same issue for the latter (and, in general, for various covering problems). Exact algorithms for these problems were studied by Hua et al. [32]. Approximation algorithms for covering integer programs were considered by Kolliopoulos [41] and Kolliopoulos and Young [42].

### 1.1.3. Voting

Computational social choice (COMSOC) is an interdisciplinary area that spans computer science, economics, and operations research, and whose goal is to (computationally) analyze group decision-making processes [6,53]. From the paper’s point of view, the most relevant part of COMSOC regards the complexity of various election-related problems. We mostly focus on the problems of manipulating election outcomes through bribery and control [19].

In the simplest variant of the bribery problem [20], we are given a description of an election (i.e., a set of candidates and a collection of voters, with their votes represented in some way) and an integer \( k \). We ask if it is possible to ensure that a given candidate becomes a winner by bribing at most some \( k \) voters (i.e., by changing the votes of at most \( k \) voters). However, there are many other variants where, for example, each voter may have a different price for being bribed [20], the prices may depend on the extent to which we change given votes [18], or where the votes are represented using some involved language [47]. Initially, bribery problems were supposed to model undesirable behavior, but later researchers realized that they also capture perfectly legal actions, such as campaigning [3,9,14,54], fraud detection [60], or analysis of candidate performance [23]. Control problems are similar in spirit to the bribery ones, but instead of modifying votes, we are allowed to add/delete either candidates or voters [2,29].

We focus on the case where we have a few candidates but (possibly) many voters. 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. 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 [11,20,22]; these results typically follow by reductions from the well-known NP-hard Partition problem. One particular example where we did not have NP-hardness for the setting with the fixed number of candidates was control by adding/deleting voters under the Approval and \( k \)-Approval voting rules. For this problem Faliszewski et al. [22] have shown XP membership, but could neither prove W[1]-hardness nor give an FPT-algorithm; in this paper we resolve this problem by proving fixed-parameter tractability.

If we consider parameterization by the number of candidates but 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 then applying Lenstra’s algorithm [44]. In essence, these results are of the same nature as the FPT algorithm for the Set Multicover problem given at the beginning of the introduction. 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.3

---

3 Since the 2015 conference version of our paper, Knop et al. [39,40] have shown asymptotically faster algorithms for many—but not all—of the problems that we study. We keep the narrative of our conference work, where we have shown the first FPT algorithms for our problems, but in each case we mention if these results were already strengthened using the \( n \)-fold IP approach or whether doing so would be problematic.
1.2. Organization

Our paper is organized as follows. First, in Section 2, we describe our technique of handling ILPs with piecewise-linear convex/concave transformations. Then, in Section 3, we show examples of applying it to covering problems and to problems regarding Approval voting (which are very close in spirit to covering problems). In Section 4 we show how further election-related problems, beyond Approval voting, can be solved using our approach, and we conclude in Section 5.

2. Mixed integer linear programming with piecewise linear convex/concave functions

We now describe our technique of solving integer linear programs with piecewise linear convex/concave transformations. The technique is based on using rational-valued variables to simulate the behavior of convex/concave transformations and it uses as a subroutine an algorithm for solving the mixed integer linear programming problem, as defined below.

Mixed Integer Linear Programming (MIP)

**Input:** An $m \times (n + r)$ matrix $A$ with integer elements and a length-$m$ integer vector $b$ of rational numbers.

**Question:** Is there a length-$(n + r)$ vector $x = (x_1, \ldots, x_{n+r})$ such that $A \cdot x \leq b$, where the values $x_1, \ldots, x_n$ are required to be integers, but the values $x_{n+1}, \ldots, x_{n+r}$ can be rational?

We use standard terminology and syntax from linear programming. In particular, we interpret the entries of vector $x$ as variables and the rows of matrix $A$ as constraints. Formally, we require matrix $A$ and vector $b$ to have integer entries, but it would be straightforward to allow them to have rational values (indeed, we implicitly assume that whenever we obtain a matrix $A$ or a vector $b$ with rational values, then they are transformed to the integer form by appropriate multiplication).

Formally, the MIP problem captures the issue of testing whether some feasible solution exists for a given mixed integer program. If we want a solution that maximizes a certain objective function of the form $c_1x_1 + c_2x_2 + \cdots + c_{n+r}x_{n+r}$ (where $c_1, \ldots, c_{n+r}$ are integers), then we can use the standard trick of including the constraint

$$c_1x_1 + c_2x_2 + \cdots + c_{n+r}x_{n+r} \geq T$$

into the program and performing a binary search for the largest value of $T$ for which a feasible solution exists (provided the problem is bounded, as is the case for all the programs that we develop in the following sections\(^4\)). If $M$ is the value of the objective function for the optimal solution, then this requires solving $2\log M$ programs instead of one (we first find an upper bound on $M$ by increasing $T$ in exponentially many steps until we find the value for which the ILP is infeasible, and then we perform a binary search using the previously found upper bound). Typically, this is an acceptable price to pay.

The following result, due to Frank and Tardos [27] and Kannan [35], gives the FPT algorithm for the MIP problem parameterized by the number of integer variables (the result is built on top of Lenstra’s original algorithm [44]).

**Theorem 1** (Lenstra [44], Frank and Tardos [27], and Kannan [35]). There is an algorithm that solves mixed integer linear programming in $O(n^2.5n^{1+\Theta(1)} \cdot |I|)$ time, where $|I|$ is the number of bits encoding the input and $n$ is the number of integer variables.

Below we describe how the MIP problem and the above theorem can be extended to the case where each variable is replaced by some piecewise linear convex/concave transformation, while still maintaining the FPT (with respect to the number $n$ of integer variables) running time.

Piecewise Linear Convex/Concave Transformations. We consider two simple types of piecewise linear transformations, piecewise linear convex functions and piecewise linear concave functions. Some function $f : \mathbb{R} \to \mathbb{R}$ is called convex if: \(\forall x_1, x_2 \in \mathbb{R}, \forall t \in [0, 1]: f(tx_1 + (1-t)x_2) \leq tf(x_1) + (1-t)f(x_2)\). A function $g : \mathbb{R} \to \mathbb{R}$ is called concave if the function $-g$ is convex. A piecewise linear convex function is a continuous convex function, defined on a finite sequence of intervals (that together give the set of all real numbers) so that for each of the intervals, the function restricted to this interval is linear. Piecewise linear concave functions are defined analogously, except that they are concave instead of convex. We present examples of functions of this type in Fig. 1.

For a piecewise linear convex/concave function $f : \mathbb{R} \to \mathbb{R}$, we denote the decomposition of its domain into a minimal number of disjoint intervals on which it is linear as follows:

$$\mathbb{R} = (-\infty, \rho(f, 1)] \cup (\rho(f, 1), \rho(f, 2)] \cup (\rho(f, 2), \rho(f, 3)] \cup \ldots \cup (\rho(f, \ell), \infty)$$.

\(^4\) A linear program is bounded with respect to the objective function $c^\top x$ if there exists some bound $T$ such that $c^\top x \leq T$ for all vectors $x$. If it is not clear that a given MIP problem is bounded, then one can use a classic result of Meyer [48]: If our problem with integrality constraints lifted (i.e., the LP relaxation of our problem) is unbounded (which can be checked using standard methods), then the original problem is either infeasible or unbounded. Since we can check the feasibility of our original problem, this way we can also check whether it is unbounded. (If the LP relaxation is bounded, then the original problem is bounded as well, of course.) We stress that this approach works only if matrix $A$ has rational values only, as is the case in our MIP problems.

---

**Please cite this article in press as:** R. Bredereck et al., Mixed integer programming with convex/concave constraints: Fixed-parameter tractability and applications to multicovering and voting, Theoret. Comput. Sci. (2020), https://doi.org/10.1016/j.tcs.2020.01.017
We say that such a function $f$ consists of $\ell + 1$ pieces and we define $\text{pieces}(f) := \ell + 1$. We refer to the function $f$ restricted to interval $(-\infty, \rho(f, 1)]$ as the zeroth piece of $f$, to the function $f$ restricted to interval $[\rho(f, 1), \rho(f, 2)]$ as to the first piece of $f$, to the function $f$ restricted to interval $[\rho(f, 2), \rho(f, 3)]$ as to the second piece of $f$, and so on. By $\text{der}(f, i)$ we denote the (constant) derivative of the $i$-th piece of $f$. The derivative of a convex (resp. concave) function is monotonically non-decreasing (resp. non-increasing).

For technical reasons we also require that for each piecewise linear convex/concave function we have that (a) $f(0)$ is an integer, (b) for each piece $i$, $0 \leq i \leq \text{pieces}(f)$, the derivative $\text{der}(f, i)$ is an integer, and (c) for each $i$, $1 \leq i \leq \text{pieces}(f) - 1$, $\rho(f, i)$ is an integer. These requirements are technical only and, for example, we could replace all the occurrences of the word “integer” with “rational value” and all our results would still hold (this is analogous to the fact that we could define the MIP problem to work with rational values in the matrix $A$ and vector $b$).

**Mixed Integer Programs with Simple Linear Convex/Concave Transformations.** The following problem captures our extension of the Mixed Integer Linear Programming problem and will be our central technical tool in the following sections.

**Mixed Integer Programming with Simple Piecewise Linear Transformations (MIP/SPLiT)**

**Input:** A collection of $(n + r)m$ piecewise linear convex functions $F = \{f_{i, j}: 1 \leq i \leq (n + r), 1 \leq j \leq m\}$, a collection of $(n + r)m$ piecewise linear concave functions $G = \{g_{i, j}: 1 \leq i \leq (n + r), 1 \leq j \leq m\}$, and a vector $b \in \mathbb{Z}^m$.

**Question:** Is there a vector $x$ such that

\[
\begin{align*}
\sum_{i=1}^{n+r} f_{i, 1}(x_i) &\leq \sum_{i=1}^{n+r} g_{i, 1}(x_i) + b_1, \\
\sum_{i=1}^{n+r} f_{i, 2}(x_i) &\leq \sum_{i=1}^{n+r} g_{i, 2}(x_i) + b_2, \\
&\vdots \\
\sum_{i=1}^{n+r} f_{i, m}(x_i) &\leq \sum_{i=1}^{n+r} g_{i, m}(x_i) + b_m.
\end{align*}
\]

$x_i \in \mathbb{N}$ for $1 \leq i \leq n$,

$x_i \in \mathbb{R}^+$ for $n + 1 \leq i \leq n + r$?

Below we describe how to use Theorem 1 to solve MIP/SPLiT with up to polynomial factors in the same asymptotic time complexity as MIP.

**Theorem 2.** There is an algorithm that solves Mixed Integer Programming with Simple Piecewise Linear Transformations in $O(n2.5n + o(n) \cdot (|I| + p_{\text{max}})O(1))$ time, where $n$ is the number of integer variables, $p_{\text{max}}$ is the maximum number of pieces per function, and $|I|$ is the number of bits encoding the input.

**Proof.** To prove the theorem, we will reduce MIP/SPLiT to MIP by replacing each non-linear constraint (containing some non-linear $f_{i, j}$ or $g_{i, j}$) with a polynomial number of linear ones, using polynomially many additional real-valued variables (but without changing the number of integer ones). This will allow us to invoke Theorem 1 to solve the resulting program.

Note that in the MIP/SPLiT problem (as in the MIP one), we consider the canonical form, where all variables are non-negative. Hence, we can assume that the zeroth piece of each function $f_{i, j}$ and each function $g_{i, j}$ includes point 0 (pieces covering only negative points are irrelevant). Furthermore, by appropriately setting the $b_j$ coefficients, we can also assume that for each $i$ and $j$ we have $f_{i, j}(0) = 0$ and $g_{i, j}(0) = 0$.
Our reduction starts with the input MIP/SPLiT instance and successively transforms it into an ordinary MIP instance. We keep all integer variables $x_1, \ldots, x_n$ and all real-valued variables $x_{n+1}, \ldots, x_{n+r}$ of the original MIP/SPLiT instance, but we introduce additional real-valued variables and we replace non-linear constraints with a number of linear ones.

Replacing a Non-Linear Constraint. Let $j$ be an integer such that the non-linear constraint

$$\sum_{i=1}^{n+r} f_{i,j}(x_i) \leq \sum_{i=1}^{n+r} g_{i,j}(x_i) + b_j$$

(4)

has not yet been replaced with linear ones. We remove it from the program and in its place we include constraint (where $w_{i,j}$ and $u_{i,j}$ are new real-valued variables):

$$\sum_{i=1}^{n+r} w_{i,j} \leq \sum_{i=1}^{n+r} u_{i,j} + b_j.$$ 

(5)

We will include additional constraints so that variables $w_{i,j}$ will upper-bound values $f_{i,j}(x_i)$ and variables $u_{i,j}$ will lower-bound values $g_{i,j}(x_i)$ (and, indeed, we will be able to assume that these variables have exactly the values $f_{i,j}(x_i)$ and $g_{i,j}(x_i)$, respectively).

Additional Variables. To ensure that variables $w_{i,j}$ and $u_{i,j}$ have correct values (in the feasible solution for our program), we need additional variables. For each $i \in [n+r]$, we introduce pieces($g_{i,j}$) real-valued variables $y_{i,j,1}, \ldots, y_{i,j,\text{pieces}(g_{i,j})}$ and pieces($f_{i,j}$) real-valued variables $z_{i,j,1,\ldots, z_{i,j,\text{pieces}(f_{i,j})}}$. We will ensure that if there is a feasible solution to our program, then there is one where $y_{i,j,\ell} = \max(0, x_i - \rho(g_{i,j}, \ell))$ and $z_{i,j,\ell} = \max(0, x_i - \rho(f_{i,j}, \ell))$. In other words, for each $\ell$, the variable $y_{i,j,\ell}$ (resp. $z_{i,j,\ell}$) measures how far the variable $x_i$ is beyond the beginning of the $\ell$-th piece of function $g_{i,j}$ (resp. of function $f_{i,j}$).

Constraining Variables $w_{i,j}$ (Convex Case). First, for each variable $z_{i,j,\ell}$ we introduce two constraints:

$$z_{i,j,\ell} \geq 0,$$

(6)

$$z_{i,j,\ell} \geq x_i - \rho(f_{i,j}, \ell).$$

Second, for each $i \in [n+r]$ we introduce the constraint:

$$x_i \cdot \text{der}(f_{i,j}, 0) + \sum_{\ell=1}^{\text{pieces}(f_{i,j})} z_{i,j,\ell} \cdot \left(\text{der}(f_{i,j}, \ell) - \text{der}(f_{i,j}, \ell - 1)\right) \leq w_{i,j}.$$ 

(7)

Constraining Variables $u_{i,j}$ (Concave Case). Analogously to the convex case, we first introduce two constraints for each variable $y_{i,j,\ell}$ (these two constraints are almost identical to the convex case):

$$y_{i,j,\ell} \geq 0,$$

(8)

$$y_{i,j,\ell} \geq x_i - \rho(g_{i,j}, \ell).$$

Second, for each $i \in [n+r]$ we introduce the constraint:

$$u_{i,j} \leq x_i \cdot \text{der}(g_{i,j}, 0) + \sum_{\ell=1}^{\text{pieces}(g_{i,j})} y_{i,j,\ell} \cdot \left(\text{der}(g_{i,j}, \ell) - \text{der}(g_{i,j}, \ell - 1)\right).$$

(9)

Correctness. We now argue that if there is a feasible solution for our transformed MIP instance, then there also is one for the original MIP/SPLiT instance. Let us assume some feasible solution for the transformed instance and focus on an arbitrary constraint number $j$ (recall Equations (4) and (5)).

In order to satisfy Constraint (5), smaller values of $w_{i,j}$ are clearly more desirable. Since each $w_{i,j}$ only occurs once on the left-hand side in Constraint (5) and once on the right-hand side in one of the constraints of the form (7), we can assume that each constraint of the form (7) is satisfied with equality (given a feasible solution, we can keep on decreasing the values $w_{i,j}$ until we hit equalities in these constraints). Further, together with the fact that $f_{i,j}$ is convex, and consequently $\text{der}(f_{i,j}, \ell) > \text{der}(f_{i,j}, \ell - 1)$ for each $\ell$, we infer that the values $z_{i,j,\ell}$ can be as small as possible. Formally, similarly as above, using constraints of the form (6), we can assume that in our feasible solution for each variable $z_{i,j,\ell}$ it holds that $z_{i,j,\ell} = \max(0, x_i - \rho(f_{i,j}, \ell))$. Consequently, we conclude that there is a feasible solution where:
Let us now analyze the value \( w_{i,j} \) provided by this equality. If \( x_i = 0 \), then we surely have \( w_{i,j} = 0 = f_{i,j}(0) \). Next, we analyze how the value of \( w_{i,j} \) changes when we increase \( x_i \) to \( \Delta \). If \( x_i \) does not exceed \( \rho(f_{i,j}, 0) \), then \( w_{i,j} \) has value \( \Delta \cdot \frac{\partial}{\partial x_i} f_{i,j}(0) \) and we have \( w_{i,j} = f(\Delta) \). If \( x_i \) is greater than \( \rho(f_{i,j}, 0) \) but smaller than \( \rho(f_{i,j}, 1) \), then \( w_{i,j} \) has value \( \Delta \cdot \frac{\partial}{\partial x_i} f_{i,j}(0) + (\Delta - \rho(f_{i,j}, 0)) \cdot (\frac{\partial}{\partial x_i} f_{i,j}(1) - \frac{\partial}{\partial x_i} f_{i,j}(0)) = f_i(\Delta) \). By analogous reasoning, we obtain \( w_{i,j} = f_{i,j}(x_i) \) for every value of \( x_i \).

Analogous reasoning shows that we can also assume that under our feasible solution it holds that \( u_{i,j} = g_{i,j}(x_i) \). Specifically, we note that to satisfy Constraint (5), larger values of \( u_{i,j} \) are clearly more desirable. Since each \( u_{i,j} \) only occurs once on the right-hand side in Constraint (5) and once on the left-hand side in one constraint of the form (9), we infer that each constraint of the form (9) can be satisfied with equality. Further, together with the fact that \( g_{i,j} \) is concave, and consequently \( \frac{\partial}{\partial x_i} g_{i,j}(x_i) < \frac{\partial}{\partial x_i} g_{i,j}(x_i - 1) \) for each \( x_i \), we infer that the values \( u_{i,j} \) can be as small as possible. Formally, similarly as above, we infer from constraints of the form (8) that we can assume that in our feasible solution for each variable \( y_{i,j,\ell} \) it holds that \( y_{i,j,\ell} = \max(0, x_i - \rho(g_{i,j}, \ell)) \). Consequently, we have that:

\[
w_{i,j} = x_i \cdot \frac{\partial}{\partial x_i} g_{i,j}(0) + \sum_{\ell=1}^{\text{pieces}(g_{i,j})} \max(0, x_i - \rho(g_{i,j}, \ell)) \cdot (\frac{\partial}{\partial x_i} g_{i,j}(\ell) - \frac{\partial}{\partial x_i} g_{i,j}(\ell - 1)).
\]

Analysis analogous to that for the case of variables \( w_{i,j} \) shows that, indeed, we have \( u_{i,j} = g_{i,j}(x_i) \).

The above reasoning, together with the fact that constraints of the form (5) are satisfied and clearly correspond to the original constraints in the MIP/SPLIT problem, proves that if there is a feasible solution for the transformed instance, then there also is one for the original instance. From our reasoning it is also apparent that the other implication holds (given variables \( x_i \) it suffices that for each \( i, j, \) and \( \ell \) we set \( w_{i,j} = f_{i,j}(x_i) \), \( u_{i,j} = g_{i,j}(x_i) \), \( z_{i,j,\ell} = \max(0, x_i - \rho(f_{i,j}, \ell)) \), and \( y_{i,j,\ell} = \max(0, x_i - \rho(g_{i,j}, \ell)) \).

**Running Time.** The running-time of the algorithm from Theorem 1 invoked on our transformed instance is upper-bounded by \( n^{2.5 + \Theta(n)} \cdot |I|^O(1) \), where \( n \) denotes the number of integer variables and \( |I| \) is the number of bits needed to encode our MIP. Finally, \( |I|^O(1) \) can be upper-bounded by \( (|I| + p_{\max})^O(1) \) since we introduced at most \( O(p_{\max} \cdot (n + r)) \) additional constraints and variables. This completes the proof. \( \square \)

We conclude this section with two observations regarding the generality of Theorem 2. First, note that in this section we used the canonical form of MIP/SPLIT, requiring all variables to be non-negative. Yet, as long as we do not actually use the piecewise linear transformations on a variable \( x_i \) (that is, as long as for each \( j \), functions \( f_{i,j} \) and \( g_{i,j} \) are linear) we can use the standard technique of replacing each occurrence of \( x_i \) with \( x_i^+ - x_i^- \), where \( x_i^+ \) and \( x_i^- \) are two nonnegative variables denoting, respectively, the positive and the negative part of \( x_i \). In other words, we may allow negative values for each variable \( x_i \) whose associated functions \( f_{i,j} \) and \( g_{i,j} \) are simply linear.

Second, note that if a certain function \( f_{i,j} \) is applied to an integer variable \( x_i \) and we know that the value of \( x_i \) must come from some set \( \{0, \ldots, t_i\} \), then it suffices that \( f_{i,j} \) is convex/concave on integer arguments, and we do not have to worry about it being piecewise linear. This last condition is vacuously satisfied: It suffices to consider pieces \( [0, 1), [1, 2), [2, 3), \ldots, [t_i - 1, t_i + 1) \), and derivatives \( \frac{d}{dx} f(0) = f(1) - f(0) \), \( \frac{d}{dx} f(1) = f(2) - f(1) \), \( \frac{d}{dx} f(2) = f(3) - f(2) \), and so on. It turns out that in many applications we apply convex/concave functions to integer variables only and this observation comes handy.

### 3. Covering and approval voting: showcases of the technique

In this section we demonstrate how to apply Theorem 2 to resolve the complexity of a number of problems related to covering and approval voting. These are interesting because the complexity of some of the problems we consider was open for several years. We also show that Theorem 2 is useful in designing an FPT approximation scheme for the Multisplit Multicover problem. This result is interesting because it is somewhat involved technically and illustrates mixing of our MIP/SPLIT-based approach with combinatorial arguments.

#### 3.1. Weighted multiset multicover with small universe

We start by focusing on the complexity of a few generalizations of the Set Cover problem. 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 1.** In the Weighted Multiset Multicover (WMM) problem we are given a family \( S = \{S_1, \ldots, S_n\} \) of multisets over the universe \( U = \{x_1, \ldots, x_m\} \), integer weights \( w_1, \ldots, w_n \) for the multisets, integer covering requirements \( r_1, \ldots, r_m \)
for the elements of the universe, and an integer budget \( B \). We ask whether there exists a subfamily \( S' \subseteq S \) of multisets from \( S \) such that:

1. for each \( x_i \in U \) it holds that \( \sum_{j \in S'} S_j(x_i) \geq r_i \) (that is, each element \( x_i \) is covered at least the required number of times), and
2. \( \sum_{j \in S'} w_j \leq B \) (the budget is not exceeded).

We will show how the complexity of WMM (parameterized with respect to the universe size) changes as we keep on adding restrictions. We observe that a straightforward polynomial-time reduction from Partition proves that WMM is NP-hard even for the case of a single-element universe. Clearly, this also means that the problem is not fixed-parameter tractable with respect to the number of elements in the universe as the parameter.

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

**Proof.** Membership in NP is clear. We show NP-hardness by a reduction from the Partition problem. An instance of Partition consists of a sequence of nonnegative integers \( k_1, \ldots, k_n \). We ask if there is a set \( I \subseteq [n] \) such that \( \sum_{i \in I} k_i = \frac{1}{2} \sum_{i=1}^n k_i \).

We form an instance of Weighted Multiset Multicover as follows. The universe contains a single element \( x \) with covering requirement equal to \( \frac{1}{2} \sum_{i=1}^n k_i \). For each \( i, 1 \leq i \leq n \), there is a single multiset \( S_i \) containing \( k_i \) occurrences of \( x \), with weight \( k_i \). We set the budget to be \( \frac{1}{2} \sum_{i=1}^n k_i \). Clearly, it is possible to cover \( x \) sufficiently many times if and only if our input instance of Partition is a “yes”-instance.

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, which again implies Para-NP-hardness with respect to the number of elements in the universe.

**Proposition 2.** Multiset Multicover is NP-complete even for universes of size two.

**Proof.** Membership in NP is clear. To show NP-hardness, we give a reduction from a variant of the Subset Sum problem. We are given a sequence \( k_1, \ldots, k_{2n} \) of positive integers, a target value \( T \), and we ask if there is a set \( I \subseteq \{1, \ldots, 2n\} \) such that \( \sum_{i \in I} k_i = T \), and \( |I| = n \).

Let \( K \) be \( \max(k_1, \ldots, k_{2n}) \). We form an instance of Multiset Multicover that contains two elements, \( x_1 \) and \( x_2 \). For each \( i, 1 \leq i \leq 2n \), we form a set \( S_i \) that contains \( x_1 \) with multiplicity \( k_i \), and \( x_2 \) with multiplicity \( nKT - k_i \). We set the covering requirement \( r_1 \) of \( x_1 \) to be \( T \), and the covering requirement \( r_2 \) of \( x_2 \) to be \( n^2KT - T \). We ask if there is a multiset multicover of size at most \( n \).

Clearly, if there is a solution for our Subset Sum instance, then the sets that correspond to this solution form a multiset multicover of \( x_1 \) and \( x_2 \). On the contrary, assume that there is a collection of at most \( n \) sets that form a multiset multicover of \( x_1 \) and \( x_2 \). There must be exactly \( n \) of these sets. Otherwise, the sum of their multiplicities for \( x_2 \) would be smaller than \( n^2KT - T \). Due to the covering requirement of \( x_1 \), these sets correspond to the numbers from \( \{x_1, \ldots, x_{2n}\} \) that sum up to at least \( T \), and due to covering requirement of \( x_2 \), these sets correspond to numbers that sum up to at most \( T \). This completes the proof.

To obtain an FPT algorithm parameterized by the size of the universe, we need to simplify WMM further. Weighted Set Multicover is a variant of WMM where each input multiset has elements with multiplicities \( 0 \) or \( 1 \) (in other words, the family \( S \) contains sets without multiplicities, but the union operation takes multiplicities into account). Uniform Multiset Multicover is a variant of Multiset Multicover (and, thus, of WMM), 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 multiplicities). We find these two problems interesting as in the next section we will show that several problems from computational social choice can be reduced to them.

As the first application of our new framework, we show that Weighted Set Multicover is fixed-parameter tractable when parameterized by the universe size. Notably, we only use convex constraints in the construction of the MIP/SPLiT instance.

**Theorem 3.** Weighted Set Multicover is fixed-parameter tractable when parameterized by the universe size.

**Proof.** Consider an instance of Weighted Set Multicover with universe \( U = \{x_1, \ldots, x_m\} \), family \( S = \{S_1, \ldots, S_n\} \) 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 MIP/SPLiT instance.
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 $S(U_i) := \{ S_j \in S \mid S_j = U_i \}$. For each $i$ and $j$, $1 \leq i \leq 2^m$, we define a convex function $f_i$ so that for each integer $j$, $1 \leq j \leq |S(U_i)|$, $f_i(j)$ is the sum of the $j$ lowest weights of the sets from $S(U_i)$. 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 (i.e., how many sets we take from each family $S(U_i)$). We introduce the following constraints: For each $i$, $1 \leq i \leq 2^m$, we have the constraints $z_i \geq 0$ and $z_i \in S(U_i)$. For each element $x_i$ of the universe, we also have the constraint $\sum_{U_i \in S_j} x_i \in U_i z_i \geq \ell_r$. These constraints ensure that the variables $z_i$ describe a possible solution for the problem (disregarding the budget). Our last constraint uses variables $z_i$ to express the requirement that the solution has cost at most $B$:

$$\sum_{i=1}^{2^m} f_i(z_i) \leq B.$$ 

Finally, we use Theorem 2 to get the statement of the theorem. \( \square \)

The above-described algorithm has running time that is doubly-exponential with respect to the universe size (specifically, the superpolynomial factor in the running time is of the form $2^{O(2^m \log m)}$; to see this, recall Theorem 2 and note that our algorithm creates $O(2^m)$ integer variables). This result was improved by Knop et al. [40], who achieved a single-exponential dependence by applying the $n$-fold IP technique (the superpolynomial factor of the running time of their algorithm is $2^{O(m^2 \log m)}$).

As a second application of our new framework, we show that Uniform Multiset Multicover is fixed-parameter tractable when parameterized by the universe size. Notably, we only use concave constraints in the corresponding MIP/SPLIT instance.

**Theorem 4.** Uniform Multiset Multicover is fixed-parameter tractable when parameterized by the universe size.

**Proof.** Consider an instance of Uniform Multiset Multicover with universe $U = \{x_1, \ldots, x_m\}$, family $S = \{S_1, \ldots, S_n\}$ of subsets, covering requirements $r_1, \ldots, r_m$ for the elements, and budget $B$. Our algorithm proceeds by solving an appropriate MIP/SPLIT instance.

Similarly as in the proof of Theorem 3, we form a family $U_1, \ldots, U_{2^m}$ of all the subsets of $U$ (note that these, indeed, are subsets and not multisets). For each $i$, $1 \leq i \leq 2^m$, let $S(U_i)$ be a subfamily of $5$ that contains those multisets in which exactly the elements from $U_i$ appear (that is, their multiplicities are non-zero). For each $i$, $1 \leq i \leq 2^m$, we define a concave function $f_i$ so that for each element $x_i \in U_i$, $1 \leq j \leq |S(U_i)|$, $f_i(j)$ denotes the maximum sum of multiplicities for each element from $U_i$, using $j$ multisets from $S(U_i)$ (To compute this function, we simply need to sort the multisets from $S(U_i)$ in the order of decreasing multiplicities. Then, $f_i(j)$ is the sum of the multiplicities with respect to an arbitrary element from $U_i$ of the first $j$ multisets.)

We have $2^m$ integer variables $z_i$, $1 \leq i \leq 2^m$. Intuitively, the variables $z_i$ describe how many multisets we take from each type. Thus, $f_i(z_i)$ describes how much each element from $U_i$ is covered by taking $z_i$ multisets of type $U_i$. We introduce the following constraints: For each $i$, $1 \leq i \leq 2^m$, we have constraints $z_i \geq 0$ and $z_i \in S(U_i)$. For each element $x_i$ of the universe, we also have the constraint $\sum_{U_i \in S_j} x_i \in U_i f_i(z_i) \geq \ell_r$. These constraints ensure that the variables $z_i$ describe a possible solution for the problem (disregarding the budget). To express the requirement that the solution has cost at most $B$, we add the constraint $\sum_{i=1}^{2^m} z_i \leq B$. Finally, we use Theorem 2 to get the statement of the theorem. \( \square \)

As in the case of Theorem 3, the superpolynomial factor in the running time of our algorithm is of the form $2^{O(2^m \log m)}$. Yet, interestingly, in this case it does not seem easy to improve it (or, in fact, even match it) using $n$-fold integer programming. The reason is, intuitively, one has to include the multiplicities of the sets into the constraints, and doing so pushes the running times of $n$-fold IP solvers beyond FPT time (they have the value of the largest coefficient in a constraint as the base of the exponent in their running times [30,40,45,46]).

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

**Definition 2.** Let $\epsilon$ be a real number, $\epsilon > 0$. We say that algorithm $A$ is an $\epsilon$-almost-cover algorithm for Multiset Multicover if, given a “yes”-instance $I$ of Multiset Multicover with budget $B$, universe $U = \{x_1, \ldots, x_m\}$, and covering requirements $r_1, \ldots, r_m$, it outputs a solution that uses at most $B$ sets and covers each element $x_i$ with multiplicity $r'_j$ such that

$$\sum_{i} \max(0, r_i - r'_i) < \epsilon \sum_{i} r_i.$$ 

In other words, on the average an $\epsilon$-almost-cover algorithm can miss each element of the universe by an $\epsilon$-fraction of its covering requirement. For the case where we really need to cover all the elements perfectly, we might first run an $\epsilon$-almost-cover algorithm and then complement its solution, for example, in some greedy way, since the remaining instance might be much easier to solve.
The key idea regarding computing an $\varepsilon$-almost-cover is that it suffices to replace each input multiset by several submultisets, each with a particular “precision level,” so that multiplicities of the elements in each sub-multiset are of a similar order of magnitude. The full argument, however, forms the most technical part of our paper.

**Theorem 5.** For every rational $\varepsilon > 0$, there is an FPT-time $\varepsilon$-almost-cover algorithm for MULTISET MULTICOVER parameterized by the universe size.

**Proof.** We describe our $\varepsilon$-almost-cover algorithm for MULTISET MULTICOVER and argue for its correctness. We consider a “yes”-instance $I$ of MULTISET MULTICOVER with a family $S = \{S_1, S_2, \ldots, S_\ell\}$ of multisets over the universe $U = \{x_1, x_2, \ldots, x_m\}$, where the covering requirements for the elements of the universe are $r_1, \ldots, r_m$, and the budget is $B$. Naturally, we cannot check if $I$ indeed is a “yes”-instance, but if it is not, then Definition 2 does not constrain the operation of our algorithm in any way. We associate each set $S$ from the family $S$ with the vector $v_S = \langle S(x_1), S(x_2), \ldots, S(x_m) \rangle$ of element multiplicities.

Let $x \in \mathcal{U}$ be the desired approximation ratio. We fix $Z = \lceil \frac{2m}{\varepsilon} \rceil$ and $Y = Z + \lceil \frac{2m^3}{\varepsilon} \rceil$. Note that $\frac{m}{Z} \leq \frac{\varepsilon}{4}$ and $\frac{2m^3}{\varepsilon Z} \leq \frac{\varepsilon}{4}$.

Let $X = \left( \frac{2m}{\varepsilon} + 1 \right)^m$ and let $V_1, \ldots, V_X$ be a sequence of all $m$-dimensional vectors whose entries come from the $\left( \frac{2m}{\varepsilon} + 1 \right)$-element set $\{0, \frac{\varepsilon}{2}, \frac{\varepsilon}{2} \cdot 2, \ldots, Y \}$. 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 set of “shapes” of all possible multisets—interpreted as vectors of multiplicities—over our $m$-element universe. For each number $\beta$, we write $\beta V_j$ to refer to the vector $\langle \beta v_j(x_1), \beta v_j(x_2), \ldots, \beta v_j(x_m) \rangle$.

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

The main step of our algorithm is to modify the instance $I$ so that we replace each multiset $S$ from the family $S$ with a sequence of vectors of the form $\beta V_j$ that altogether add to at most the multiset $S$ (each such sequence can contain multiple vectors of different “shapes” $V_j$ and of different scaling factors $\beta$). 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 $\sum_j \max(0, r_i - r'_i) < \varepsilon \sum r_i$. 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 $(\beta, V)$ for each vector $\beta V_j$ that 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 $S$, 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 29). We will come back to this issue later.

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 3 of Algorithm 1). Let $< \varepsilon$ be the order in which Algorithm 1 considers the elements (so $x_i < x_j$ means that $x_i$ is considered before $x_j$), and let $x_1', \ldots, x_m'$ be the elements from the universe renamed so that $x_i' < x_{j+1}' < \cdots < x_m'$. 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 Fig. 2, where for the sake of simplicity of presentation 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 emitted. Note that each element $x$ from the universe such that $x_i' < x$ has the same multiplicity in $S_k$ as element $x_i'$ (lines 19 and 20 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:** $S_k(x_i') = Z \cdot S_k(x_i' - 1)$.

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

**Observation 3:** We have that $S_{k+1}(x_i') \geq \frac{(Y - Z)}{Z} S_k(x_i') \geq \frac{(Y - Z)}{Z} t_k$. Further, we have that $S_{k+1}(x_i') \geq \frac{(Y - Z)}{Z m^3} \sum_j t_j$. (To see why the last inequality holds, note that $k \leq m$.)

**Observation 4:** For $i < i_k$ it holds that $\sum_{i \leq i_k} S_k(x_i') = S(x_i')$.

Now let us consider some solution for instance $I$ that consists of $K$ sets, $S = \{S_1, S_2, \ldots, S_K\} \subseteq \mathcal{S}$. These sets, altogether, cover all elements from the universe with required multiplicities. That is, it holds that for each $i$ we have
Algorithm 1: The transformation algorithm used in the proof of Theorem 5. The algorithm replaces a given set $S$ with a sequence of vectors of the form $\beta V_j$.

1. **Main $(S)$:**
   2. $\text{multip} \leftarrow \{(1, S(x_1)), (2, S(x_2)), \ldots, (m, S(x_m))\}$;
   3. $\text{sorted} \leftarrow \text{sort} (\text{multip})$;  
      \hspace{1cm} // sort in ascending order of multiplicities
   4. $i \leftarrow 0$;
      \hspace{1cm} // sorted[i].first refers to the i'th item's number
      \hspace{1cm} // sorted[i].second refers to its multiplicity
   5. while $\text{sorted}[i].second = 0$ do
      6. $i \leftarrow i + 1$;
   7. $\text{Main}_\text{Rec}(i, \text{sorted})$;

8. **Main$_\text{Rec}$(i, multip):**
   9. $V \leftarrow \{0, 0, \ldots, 0\}$ (vector of $m$ zeros);
   10. $\beta \leftarrow \text{multip}[i].second$;
   11. $V[\text{multip}[i].first] \leftarrow 1$;
   12. $i \leftarrow i + 1$;
   13. while $i \leq m$ do
      14. if $\text{multip}[i].second < Y \cdot \text{multip}[i-1].second$ then
      15. $V[\text{multip}[i].first] \leftarrow \frac{\text{multip}[i].second}{\beta}$;
      16. $i \leftarrow i + 1$;
   17. else
      18. for $j \leftarrow i$ to $m$ do
      19. \hspace{1cm} $V[\text{multip}[j].first] \leftarrow \frac{\text{multip}[j].second}{\beta}$;
      20. $\text{Round_And_Emit}(\beta, V)$;
    21. for $j \leftarrow 1$ to $m$ do
      22. \hspace{1cm} $\text{multip}[j].second \leftarrow \text{multip}[j].second - \beta V[\text{multip}[j].first]$;
      23. $\text{Main}_\text{Rec}(i, \text{multip})$;
    24. return
      25. $\text{Round_And_Emit}(\beta, V)$;

26. **Round_And_Emit(\beta, V):**
   27. for $\ell \leftarrow 1$ to $m$ do
      28. \hspace{1cm} $V[\ell] \leftarrow \frac{2^{\text{multip}[\ell].second}}{2}$;
      29. $\text{Emit}(\beta, V)$;
   30. $\text{return}$

31. **Round_And_Emit(\beta, V):**

28. **Round_And_Emit(\beta, V):**

Fig. 2. An example for Algorithm 1: The algorithm replaces $S$ with sets $S_1, S_2, S_3$. 

Please cite this article in press as: R. Bredereck et al., Mixed integer programming with convex/concave constraints: Fixed-parameter tractability and applications to multicoverying and voting, Theoret. Comput. Sci. (2020), https://doi.org/10.1016/j.tcs.2020.01.017
We intuitively, Case (a) defined the actually \( \sum_j J_i \). For each set \( S \in S^{opt} \) 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 \( S \in S^{opt} \) and every \( x_i \), \( y_{S,i} \leq S(x_i) \).
2. For every \( x_i \), \( \sum_{S \in S^{opt}} y_{S,i} = r_i \).

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 cover the elements. Based on these numbers, we will show how to replace each set from \( S^{opt} \) with one of the sets emitted for it, so that the resulting family of sets has the almost-cover approximation property.

Consider a set \( S \in S^{opt} \) 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 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 \( R = \{ S_1, S_2, \ldots, S_r \} \), let \( i_{max} = \arg \max_i y'_{S,i} \), and let \( S_{repl} \) be the set from \( R \) defined in the following way:

1. If for every set \( S_k \in R \) we have \( S_k(x'_{i_{max}}) < y'_{S,i_{max}} \), then \( S_{repl} \) is the set \( S_k \in R \) 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 Fig. 3.
2. Otherwise \( S_{repl} \) is the set \( S_k \in R \) 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 Fig. 3.

We now show that \( S_{repl} \) is a good candidate for replacing \( S \), that is, that

\[
\sum_i \max(0, y'_{S,i} - S_{repl}(x'_i)) < \epsilon \sum_i y'_{S,i}.
\]

To this end, we consider the three cases depicted in Fig. 3:

**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 \( \sum_i \max(0, y'_{S,i} - S_1(x'_i)) = 0 \). By the criterion for choosing set \( S_{repl} \), we have that \( S_{repl} = S_1 \).

**Case (b)** There exist sets \( S_{k-1}, S_k \in R \) such that \( S_k(x'_{i_{max}}) \geq y'_{S,i_{max}} > S_{k-1}(x'_{i_{max}}) \) (and thus, \( S_{repl} = S_k \)). Let \( x'_{i_{k-1}} = x'_{i_{k-1}} \) (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) \( y'_{S,i_{max}} \leq S_k(x'_i) \): 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 \( y'_{S,i} \leq \sum_{l \leq k-1} S_i(x'_i) \) (this follows from Observation 4 and the fact that \( y'_{S,i} \leq S(x'_i) \)). Based on this inequality, we get:

\[
\sum_{i<j} y'_{S,i} \leq \sum_{i<j} \sum_{l \leq k-1} S_i(x'_i) \leq \sum_{l \leq k-2} t_l + \sum_{i<j} S_{k-1}(x'_i)
\]

Please cite this article in press as: R. Bredereck et al., Mixed integer programming with convex/concave constraints: Fixed-parameter tractability and applications to multicovering and voting, Theoret. Comput. Sci. (2020), https://doi.org/10.1016/j.tcs.2020.01.017
\[ \begin{align*}
\leq & \frac{Z m^2}{(Y - Z)} S_{k-1}(x') + \frac{m}{Z} S_{k-1}(x') \\
\leq & \frac{\epsilon}{2} S_{k-1}(x') \leq \frac{\epsilon}{2} y_{S,\text{max}}.
\end{align*} \]

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

(ii) \( y'_{S,\text{max}} > S_k(x') \): For \( \ell \geq i_k \), we have \( S_k(x') \geq y'_{S,\ell} \)(this follows from the fact that \( S_k(x'_{\text{max}}) \geq y'_{S,\text{max}} \) and the definition of \( i_k \)). For \( \ell < i_k \), by Observation 4 and the fact that \( y_{S,\ell} \leq S(x') \), we have \( y'_{S,\ell} - S_k(x') \leq \sum q \leq k S_q(x') \). Thus we get:

\[ \begin{align*}
\sum \max(0, y'_{S,\ell} - S_k(x')) & \leq \sum \max(0, y'_{S,\ell} - S_k(x')) \\
& \leq \sum_{\ell < i_k} \sum q \leq k-1 \leq \sum \tau_q \\
& \leq \frac{Z m^2}{(Y - Z)} S_k(x') \\
& \leq \frac{Z m^2}{(Y - Z)} y'_{S,\text{max}} \leq \frac{\epsilon}{2} \sum \ell y_{S,\ell}.
\end{align*} \]

Case (c) Every set \( S_k \in \mathcal{R} \) has \( S_k(x'_{\text{max}}) < y'_{S,\text{max}} \).

By the choice of \( S_k \) (the set from \( \mathcal{R} \) that has highest multiplicity of \( x'_{\text{max}} \)), we infer that \( \sum q \leq k S_q(x'_{\text{max}}) = S(x'_{\text{max}}) \). Also, for every \( \ell < i_{\text{max}} \), we have \( \sum q \leq k S_q(x') = S(x') \). Consequently, for every \( \ell \leq i_{\text{max}} \) we have

\[ y_{S,\ell} - S_k(x') \leq \sum q \leq k S_q(x') \leq \sum q \leq k S_q(x'_{\text{max}}). \]

Further, for every \( \ell > i_{\text{max}} \), we have

\[ y'_{S,\ell} - S_k(x') \leq y'_{S,\text{max}} - S_k(x'_{\text{max}}) \leq S(x'_{\text{max}}) - S_k(x'_{\text{max}}) \leq \sum q \leq k S_q(x'_{\text{max}}). \]

Based on these observations, we get the following:

\[ \begin{align*}
\sum \max(0, y_{S,\ell} - S_k(x')) & \leq m \sum q \leq k-1 S_q(x'_{\text{max}}) \leq m \sum \tau_q \\
& \leq m \frac{Z m^2}{(Y - Z)} S_k(x') \\
& \leq m \frac{Z m^3}{(Y - Z)} y'_{S,\text{max}} \leq \frac{\epsilon}{2} \sum \ell y_{S,\ell}.
\end{align*} \]

Thus we obtain the desired bound.

The above case analysis almost shows that we indeed have the almost-cover approximation property. It remains to consider the issue of rounding (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 \( S' \) of at most \( K \) sets that form an \( \epsilon \)-almost-cover). It is possible to do so using our technique from Section 2.

Let us recall that the new instance consists of the sets that are of the form \( \beta V_j \) (recall the discussion at the beginning of the proof). For each vector \( V_j, \ 1 \leq j \leq X \), we introduce an integer variable \( v_j \), which, intuitively, gives the number of sets with shape \( V_j \) taken into the solution. Further, for each \( v_j, \ 1 \leq j \leq X \), and each \( x_i, \ 1 \leq i \leq m \), we introduce a concave function \( f_{i,j} \) so that \( f_{i,j}(v_j) \) is the maximum multiplicity with which element \( x_i \) is covered by some \( v_j \) “best” sets of the shape \( V_j \) (these are the \( v_j \) sets which have been emitted with the highest values of \( \beta \)). Finally, we introduce variables \( \text{miss}_1, \ldots, \text{miss}_m \), responsible for measuring the inaccuracy levels (in other words, \( \text{miss}_i \) gives the missing multiplicity for element \( x_i \)). The constraints for our mixed integer linear program are given below:

1. \( \sum_{j=1}^{X} v_j \leq B \).
2. For each \( j, 1 \leq j \leq X: v_j \geq 0 \).
3. \( \sum_{i=1}^{m} \text{miss}_i \leq \epsilon \sum_{i} n_i \).
4. For each $i$, $1 \leq i \leq m$: $\sum_{j=1}^{X} f_{i,j}(v_j) \geq r_i - \text{miss}_i$.

One can verify that solutions to this program directly correspond to $\epsilon$-almost-covers for input instance $I$ that use at most $B$ sets (as required). This completes the proof. □

We note that in the above algorithm we use many constraints with large coefficient values. Thus, as in the case of Theorem 4, it seems challenging to replace our approach with $n$-fold integer programming.

3.2. From covering problems to approval voting

In this section we show a relation between several problems regarding Approval voting and the covering problems studied above. In consequence, we will explain how our technique behind Theorem 2 can be used for obtaining fixed-parameter tractability results for these voting problems.

We model an election as a pair $E = (C, V)$, where $C = \{c_1, \ldots, c_m\}$ is a 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). In other words, 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 $\text{score}_E(c_i)$ 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, voters also have integer weights in addition to their preferences. A voter $v$ with weight $\omega(v)$ counts as $\omega(v)$ copies of an unweighted voter.$^5$

We are interested in the following three problems.

**Definition 3** (Bartholdi et al. [2], Faliszewski et al. [20,49]). In each of the problems Approval-$S\text{Bribery}$ (priced bribery), Approval-$S\text{CAV}$ (priced control by adding voters), and Approval-$S\text{CDV}$ (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-$S\text{Bribery}$, for each voter $v_i$, $1 \leq i \leq n$, we are given a nonnegative integer price $\pi_i$; for this price we can change $v_i$’s approval set in any way we choose.
2. In Approval-$S\text{CAV}$ (CAV stands for “Constructive Control by Adding Voters”) we are given a collection $Q = \{q_1, \ldots, q_{n'}\}$ of additional voters. For each additional voter $q_i$, $1 \leq i \leq n'$, we also have a nonnegative integer price $\pi_i$ for adding $q_i$ to the original election.
3. In Approval-$S\text{CDV}$ (CDV stands for “Constructive Control by Deleting Voters”), we have a nonnegative integer price $\pi_i$ for removing each voter $v_i$ from the election.

In the weighted variants of these problems (which we denote by putting “$W$ighted” 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. Briefly put, the relation between WMM and various election problems (as 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.

**Construction 1.** Consider an instance of Approval-$S\text{CDV}$ with election $E = (C, V)$, where $C = \{p, c_1, \ldots, c_m\}$ and $V = \{v_1, \ldots, v_n\}$, with prices $\pi_1, \ldots, \pi_n$ that one needs to pay to the respective voters in order to convince them 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 $\pi_i$ 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 $\max(\text{score}_E(c_i) - \text{score}_E(p), 0)$. 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 $B$.

$^5$ There is a name clash between the literature on covering problems and that on elections. In the former, “weights” refer to what the 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 the respective parts of the literature to make our results more accessible to researchers from both communities.
Naturally, we do not use the full generality of WMM in Construction 1: in fact, we provide a reduction to Weighted Set Multicover, where the multiplicities of input multisets are either 0 or 1. This is important since Proposition 1 says that WMM is NP-hard even for a single element in the universe. From the viewpoint of voting theory, it is also interesting to consider Uniform Multiset Multicover, where for each multiset $S_i$ in the input instance there is a number $t_i$ such that all elements belonging to $S_i$ have multiplicity either equal to zero or to $t_i$. Using an argument similar to that used in Construction 1, it is easy to show that Uniform Multiset Multicover is, in essence, equivalent to Approval-Weighted-CDV.

In Construction 1 we have considered Approval-$\text{Bribery}$ 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-$\text{CCAV}$ and for Approval-$\text{Bribery}$. Formally, we have the following result.

**Proposition 3.** Each of the problems Approval-$\text{SCCAV}$, Approval-$\text{SCCDV}$, and Approval-$\text{Bribery}$ reduce to Weighted Set Multicover (with Turing reductions), whereas both Approval-Weighted-$\text{CCAV}$ and Approval-Weighted-$\text{CDV}$ reduce to Uniform Multiset Multicover (with Turing reductions). All these reductions run in polynomial time and have the property that the universe sets in the output instances are the same as the candidate sets in the input instances.

**Proof.** Formally, we either use standard many-one reductions or very simple special cases of Turing-reductions: In an outer loop, we iterate through certain values, which give an additional hint on how the solution looks like (we refer to it as “guessing”), then solve the remaining problem by a transformation to one of the two covering problems. We finally answer yes if one of the covering instances was a yes-instance. In our reductions, the universe set $U$ is always identical to the candidate set $C$, but the covering requirements, the family $S$ of the (multi)sets, the weights, and the prices differ.

**Approval-$\text{SCCDV}$**. See Construction 1.

**Approval-$\text{Bribery}$**. Consider an instance of Approval-$\text{Bribery}$ 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 changing the voter’s approval set, and with budget $B$. Observe that Approval-$\text{SCCDV}$ is very similar to Approval-$\text{SCCDV}$, because we can assume without loss of generality that each bribed voter finally approves only candidate $p$. However, the decisive difference is that we do not know the final number of approvals that $p$ will get because this depends on the given budget $B$, on the prices of the voters, and on how many bribed voters already approved $p$ (but together with some other candidates). We circumvent this lack of knowledge by guessing the number $\ell$ of additional approvals $p$ obtains through the bribery process. This also gives us the score $s^* := \text{score}_E(p) + \ell$ of $p$ in the final election (containing the $\ell$ bribed voters). Now, we have to ensure (i) that $p$ really obtains the guessed score and (ii) that all other candidates which originally have a higher score lose enough approvals through the bribery process. We can express this as an instance of Weighted Set Multicover as follows. For each voter $v_i \in V$, we form a set $S_i$ with weight $\pi_i$ that includes all the candidates approved by $v_i$ and also candidate $p$, but only if $v_i$ does not approve $p$. For each candidate $c \in C \setminus \{p\}$, we set its covering requirement to be $\max(\text{score}_E(c) - s^*, 0)$. For $p$ we set the covering requirement to $\ell$. Finally, we set the budget to be $B$. It is easy to see that there is a way to ensure $p$’s victory by adding voters of total cost at most $B$, who increase $p$’s score by at least $\ell$, if and only if we have constructed a “yes”-instance of Weighted Set Multicover.

**Approval-$\text{SCCAV}$**. Consider an instance of Approval-$\text{SCCAV}$ with election $E = (C, V)$, where $C = \{p, c_1, \ldots, c_m\}$, $V = \{v_1, \ldots, v_n\}$, and $Q = \{q_1, \ldots, q_n\}$, with prices $\pi_1, \ldots, \pi_m$ that one needs to pay to the respective voters from $Q$ in order to convince them to participate in the election, and with budget $B$. It is never useful to add a voter that does not approve candidate $p$. Adding a voter $w$ (who approves $p$) to the election has one decisive effect: it decreases the score difference between candidate $p$ and each candidate that is not approved by $w$. Hence, we can express this instance as an instance of Weighted Set Multicover as follows. For each voter $q_i \in Q$ approving $p$, we form a set $S_i$ with weight $\pi_i$ that includes exactly the candidates not approved by $q_i$. For each candidate $c \in C$, we set its covering requirement to be $\max(\text{score}_E(c) - \text{score}_E(p, 0), 0)$. It is easy to see that there is a way to ensure $p$’s victory by adding voters of total cost at most $B$ if and only if it is possible to solve the presented instance of Weighted Set Multicover with budget $B$.

**Approval-Weighted-CCDV and Approval-Weighted-CCAV**. By analogous arguments as above we do the same construction as for Approval-SCDV (resp. Approval-SCCAV) except that (i) we omit the weights of the multisets, and (ii) we set the multiplicity for each element in the multiset to the weight of the corresponding voter. In consequence, we obtain instances of Uniform Multiset Multicover.

Using the above result and the algorithms from Section 3.1 (or the faster algorithm for Weighted Set Multicover [40]), we immediately get the following corollary.

**Corollary 1.** Approval-$\text{SCCAV}$, Approval-$\text{SCCDV}$, Approval-$\text{Bribery}$, Approval-Weighted-$\text{CCAV}$, and Approval-Weighted-$\text{CDV}$ are fixed-parameter tractable when parameterized by the number of candidates.
For the first three problems mentioned in Corollary 1, we can either use the algorithm of Knop et al. [40] to solve \textsc{Weighted Set Multicover}, or we can use our approach. The former gives running time with single-exponential dependence on the number of candidates, whereas the latter gives double-exponential dependence, but makes it easy to use off-the-shelf commercial MIP solvers. For the other two problems we have no choice and we have to use our algorithm for \textsc{Uniform Multiset Multicover}, which yields double-exponential dependence on the number of candidates.

Finally, it is either shown explicitly by Faliszewski et al. [20] or follows trivially that when the problems mentioned in Corollary 1 have both prices and weights, then they are NP-hard already for two candidates (that is, they are \textsc{Para-NP}-hard with respect to the number of candidates).

4. Further generalizations of the results related to voting

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 $\alpha_1$ points to his or her most preferred candidate, $\alpha_2$ 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 $R$ in the ordinal model, it is straightforward to define $R$-$\$SCCAV, $R$-$\$SCCDV, and $R$-$\$SBribery. Using our new framework, we obtain the following result.

\textbf{Theorem 6.} For every voting rule $R$ for which winner determination can be expressed through a set of integer inequalities over variables that indicate how many voters with each given preference order are in the election, $R$-$\$SCCAV, $R$-$\$SCCDV, and $R$-$\$SBribery are fixed-parameter tractable when parameterized by the number of candidates.

\textbf{Proof.} The proof follows the same structure as that of Theorem 3. We will present the proof only for $R$-$\$SCCDV; the other cases follow by applying the same approach. Let us consider an instance of $R$-$\$SCCDV with the set of candidates $C = \{c_1, \ldots, c_m\}$, the collection of voters $V = \{v_1, \ldots, v_n\}$, prices $\pi_1, \ldots, \pi_n$, and with budget $B$.

Let $X$ be the set of integer variables which indicate how many voters with each given preference order are in the election. By $x_\sigma$ we denote the variable from $X$ which corresponds to the preference ranking $\sigma$. Clearly, the size of $X$ is upper-bounded by $m!$, i.e., by a function of the number of the candidates. Let $S$ be the set of inequalities over variables from $X$ that encode that $p$ is a winner in the election.

We construct an integer program with convex transformations as follows. For each preference order $\sigma$ we introduce one integer variable $v_\sigma$. Intuitively, this variable indicates how many voters with the preference order $\sigma$ we need to remove from the election. Additionally, we introduce a function $f_\sigma$ such that $f_\sigma(v_\sigma)$ is the total price of $v_\sigma$ least expensive voters whose preference order is $\sigma$. For each $x_\sigma \in X$ we replace $x_\sigma$ in $S$ with the number of voters from $V$ whose preference ranking is $\sigma$ minus $v_\sigma$, and we add a constraint enforcing that this difference is at least to zero. Finally, we add the budget constraint $\sum_{\sigma} f_\sigma(v_\sigma) \leq B$. It is apparent that our ILP is feasible if and only if the answer to the original instance is “yes”.

We solve it in FPT time via Theorem 2. \qed

For a more detailed description of the class of voting rules where “winner determination can be expressed through integer linear inequalities,” we point the reader to the works of Dorn and Schlotter [14] or of Faliszewski et al. [21]. In particular, Theorem 6 applies to all scoring rules. Indeed, careful inspection of the above proof gives the following corollary.

\textbf{Corollary 2.} There is an algorithm that, given as input a scoring rule $R$ for $m$ candidates and an instance of $R$-$\$SCCAV, $R$-$\$SCCDV, or $R$-$\$SBribery with $m$ candidates, outputs a correct answer for this instance in FPT time with respect to the number of candidates as parameter.

This, and the results from the previous section, resolves an issue dating back to the work of Faliszewski et al. [20, Theorem 4.4, Theorem 4.13; the conference version of their work was published in 2006], who have shown that $\$SBribery is in XP for approval voting and for each scoring rule, given as part of the input (for the parameterization by the number of candidates).\footnote{They did not speak of XP-membership explicitly, but this is exactly what they have shown. Bredereck et al. [7] popularized the issue of resolving whether $\$SBribery problems parameterized by the number of candidates are in FPT or are W[1]-hard, leading in particular to our solution and to the slightly later work of Knop et al. [39].} Until our work, the exact parameterized complexity of these problems was unknown (since the conference version of our paper, Knop et al. [39] provided a more general variant of this result).

Knop et al. [39,40] also showed algorithms for the problems mentioned in Theorem 6 and, indeed, in many cases they achieve asymptotic running times. Specifically, their algorithms run in single-exponential time with respect to the
number of candidates—provided that the linear programs describing the voting rules do not use large coefficients—whereas for our algorithms this dependence is double-exponential (but irrespective of the coefficients used in the programs specifying the voting rules). Yet, in some cases using our approach has advantages. First, in many practical settings our algorithms can use off-the-shelf commercial MIP solvers, which can be faster than n-fold IP solvers (indeed, we can use any standard MIP solver, and many of them are highly optimized; n-fold IP solvers are still in their infancy). Second, using our approach it is possible to derive Corollary 2, which gives a single FPT algorithm for all our problems and for all scoring protocols; doing the same based on n-fold integer programming seems difficult. The reason is that in both approaches one needs (seems to need) to put the score values from the scoring vectors as coefficients within the constraints. This comes at essentially no price for our approach, but pushes the running times of n-fold IP solves beyond FPT time (this does not preclude FPT algorithms for specific scoring protocols, such as Borda, or even the scoring rule defined by vectors \((2^{m-1}, 2^{m-2}, \ldots, 1)\),

because then the highest score value can be seen as a function of the parameter, i.e., the number of candidates).

Our framework also allows to partially resolve an open problem posed by Bredereck et al. [9] 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”, 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 \(v''\). Using this assumption, we obtain the following result (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 7.** For Borda (and for Maximin and Copeland voting rules), Shift Bribery for sortable price functions and for all-or-nothing price functions is fixed-parameter tractable when parameterized by the number of candidates.

Bredereck et al. [9] gave an FPT approximation scheme for the problems from Theorem 7; we can use part of the corresponding algorithm and apply our new framework in order to derive an exact and not only approximate solution. Their algorithm rephrases the problem and then applies a bounded search through the solution space. We can use their rephrasing but replace the search by solving some MIP/SPlIt instance. We omit technical details since recently Knop et al. [39,40] showed faster FPT algorithms for the Swap Bribery problem (also parameterized by the number of candidates) using the n-fold IP technique. Swap Bribery is a generalization of Shift Bribery, so the results of Knop et al. [39,40] are strictly stronger than ours (in particular, they achieve single-exponential dependence of the running time on the number of candidates, whereas we achieve double-exponential one).

### 5. Discussion & outlook

We have proposed an extension of Lenstra’s famous result for solving ILPs. In our extended formulation, one can replace any integer variable with its simple piecewise linear transformation—this transformation needs to be either convex or concave, depending on the position of the variable in the ILP. We have shown that such extended ILPs can still be solved in FPT time with respect to the number of integer variables, as long as there are at most polynomially many pieces.

We have demonstrated several applications of our general result which relate to classic covering problems and to selected voting problems. Most notably, we have proven that Weighted Set Multicover and Uniform Multiset Multicover are fixed-parameter tractable when parameterized by the number of elements to cover (the former result has meanwhile been strengthened by Knop et al. [40]). Further, building upon our general result, but using a more technically involved argument, we have proved the existence of an FPT approximation scheme for Multiset Multicover, also for the parameterization by the number of elements. We have also explained how our general results can be used in studies on control and bribery in elections—we have shown that certain variants of these problems are in FPT when parameterized by the number of candidates. In particular, we have resolved the parameterized complexity of some problems that were open for around ten years (again, some of these results—but not all—were strengthened by Knop et al. [39,40]; see the discussion in the results sections for details).

Our paper leads to several possible directions for future work. First, unfortunately, while Lenstra’s algorithm is a very powerful tool for proving FPT membership, it might be too slow in practice. Thus, as pointed out by Bredereck et al. [7], 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 usually seems very difficult. In practice, one would probably not use Lenstra’s algorithm for solving MILPs, but, instead, one of the off-the-shelf optimized heuristics. In the conference version of this paper [8], we provided a preliminary empirical comparison of the running times of the MIP-based algorithm (using an off-the-shelf MIP 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). Our results suggest that FPT algorithms based on solving MILPs can be very efficient in practice. A more thorough experimental analysis of these and similar questions would help in understanding the real power and limitations of the techniques based on

---

7 However, for such a scoring rule the n-fold IP approach also leads to algorithms whose running time has double-exponential dependence on the number of candidates.
MILPs, thus we believe it is an important research direction. (For the case of $n$-fold integer programming, some practical evaluations of their effectiveness are due to Altmanová et al. [1]). Second, our work advances a fairly modest literature on FPT approximation schemes. It would be very interesting to further explore the (practical) relevance of such algorithms.

Declaration of competing interest

No conflict of interest.

Acknowledgements

A preliminary version of this paper appeared under the title “Elections with Few Candidates: Prices, Weights, and Covering Problems” in Proceedings of the 4th International Conference on Algorithmic Decision Theory, ADT 2015 [8]. This journal version focuses on the theoretical part which is substantially revised and improved.

We thank Dušan Knop and Martin Koutecký for their exceedingly helpful input and feedback concerning related work.

Robert Bredereck was from September 2016 to September 2017 on postdoctoral leave at the University of Oxford, supported by the DFG fellowship BR 5207/2 and at project start in early 2015 supported by DFG project PAWS (NI 369/10). Piotr Skowron was supported by a Humboldt Research Fellowship for Postdoctoral Researchers (Alexander von Humboldt Foundation, Bonn, Germany) while staying at TU Berlin from February 2017 till January 2018, and later on by the Foundation for Polish Science within the Homing programme (Project title: “Normative Comparison of Multiwinner Election Rules”). Piotr Faliszewski was supported by DFG Mercator Fellowship within project PAWS (NI 369/10) during his stay at TU Berlin and by AGH University grant 11.11.230.337 (statutory research) afterward, and eventually by a Friedrich Wilhelm Bessel Research Award from the Alexander von Humboldt Foundation. Nimrod Talmon was supported by DFG, Research Training Group “Methods for Discrete Structures” (GRK 1408), while the author was affiliated with TU Berlin, where most of the work was done. Later on, Nimrod Talmon was supported by the Israel Science Foundation (ISF; Grant No. 630/19). This work was also partly supported by COST Action IC1205 on Computational Social Choice.

References

[1] K. Altmanová, D. Knop, M. Koutecký, Evaluating and tuning n-fold integer programming, in: Proceedings of the 17th International Symposium on Experimental Algorithms (SEA ’18), 2018, chapter 10, 14 pp.
[2] J.J. Bartholdi III, C.A. Tovey, M.A. Trick. How hard is it to control an election? Math. Comput. Model. 16 (8–9) (1992) 27–40.
[3] D. Baumeister, P. Faliszewski, J. Lang, J. Rothe, Campaigns for lazy voters: truncated ballots, in: Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS ’12), 2012, pp. 577–584.
[4] P. Berman, B. DasGupta, E. Sontag, Randomized approximation algorithms for set multicover problems with applications to reverse engineering of protein and gene networks, Discrete Appl. Math. 155 (6–7) (2007) 739–749.
[5] E. Bonnet, V. Paschos, F. Sikora, Parameterized exact and approximation algorithms for maximum $k$-set cover and related satisfiability problems, RAIRO Theor. Infm. Appl. 50 (3) (2016) 227–240.
[6] F. Brandt, V. Conitzer, U. Endriss, J. Lang, A.D. Procaccia (Eds.), Handbook of Computational Social Choice. Cambridge University Press, 2016.
[7] R. Bredereck, J. Chen, P. Faliszewski, J. Guo, R. Niedermeier, G.J. Woeginger, Parameterized algorithms for computational social choice: nine research challenges, Tsinghua Sci. Technol. 19 (4) (2014) 358–373.
[8] R. Bredereck, P. Faliszewski, R. Niedermeier, P. Skowron, N. Talmon, Elections with few candidates: prices, weights, and covering problems, in: Proceedings of the 4th International Conference on Algorithmic Decision Theory (ADT ’15), 2015, pp. 414–431.
[9] R. Bredereck, J. Chen, P. Faliszewski, A. Nichtelein, R. Niedermeier, Prices matter for the parameterized complexity of shift bribery, Inf. Comput. 251 (2016) 140–164.
[10] I. Caragiannis, D. Kurokawa, H. Moulin, A.D. Procaccia, N. Shah, J. Wang, The unreasonable fairness of maximum Nash welfare, ACM Trans. Econ. Comput. 7 (3) (2019) 12, 32 pp.
[11] V. Conitzer, T. Sandholm, J. Lang, When are elections with few candidates hard to manipulate? J. ACM 54 (3) (2007) 1–33.
[12] D. Dadush, C. Peikert, S. Vempala, Enumerative lattice algorithms in any norm via M-ellipsoid coverings, in: Proceedings of the 52nd Annual IEEE Symposium on Foundations of Computer Science (FOCS ’11), 2011, pp. 580–589.
[13] I. Dinur, D. Steurer, Analytical approach to parallel repetition, in: Proceedings of the 46th ACM Symposium on Theory of Computing (STOC ’14), 2014, pp. 624–633.
[14] B. Dorn, I. Schlotter, Multivariate complexity analysis of swap bribery, Algorithmica 64 (1) (2012) 126–151.
[15] R.G. Downey, M.R. Fellows, Fundamentals of Parameterized Complexity, Springer, 2013.
[16] P. Dvořák, E. Eiben, R. Ganian, D. Knop, S. Ordyniak, Solving integer linear programs with a small number of global variables and constraints, in: Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI ’17), 2017, pp. 607–613.
[17] F. Eisenbrand, C. Hunke, C. Klein, Faster algorithms for integer programs with block structure, in: Proceedings of the 45th International Colloquium on Automata, Languages, and Programming (ICALP ’18), 2018, chapter 49, 13 pp.
[18] E. Elkind, P. Faliszewski, A. Slinko, Swap bribery, in: Proceedings of the 2nd International Symposium on Algorithmic Game Theory (SAGT ’15), Oct. 2009, pp. 299–310.
[19] P. Faliszewski, J. Rothe, Control and bribery in voting, in: F. Brandt, V. Conitzer, U. Endriss, J. Lang, A.D. Procaccia (Eds.), Handbook of Computational Social Choice, Cambridge University Press, 2016, chapter 7.
[20] P. Faliszewski, E. Hemaspaandra, L.A. Hemaspaandra, How hard is bribery in elections? J. Artif. Intell. Res. 35 (2009) 485–532.
[21] P. Faliszewski, E. Hemaspaandra, L.A. Hemaspaandra, Multimode control attacks on elections, J. Artif. Intell. Res. 40 (2011) 305–351.
[22] P. Faliszewski, E. Hemaspaandra, L.A. Hemaspaandra, Weighted electoral control, J. Artif. Intell. Res. 52 (2015) 507–542.
[23] P. Faliszewski, P. Skowron, N. Talmon, Bribery as a measure of candidate success: complexity results for approval-based multiwinner rules, in: Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS ’17), 2017, pp. 6–14.
[24] P. Faliszewski, P. Skowron, A. Slinko, N. Talmon, Multiwinner analogues of the plurality rule: axiomatic and algorithmic views, Soc. Choice Welf. 51 (3) (2018) 513–550.
[25] U. Feige. A threshold of lnn for approximating set cover. J. ACM 45 (4) (1998) 634–652.
[26] T. Fluschnik, P. Skowron, M. Triphaus, K. Wilker, Fair knapsack, in: Proceedings of the 33rd Conference on Artificial Intelligence (AAAI ’19), 2019, pp. 1941–1948.
[27] A. Frank, É. Tardos, An application of simultaneous Diophantine approximation in combinatorial optimization, Combinatorica 7 (1) (1987) 49–65.
[28] T. Gavenčiak, D. Knop, M. Koutecký, Integer programming in parameterized complexity: three miniatures, in: 13th International Symposium on Parameterized and Exact Computation (IPEC ’18), 2019, chapter 21, 16 pp.
[29] E. Hemaspaandra, L.A. Hemaspaandra, J. Rothe, Anyone but him: the complexity of precluding an alternative, Artif. Intell. 171 (5–6) (2007) 255–285.
[30] R. Hemmecke, S. Onn, L. Romanchuk, n-fold integer programming in cubic time, Math. Program. 137 (1–2) (2013) 325–341.
[31] R. Hildebrand, M. Köppe, A new Lcnstra-type algorithm for quasi-convex polynomial integer minimization with complexity $2^{\bar{O}(n^{10})}$, Discrete Optim. 10 (1) (2013) 69–84.
[32] Q. Hu, D. Yu, F.C.M. Lau, Y. Wang, Exact algorithms for set multicolor and multiset multicolor problems, in: Proceedings of the 20th International Symposium on Algorithms and Computation (ISAAC ’09), 2009, pp. 34–44.
[33] R. Izsák, N. Talmon, G.J. Woeginger, Committee selection with intraclass and interclass synergies, in: Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI ‘18), 2018, pp. 1071–1078.
[34] D.S. Johnson, Approximation algorithms for combinatorial problems, J. Comput. Syst. Sci. 9 (3) (1974) 256–278.
[35] R. Kannan, Minkowski’s convex body theorem and integer programming, Math. Oper. Res. 12 (3) (1987) 415–440.
[36] R.M. Karp, Reducibility among combinatorial problems, in: R.E. Miller, J.W. Thatcher (Eds.), Complexity of Computer Computations, Plenum Press, 1972, pp. 85–103.
[37] L. Khachiyan, L. Porkolab, Integer optimization on convex semialgebraic sets, Discrete Comput. Geom. 23 (2) (2000) 207–224.
[38] D. Knop. M. Koutecký, Scheduling meets n-fold integer programming, J. Sched. 21 (5) (2018) 493–503.
[39] D. Knop, M. Koutecký, M. Mnich, Voting and bribing in single-exponential time, in: Proceedings of the 34th Symposium on Theoretical Aspects of Computer Science (STACS ’17), 2017, chapter 46, 14 pp.
[40] D. Knop, M. Koutecký, M. Mnich, Combinatorial n-fold integer programming and applications, in: Proceedings of the 25th Annual European Symposium on Algorithms (ESA ’17), vol. 87, 2017, chapter 54, 14 pp.
[41] S.G. Kolliopoulos, Approximating covering integer programs with multiplicity constraints, Discrete Appl. Math. 129 (2) (2003) 461–473.
[42] S.G. Kolliopoulos, N.E. Young, Approximation algorithms for covering/packing integer programs, J. Comput. Syst. Sci. 71 (4) (2005) 495–505.
[43] M. Koutecký, A. Levin, S. Onn, A parameterized strongly polynomial algorithm for block structured integer programs, in: Proceedings of the 45th International Colloquium on Automata, Languages, and Programming (ICALP ’18), 2018, chapter 85, 14 pp.
[44] H.W. Lenstra Jr., Integer programming with a fixed number of variables, Math. Oper. Res. 8 (4) (1983) 358–354.
[45] J.D. Loera, R. Hemmecke, S. Onn, R. Weismantel, n-fold integer programming, Discrete Optim. 5 (2) (2008) 231–241.
[46] J.D. Loera, R. Hemmecke, M. Köppe, Algebraic and geometric ideas in the Theory of Discrete Optimization, Society for Industrial and Applied Mathematics, 2012.
[47] N. Mattei, M.S. Pini, F. Rossi, K.B. Venable, Bribery in voting over combinatorial domains is easy, in: Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS ’12), 2012, pp. 1407–1408.
[48] R.R. Meyer, On the existence of optimal solutions to integer and mixed–integer programming problems, Math. Program. 7 (1) (1974) 223–233.
[49] T. Małgosz, P. Faliszewski, The complexity of priced control in elections, Ann. Math. Artif. Intell. 77 (3–4) (2016) 225–250.
[50] T. Oertel, C. Wagner, R. Weismantel, Integer convex minimization by mixed integer linear optimization, Oper. Res. Lett. 42 (6) (2014) 424–428.
[51] D. Peters, Single-peakedness and total uninomodularity: new polynomial-time algorithms for multi-winner elections, in: Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, 2018, pp. 1169–1176.
[52] S. Rajagopalan, V. Vaziri, Primal-dual RNC approximation algorithm for set cover and covering integer programs, SIAM J. Comput. 28 (2) (1999) 526–541.
[53] J. Roché (Ed.), Economics and Computation, An Introduction to Algorithmic Game Theory, Computational Social Choice, and Fair Division, Springer, 2016.
[54] J. Schlotter, P. Faliszewski, E. Elkind, Campaign management under approval-driven voting rules, Algorithmica 77 (1) (2017) 84–115.
[55] A. Schrijver, Theory of Linear and Integer Programming, Wiley, 1998.
[56] P. Skowron, FPT approximation schemes for maximizing submodular functions, Inf. Comput. 257 (2017) 65–78.
[57] P. Skowron, P. Faliszewski, Chamberlin–Courant rule with approval ballots: Approximating the maxcover problem with bounded frequencies in FPT time, J. Artif. Intell. Res. 60 (2017) 687–716.
[58] V.V. Vaziri, Approximation Algorithms, Springer, 2001.
[59] D.P. Williamson, D.B. Shmoys, The Design of Approximation Algorithms, Cambridge University Press, 2011.
[60] L. Xia, Computing the margin of victory for various voting rules, in: Proceedings of the 13th ACM Conference on Electronic Commerce (EC ’12), June 2012, pp. 982–999.

Please cite this article in press as: R. Bredereck et al., Mixed integer programming with convex/concave constraints: Fixed-parameter tractability and applications to multicovering and voting, Theoret. Comput. Sci. (2020), https://doi.org/10.1016/j.tcs.2020.01.017