Voting and Bribing in Single-Exponential Time

DUŠAN KNOP, Department of Theoretical Computer Science, Faculty of Information Technology, Czech Technical University in Prague, Czech Republic
MARTIN KOUTECKÝ, Charles University, Czech Republic
MATTHIAS MNICH, TU Hamburg, Germany

We introduce a general problem about bribery in voting systems. In the $\mathcal{R}$-Multi-Bribery problem, the goal is to bribe a set of voters at minimum cost such that a desired candidate is a winner in the perturbed election under the voting rule $\mathcal{R}$. Voters assign prices for withdrawing their vote, for swapping the positions of two consecutive candidates in their preference order, and for perturbing their approval count to favour candidates.

As our main result, we show that $\mathcal{R}$-Multi-Bribery is fixed-parameter tractable parameterized by the number of candidates $|C|$ with only a single-exponential dependence on $|C|$, for many natural voting rules $\mathcal{R}$, including all natural scoring protocols, maximin rule, Bucklin rule, Fallback rule, SP-AV, and any C1 rule. The vast majority of previous work done in the setting of few candidates proceeds by grouping voters into at most $|C|!$ types by their preference, constructing an integer linear program with $|C|!^2$ variables, and solving it by Lenstra's algorithm in time $|C|!^{|C|!^2}$, hence double-exponential in $|C|$. Note that it is not possible to encode a large number of different voter costs in this way and still obtain a fixed-parameter algorithm, as that would increase the number of voter types and hence the dimension. These two obstacles of double-exponential complexity and restricted costs have been formulated as "Challenges #1 and #2" of the "Nine Research Challenges in Computational Social Choice" by Bredereck et al.

Hence, our result resolves the parameterized complexity of $\mathcal{R}$-Swap-Bribery for the aforementioned voting rules plus Kemeny's rule, and for all rules except Kemeny brings the dependence on $|C|$ down to single-exponential. The engine behind our progress is the use of a new integer linear programming formulation, using so-called "n-fold integer programming." Since its format is quite rigid, we introduce "extended n-fold IP," which allows many useful modeling tricks. Then, we model $\mathcal{R}$-Multi Bribery as an extended n-fold IP and apply an algorithm of Hemmecke et al. [Math. Prog. 2013].

CCS Concepts: • **Theory of computation → Parameterized complexity and exact algorithms**;

Additional Key Words and Phrases: Voting system, swap bribery, integer programming

ACM Reference format:
Dušan Knop, Martin Koutecký, and Matthias Mnich. 2020. Voting and Bribing in Single-Exponential Time. *ACM Trans. Econ. Comput.* 8, 3, Article 12 (June 2020), 28 pages.
https://doi.org/10.1145/3396855

The work is supported by the European Research Council under Grant No. 306465, the Deutsche Forschungsgemeinschaft under Grant No. MN 59/4-1, by the OP VVV MEYS funded project CZ.02.1.01/0.0/0.0/16 019/0000765 "Research Center for Informatics," by the Charles University project UNCE/SCI/004, and by the project 19-27871X of GA ČR.
Authors' addresses: D. Knop, Department of Theoretical Computer Science, Faculty of Information Technology, Czech Technical University, Thákurova 9, 160 00, Prague, Czech Republic; email: dusan.knop@fit.cvut.cz; M. Koutecký, Univerzita Karlova, Matematicko-fyzikální fakulta, Informatický ústav Univerzity Karlovy, Malostranské nám. 25, 118 00, Prague, Czech Republic; email: koutecky@iuuk.mff.cuni.cz; M. Mnich, Hamburg University of Technology, Institute for Algorithms and Complexity, Blohmstr. 15, 21079 Hamburg, Germany; email: mmnich@tuhh.de.

2167-8375/2020/06-ART12
https://doi.org/10.1145/3396855

1 INTRODUCTION

In this work, we address algorithmic problems from the area of voting and bribing. In these problems, we are given as input an election, which consists of a set C of candidates and a set V of voters v, each of which is equipped with a linear order $\succ_v$ indicating their preferences over the set C of candidates. Further, we have a fixed voting rule $\mathcal{R}$ (that is not part of the input), which determines how the orders of the voters are aggregated to determine the winner(s) of the election among the candidates. Popular examples of voting rules $\mathcal{R}$ include "scoring protocols" like *plurality*—where the candidate(s) ranked first by most voters win(s)—or the Borda rule, where each candidate receives $|C| - i$ points from being ranked ith by a voter and the candidate with most points is a winner; and the Copeland rule, which orders candidates by their number of pairwise victories (by majority) minus their number of pairwise defeats.

The goal is to *perturb* the given election (C, V) by *bribing voters* through *bribery actions* Γ in such a way that a designated candidate $c^\star \in C$ is a winner in the perturbed election $(C, V)^\Gamma$ under the voting rule $\mathcal{R}$. Such perturbation problems model various real-life issues, such as actual bribery, campaign management, or post-election checks, as in destructive bribery (known as "margin of victory"); for an overview of the many flavors of bribery problems we refer to a recent survey by Faliszewski and Rothe [35].

$\mathcal{R}$- Multi-Bribery. Perturbation is performed by the actions of *swapping* the position of two adjacent candidates in the preference order of some voter, by *push actions* that perturb the approval count of a voter[1] and *control changes* that (de)activate some voters. The algorithmic problem is to achieve the goal by performing the most cost-efficient actions. To measure cost of swaps, we consider the model introduced by Elkind et al. [25] where each voter may assign different prices $\sigma^v(c, c')$ for swapping two consecutive candidates c, c' in their preference order; this captures the notion of small changes and comprises the preferences of the voters. If voter v is involved in a swap or push action, then a one-time influence cost ι^v occurs. We additionally allow voter-individual cost π^v for push actions and voter-individual cost α^v and δ^v for activation and deactivation. Our model is general enough that we even allow *negative* costs. Unless explicitly stated, we assume preference orders are total, with the following exceptions. We assume general partial orders to be given in the Possible Winner problem. Further, we assume top-truncated orders to be given in $\mathcal{R}$-Extension-Bribery; with top-truncated orders voters rank only their top candidates and are indifferent toward the other candidates. Last, we can also handle weak (or bucket) orders, which are linear orders over partitions of candidates, where a voter is indifferent between candidates in each part, see Section 2. The topic of Swap Bribery in general partial orders is a non-trivial and interesting one, and we discuss it later. We call this very general set-up the $\mathcal{R}$-Multi-Bribery problem.

Various special cases of the $\mathcal{R}$-Multi-Bribery problem have been studied in the literature; see Table 1 for an overview of which problems are captured by $\mathcal{R}$-Multi-Bribery. For instance, in $\mathcal{R}$-Swap-Bribery, only swaps are permitted. However, in $\mathcal{R}$-Support-Bribery, only push actions are allowed. And in the Young Score problem, only control changes are allowed.

The aforementioned many special cases of $\mathcal{R}$-Multi-Bribery have been investigated extensively from an algorithmic viewpoint. As it turns out, most of the cases looked at are NP-hard, such as $\mathcal{R}$-Swap-Bribery, $\mathcal{R}$-Shift-Bribery, and others. Therefore, we expect algorithms solving these problem exactly to take superpolynomial time (assuming P $\neq$ NP). Yet, in many application scenarios it is reasonable to assume that the number $|C|$ of candidates is small; it has therefore been of high interest to design algorithms for NP-hard score (winner determination) and bribery

[1]Here, the approval count of a voter v specifies how many top candidates receive some points from v. See also the description of SP-AV.

Table 1. $\mathcal{R}$-Multi-Bribery Generalizes Several Studied Bribery Problems, Whose Formal Definitions We Give in Appendix A

Problem name	Specialization of $\mathcal{R}$-Multi-Bribery
$\mathcal{R}$-$Bribery	$\sigma^v \equiv 0, \pi^v \equiv \infty, \alpha^v \equiv \delta^v \equiv \infty$
$\mathcal{R}$-Manipulation	$\iota^v \equiv 0$ for $v \in M$, $\iota^v \equiv \infty$ for $v \notin M$
$\mathcal{R}$-CCAV/$\mathcal{R}$-CCDV	$\iota^v \equiv 0, \sigma^v \equiv \pi^v \equiv \infty$
$\mathcal{R}$-Swap-Bribery	$\pi^v \equiv \infty, \alpha^v \equiv \delta^v \equiv \infty, \iota^v \equiv 0$
$\mathcal{R}$-Shift-Bribery	$\mathcal{R}$-Swap-Bribery with $\sigma^v(c, c') = \infty$ for $c^\star \notin \{c, c'\}$
$\mathcal{R}$-Support Brib.	$\sigma^v \equiv \alpha^v \equiv \delta^v \equiv \infty, \iota^v \equiv 0$
$\mathcal{R}$-Mixed-Bribery	$\alpha^v \equiv \delta^v \equiv \infty, \iota^v \equiv 0$
$\mathcal{R}$-Extension-Bribery	$\sigma^v(c, c') = 0$ if c, c' unranked, else $\sigma^v(c, c') = \infty$; $\alpha^v \equiv \delta^v \equiv \infty, \iota^v = 0$
$\mathcal{R}$-Possible Winner	reduce to $\mathcal{R}$-Swap-Bribery [25, Thm. 2]
Dodgson Score	Condorcet-Swap-Bribery with $\sigma^v \equiv 1$
Young Score	Condorcet-CCDV with $\delta^v \equiv 1$

For $\mathcal{R}$-Manipulation the set of manipulators $M \subseteq V$ is given in the input. If voter v is involved in a swap or push action, then a one-time influence cost ι^v occurs.

problems that exploit this property. In particular, a quest for algorithms that solve instances I of size n in time $f(|C|) \cdot n^{O(1)}$ for some computable function f has been made; such algorithms are called *fixed-parameter algorithms*. Fixed-parameter algorithms are contrasted with so-called XP-algorithms that have runtimes of the form $n^{f(|C|)}$. Whereas XP-algorithms are generally considered impractical even for small instances I and few candidates, fixed-parameter algorithms have the potential to be practical, provided that the function f exhibits moderate growth.

The current situation for manipulation, control and bribery problems is that indeed, a large number of fixed-parameter algorithms for NP-hard problems in election manipulation, control, and bribery have been designed, when parameterized by $|C|$, for a multitude of voting rules [2, 6, 10, 11, 13–15, 22, 30]. For instance, a prototypical algorithm in that direction is due to Dorn and Schlotter [22], who show how to solve $\mathcal{R}$-Swap-Bribery with uniform costs[2] in time $2^{2^{O(|C|)}} \cdot n^{O(1)}$ for all so-called linearly describable voting rules $\mathcal{R}$; here, uniform costs means that swapping any two adjacent candidates always has the same cost for all voters. In the Dorn-Schlotter algorithm, as well as several other fixed-parameter algorithms designed for few candidates, the function $f(|C|)$ typically grows double-exponentially in $|C|$, which *a priori* makes these algorithms impractical even for very few candidates. As the double-exponential dependence on $|C|$ in those algorithms often stems from solving, as a subroutine, a certain integer linear program (ILP) by means of the celebrated algorithm of Lenstra [53], Bredereck et al. [9] put forward the following "Challenge #1," as part of their "Nine Research Challenges in Computational Social Choice":

> *Many [FPT results in computational social choice] rely on a deep result from combinatorial optimization due to Lenstra [that] is mainly of theoretical interest; this may render corresponding fixed-parameter tractability results to be of classification nature only. Can the mentioned ILP-based [...] results be replaced by direct combinatorial [...] fixed-parameter algorithms?*

[2]Bredereck et al. [10] pointed out that the algorithm by Dorn and Schlotter only works for uniform costs.

Another downside of the "ILP-based approach" is that it inherently treats voters not as individuals but as groups that share preferences. This makes it difficult to obtain algorithms where voters from the same group differ in some way, such as by the cost of bribing them. Their "Challenge #2" thus reads:

> *[T]here is a huge difference between [...] problems, where each voter has unit cost for being bribed, and the other flavors of bribery, where each voter has individually specified price [...] and it is not known if they are in* FPT *or hard for* W[1]. *What is the exact parameterized complexity of the* $\mathcal{R}$-Swap-Bribery *and* $\mathcal{R}$-Shift-Bribery *parameterized by the number of candidates, for each voting rule* $\mathcal{R}$*?*

1.1 Our Contribution

Our main result is a fixed-parameter algorithm for $\mathcal{R}$-Multi-Bribery parameterized by the number of candidates, for many fundamental voting rules $\mathcal{R}$. Our algorithm has a few advantages over previous works, in that it works for voter-dependent cost functions, and it runs in time that is only single-exponential in $|C|$. The algorithm applies to a large number of voting rules $\mathcal{R}$, such as all *natural* scoring protocols where each candidate receives at most $|C|$ points from each voter.

Theorem 1.1. $\mathcal{R}$-Multi-Bribery *is fixed-parameter tractable parameterized by the number of candidates, and can be solved in time*

- $2^{O(|C|^6 \log |C|)} \cdot n^3$ *when* $\mathcal{R}$ *is any natural scoring protocol, any C1 rule, or sincere-strategy preference-based approval voting (SP-AV),*
- $2^{O(|C|^6 \log |C|)} \cdot n^4$ *when* $\mathcal{R}$ *is the maximin, Bucklin, or Fallback rule, and*
- $2^{O(|C|!^6 \log |C|)} \cdot n^3$ *when* $\mathcal{R}$ *is the Kemeny rule.*

In summary, our algorithm subsumes, and improves, *all* previously devised algorithms for the problems listed in Table 1. For some problems, such as Young-Score, our algorithm yields the first improvement since 1977; we summarize the comparison in Table 2.

Applications of Theorem 1.1. We argued that $\mathcal{R}$-Multi-Bribery generalizes many well-studied voting and bribing problems, parameterized by the number of candidates. A direct corollary of Theorem 1.1 is as follows:

Corollary 1.2. *Let* $\mathcal{R}$ *be any natural scoring protocol, a C1 rule, the maximin rule, the Bucklin rule, the SP-AV rule, the Fallback rule, or Kemeny rule. Then* $\mathcal{R}$*-Swap-Bribery is fixed-parameter tractable parameterized by the number* $|C|$ *of candidates.*

This solves "Challenge #2" by Bredereck et al. [9]. In particular, for scoring protocols, maximin rule, and Bucklin rule, Corollary 1.2 extends and improves an algorithm by Dorn and Schlotter [22] that is restricted to the uniform cost case of $\mathcal{R}$-Swap-Bribery, and requires double-exponential runtime $2^{2^{O(|C|)}} \cdot n^{O(1)}$.

We remark that it is unclear (see Reference [30, p. 338]) if the Kemeny rule can be *described by linear inequalities* as defined by Dorn and Schlotter [22]; even if it does, ours is the first fixed-parameter algorithm for $\mathcal{R}$-Swap-Bribery under the Kemeny rule, as Dorn and Schlotter's algorithm only applies to the unit-cost case.

Table 2. Summary of Results from This Paper for $\mathcal{R}$-Multi-Bribery Compared to Previous Works for Special Cases

	Previous best result		New result
Problem	Runtime/hardness	Voting rules $\mathcal{R}$	
$\mathcal{R}$-$Bribery	$2^{2^{O(\lvert C\rvert)}}n^{O(1)}$	Approval [13]	$2^{O(\lvert C\rvert^6 \log \lvert C\rvert)} \cdot n^3$
$\mathcal{R}$-Manipulation	$2^{2^{O(\lvert C\rvert)}}n^{O(1)}$	Borda [7]	$2^{O(\lvert C\rvert^6 \log \lvert C\rvert)} \cdot n^3$
$\mathcal{R}$-CCAV/$\mathcal{R}$-CCDV	$2^{2^{O(\lvert C\rvert)}}n^{O(1)}$	Approval [13]	$2^{O(\lvert C\rvert^6 \log \lvert C\rvert)} \cdot n^3$
$\mathcal{R}$-Swap-Bribery	$2^{2^{O(\lvert C\rvert)}}n^{O(1)}$, uniform cost	Approval [22]	$2^{O(\lvert C\rvert^6 \log \lvert C\rvert)} \cdot n^3$
$\mathcal{R}$-Shift-Bribery	XP-algor., arbitrary cost, FPT-AS, restricted cost	Borda, Cope-land, Maximin [14]	$2^{O(\lvert C\rvert^6 \log \lvert C\rvert)} \cdot n^4$
$\mathcal{R}$-Support Brib.	NP-hard	Fallback, SP-AV [60]	$2^{O(\lvert C\rvert^6 \log \lvert C\rvert)} \cdot n^4$
$\mathcal{R}$-Mixed-Bribery	NP-hard	SP-AV [25]	$2^{O(\lvert C\rvert^6 \log \lvert C\rvert)} \cdot n^3$
$\mathcal{R}$-Extension-Bribery	NP-hard	Borda, Cope-land^0, Maximin [4]	$2^{O(\lvert C\rvert^6 \log \lvert C\rvert)} \cdot n^4$
$\mathcal{R}$-Possible Winner	$2^{2^{O(\lvert C\rvert)}}n^{O(1)}$	Bucklin, Copeland, pos. scoring protocols [6]	$2^{O(\lvert C\rvert^6 \log \lvert C\rvert)} \cdot n^3$
$\mathcal{R}$-$Bribery	XP-algor., arbitrary cost	Kemeny [29]	$2^{O(\lvert C\rvert!^6 \log \lvert C\rvert)} \cdot n^4$
Dodgson Score	$2^{2^{O(\lvert C\rvert)}}n^{O(1)}$ [2]		$2^{O(\lvert C\rvert^6 \log \lvert C\rvert)} \cdot n^3$
Young Score	$2^{2^{O(\lvert C\rvert)}}n^{O(1)}$ [63]		$2^{O(\lvert C\rvert^6 \log \lvert C\rvert)} \cdot n^3$

In each row corresponding to a problem $\mathcal{R}$-Problem, we state the previously best known dependency on $|C|$ for voting rules $\mathcal{R}$ for which $\mathcal{R}$-Problem is known to be NP-hard. For $\mathcal{R}$-Shift-Bribery, FPT-AS refers to a *fixed-parameter approximation scheme*, which is an algorithm that yields a $(1-\varepsilon)$-approximate solution in time $f(1/\varepsilon, |C|) \cdot n^{O(1)}$ for some superpolynomial function f and any $\varepsilon > 0$; it is thus a weaker result than a fixed-parameter algorithm.

Another corollary of Theorem 1.1 is the following:

Corollary 1.3. *$\mathcal{R}$-Shift-Bribery is fixed-parameter tractable parameterized by the number of candidates, for $\mathcal{R}$ being the Borda rule, the maximin rule, and the Copeland$^\alpha$ rule.*

This way, we simultaneously improve the fixed-parameter algorithm by Dorn and Schlotter [22] for uniform cost, the XP-algorithm and the fixed-parameter approximation scheme for arbitrary cost by Bredereck et al. [10]. Further, we have the following:

Corollary 1.4. *Approval-$Bribery, Approval-$CCAV, and Approval-$CCDV can be solved in time $2^{O(|C|^6 \log |C|)} \cdot n^4$.*

This improves a recent result by Bredereck et al. [13], who solved these problems in time that is double-exponential in $|C|$.

1.2 Our Approach

The runtimes that we achieve in Theorem 1.1 are (except for the Kemeny rule) only single-exponential in $|C|$. To achieve this, we avoid using Lenstra's algorithm for solving fixed-dimension ILPs [53], which was the method of choice so far (and which led to double-exponential runtimes). Typically, when using Lenstra's algorithm one has to "group objects" to be able to bound their number in terms of the used parameters. Instead, we formulate the $\mathcal{R}$-Multi-Bribery problem in terms of an n-fold integer program (IP). Unlike fixed-dimension ILPs, n-fold IPs allow variable dimension at the expense of a more rigid block structure of the constraint matrix. We manage to encode the $\mathcal{R}$-Multi-Bribery problem for many voting rules $\mathcal{R}$ in a constraint matrix that has this

required structure. The formulations are not straightforward: Rather, we model the problems in terms of an "extended" n-fold IP that has a more general format than is required by the "standard" n-fold IP discussed in the literature. Then we show how to efficiently transform any extended n-fold IP into a standard n-fold IP. The dimension (i.e., the number of variables) of this IP is not bounded by *any function* of the number $|C|$ of candidates; however, we bound the dimension of each block by a *polynomial* in $|C|$. Then we solve the standard n-fold IP via an algorithm of Hemmecke et al. [44] whose runtime depends exponentially only on the largest dimension of each of its blocks. That algorithm has a rather combinatorial flavor as it works by iterative augmentation, an approach similar to solving min-cost flow by (fast) cycle cancelation; see Hemmecke et al. [44]. This way, we substantially contribute toward resolving "Challenge #1" by Bredereck et al. [9].

Parameterized complexity. A parameterized problem is a decision problem (language) $\mathcal{P} \subseteq \Sigma^*$ accompanied with a parameter $\kappa : \Sigma^* \to \mathbb{N}$ mapping instances of $\mathcal{P}$ to values of the parameter. A parameterized problem $(\mathcal{P}, \kappa)$ is *fixed-parameter tractable* (and is in the class FPT) if there is an algorithm deciding whether x belongs to $\mathcal{P}$ or not with running time $f(\kappa(x)) \cdot |x|^{O(1)}$, where $f : \mathbb{N} \to \mathbb{N}$ is a computable function (independent of $|x|$). A parameterized problem $(\mathcal{P}, \kappa)$ is in the class XP if there is an algorithm deciding whether x belongs to $\mathcal{P}$ or not with running time $|x|^{f(\kappa(x))}$, where $f : \mathbb{N} \to \mathbb{N}$ is a computable function. Let $(\mathcal{P}, \kappa), (\mathcal{P}', \kappa')$ be two parameterized problems. We say that $(\mathcal{P}', \kappa')$ *reduces* to $(\mathcal{P}, \kappa)$ if there is an algorithm that on input $(y, \kappa'(y))$ in time $f(\kappa(y)) \cdot |y|^{O(1)}$ produces an instance x with $\kappa(x) \leq f(\kappa'(y))$ (for some computable function $f : \mathbb{N} \to \mathbb{N}$) such that $y \in \mathcal{P}'$ if and only if $x \in \mathcal{P}$. A parameterized problem $(\mathcal{P}, \kappa)$ is W[1]*-hard* if every problem in W[1] can be reduced to it. For background, we refer to the books by Flum and Grohe [39] or Cygan et al. [18].

1.3 Related Work

Bribery problems in voting systems are well studied [10, 22, 25, 29]. The most important bribery model for our study—the swap bribery—was introduced and first considered by Elkind et al. [25]. We also continue the study of Faliszewski et al. [30] who, in the control framework of Bartholdi et al. [2], proposed the study of *multiple* attacks (i.e., using more control paradigms at the same time—e.g., deleting a candidate and, simultaneously, adding few so far latent voters). Dorn and Schlotter [22] used an ILP model of bribery to give fixed-parameter algorithms for the parameter number of candidates in the given election, e.g., with uniform costs per swap and for $\mathcal{R}$ being a scoring protocol. Bredereck et al. [10] consider parameterized complexity of shift bribery (introduced by Elkind and Faliszewski [24] that, in particular, also focused on the connection between bribery and campaign management), where candidates can be shifted up a number of positions in a voter's preference order; this is a special case of swap bribery. An extension of their model [15] allows campaign managers to affect the position of the preferred candidate in multiple votes, either positively or negatively, with a single bribery action, which applies to large-scale campaigns. Complexity of bribery of elections admitting for multiple winners, such as when committees are formed, has been studied by Bredereck et al. [14]. Computational complexity of (swap) bribery in Bucklin and fallback elections is studied by Faliszewski et al. [34]. Bribery as a manipulation in iterative voting procedures, i.e., where the candidates are eliminated one by one in rounds, was studied by Maushagen et al. [55]—here typically NP-hardness of bribery with e.g., the Baldwin or Nanson rules is established. Dey et al. [21] study the so-called Frugal Bribery, where it is possible to bribe a voter only if the voter benefits by this, i.e., if the voter prefers the new outcome of the election to the current one. The same set of authors [20] study the complexity of finding a possible

set of manipulators (i.e., voters that can if work in a synergy change the outcome of an election). Their study is mostly focused on classical complexity mainly in the case of small groups of manipulators (e.g., sets of size $O(1)$). Also, different cost models have been considered: Faliszewski et al. [29] require that each voter has their own price that is independent of the changes made to the bribed vote. The more general models of Faliszewski [27] and Faliszewski et al. [31] allow for prices that depend on the amount of change the voter is asked for by the briber (in the later case of the so-called microbribery).

Of a particular interest and motivation for our work are the work that use integer linear programming models for solving bribery problems or their relaxations in to obtain approximation algorithms. We begin with the work of Keller et al. [48] who presented a PTAS for shift bribery of scoring rules. It is worth mentioning that a factor 2-approximation for the same problem was presented already by Elkind and Faliszewski [24]. In these results one tries to approximate the number of manipulators needed. An orthogonal line of research focuses on minimizing the number of points by which the preferred candidate losses the altered election using a scoring rule [49]; see also Reference [33]. An interesting model of persuasion, which is related to bribery, where in rounds an agent proposes new preference profiles to other voters (who switch to the suggested one if they can benefit from this change) is studied by Hazon et al. [43].

The main motivation for studying (destructive) bribery comes from the study of the so-called margin of victory, or robustness. This paradigm, to the best of our knowledge, was introduced independently by Magrino et al. [54] and Cary [16]. Later Xia [62] studied various (classical) complexity aspects of computing the margin of victory. It should be noted that these papers only deal with unit costs, i.e., the cost of bribery is uniform for all the voters. In the same context Shiryaev et al. [61] study swap bribery as a more fine grained measure. In the multiwinner setting a possible motivation for studying (constructive) bribery might come from the need to measure the candidate success (in the case he or she is not in the winning committee) [36]; see also References [12, 42, 46].

For various other bribery models that have been investigated algorithmically, see, e.g., Baumeister and Rothe [5, Chapter 4.3.5] or Faliszewski and Rothe [35]; see also, e.g., References [3, 59].

Regarding ILPs, tractable fragments include ILPs whose defining matrix is totally unimodular (due to the integrality of the corresponding polyhedra and the polynomiality of linear programming), and ILPs in fixed dimension (due to the algorithms of Lenstra [53] and Kannan [47]). Courcelle's theorem [17] and Freuder's algorithm [40] implies that solving ILPs is fixed-parameter tractable parameterized by the treewidth of the constraint matrix and the maximum domain size of the variables. Ganian and Ordyniak [41] showed fixed-parameter tractability for the combined parameter the treedepth and the largest absolute value in the constraint matrix, and contrasted this with a W[1]-hardness result when treedepth is exchanged for treewidth.

The present article played an important role as a catalyst for focusing attention on n-fold IP and related IPs. By close inspection of the algorithm of Hemmecke et al. [44], we were able to obtain an improved version for the special case of "combinatorial n-fold IPs" that include the presently studied models for $\mathcal{R}$-Multi Bribery [52]. The technical contribution established above in the form of an improved dynamic programming approach, has then led to speed-ups for general n-fold IPs, both in terms of the parameters and the dimension n, culminating in the currently best algorithms due to Eisenbrand et al. [23]. Also, an initial experimental evaluation was carried out by Altmanová et al. [1] suggesting that iterative augmentation algorithms have potential for practical applications, because of certain "adaptivity" properties. The notion of extended n-fold IP was so far not used elsewhere, but it perhaps popularized some modeling tricks used here, see Falisezwski et al. [28].

Organization. In Section 2, we provide the necessary background on the problems that we solve. Then, in Section 3, we define extended n-fold IPs, which later allow for easier problem modeling. We do so in Section 4, where we give extended n-fold IP formulations for several instantiations of the $\mathcal{R}$-Multi-Bribery problem. The complexity lower bounds and hardness results are given in Section 5. We conclude in Section 6.

2 VOTING AND BRIBING PROBLEMS

We give notions for the problems we deal with; for background, we refer to the surveys of Brams and Fishburn [8] and Faliszewski and Rothe [35].

Elections. An election (C, V) consists of a set C of candidates and a set V of voters, who indicate their preferences over the candidates in C. There are many ways in which a voter's preferences can be modeled; throughout this article we use a variant of the ordinal model, where each voter v's preferences are represented via a *preference order* $\succ_v$, which is a total order over C unless stated otherwise. In some problems we study voters who indicate their preferences only for their "top candidates"; we model this with "truncated orders." For an integer $t \in \mathbb{N}$, a preference order $\succ_v$ is *t-top-truncated* if there is a permutation π over $\{1, \ldots, |C|\}$ such that $\succ_v$ is of the form $c_{\pi(1)} \succ_v \cdots \succ_v c_{\pi(t)} \succ_v \{c_{\pi(t+1)}, \ldots, c_{\pi(|C|)}\}$; that is, v is indifferent among the members of the set $\{c_{\pi(t+1)}, \ldots, c_{\pi(|C|)}\}$, which we call *unranked candidates*; we refer to $\{c_{\pi(1)}, \ldots, c_{\pi(t)}\}$ as to the *ranked candidates*. For a ranked candidate c we denote by $\mathrm{rank}(c, v)$ their rank in $\succ_v$; then v's most preferred candidate has rank 1 and their least preferred candidate has rank $|C|$. Also, for t-top-truncated preference orders $\succ_v$ it holds that $\mathrm{rank}(c, v) \leq t$ for all *ranked* candidates $c \in C$. We note here that if $\succ_v$ is a *weak order* (or *bucket order*), i.e., when it is a linear order over disjoint groups of candidates with the voter having no preference over candidates in one group, we may replace it with any linear extension of $\succ_v$ and set the cost of swapping two candidates c, c' to 0 whenever $\mathrm{rank}(c, v) = \mathrm{rank}(c', v)$ in the original order. Independently of voters, for the set of candidates C we also refer to a linear order $\succ_C$ over C as to a *ranking* of C (i.e., ranking is a shorthand for a linear order on candidates). For distinct candidates $c, c' \in C$, we write $c \succ_v c'$ if voter v prefers c over c'. To simplify notation, we sometimes identify the candidate set C with the set $\{1, \ldots, |C|\}$, in particular when expressing permutations over C. All studied problems designate a candidate in C; we always denote it by $c^\star$.

Next, we describe the actions by which we perturb a given election (C, V). Applying a set Γ of actions to (C, V) yields a *perturbed* election that we denote by $(C, V)^\Gamma$. Performing an action incurs a cost; we specify these costs by functions that for each voter $v \in V$ specify their individual cost of performing the action.

2.1 Actions for Manipulation

Swaps. Let (C, V) be an election, let $v \in V$ be a voter, and let $\succ_v$ be their preference order. For candidates $c, c' \in C$, a *swap* $s = (c, c')_v$ means to exchange the positions of c and c' in $\succ_v$; denote the perturbed order by $\succ_v^s$. A swap $(c, c')_v$ is *admissible in* $\succ_v$ if $\mathrm{rank}(c, v) = \mathrm{rank}(c', v) - 1$. A set S of swaps is *admissible in* $\succ_v$ if they can be applied sequentially in $\succ_v$, one after the other, in some order, such that each one of them is admissible. Note that the perturbed vote, denoted by $\succ_v^S$, is independent from the order in which the swaps of S are applied. We also extend this notation for applying swaps in several votes and denote it V^S. We specify v's cost of swaps by a function $\sigma^v : C \times C \to \mathbb{Z}$. A special case of swaps are "shifts," where we want to make $c^\star$ win the perturbed election by shifting them forward in some of the votes, at an appropriate cost, without exceeding a given budget. Shifts can be modeled by swaps only involving $c^\star$.

Push actions. Let (C, V) be an election. In certain voting rules, such as SP-AV or Fallback (to be defined below), each voter $v \in V$ additionally has an *approval count* $a^v \in \{0, \dots, |C|\}$. Voter v's approval count[3] a^v indicates that they approve the top-ranked a^v many candidates in their preference order, and disapprove all others. A "push action" can change a voter's approval count: formally, for voter v and $t \in \{-a^v, \dots, |C| - a^v\}$, a *push action* $p^v = t$ changes their approval count to $a^v + t$. We specify the cost of push actions by a function $\pi^v : \{-a^v, \dots, |C| - a^v\} \to \mathbb{Z}$; we stipulate that $\pi^v(0) = 0$. If a voter v is involved in a swap or a push action, a one-time *influence cost* ι^v occurs.

Control changes. Let (C, V) be an election. We partition the set V into a set V_a of *active* voters and a set V_ℓ of *latent* voters. Only active voters participate in an election, but through a "control change" latent voters can become active or active voters can become latent. (If no partition of V into V_a and V_ℓ is specified, then we implicitly assume that $V = V_a$.)

Formally, a *control change* γ activates some latent voters from V_ℓ and deactivates some active voters from V_a; denote the changed set of voters by $(V_\ell \cup V_a)^\gamma$. We denote the cost of activating voter $v \in V_\ell$ by α^v and the cost of deactivating voter $v \in V_a$ by δ^v.

2.2 Voting Rules

A voting rule $\mathcal{R}$ is a function that maps an election (C, V) to a subset $W \subseteq C$, called the *winners*. We study the following voting rules:

Scoring protocols. A scoring protocol is defined through a vector $\mathbf{s} = (s_1, \dots, s_{|C|})$ of integers with $s_1 \geq \cdots \geq s_{|C|} \geq 0$. For each position $p \in \{1, \dots, |C|\}$, the value s_p specifies the number of points that each candidate c receives from each voter that ranks c as p^{th} best. Any candidate with the maximum number of points is a winner. Examples of scoring protocols include the Plurality rule with $\mathbf{s} = (1, 0, \dots, 0)$, the d-Approval rule with $\mathbf{s} = (1, \dots, 1, 0, \dots, 0)$ with d ones, and the Borda rule with $\mathbf{s} = (|C| - 1, |C| - 2, \dots, 1, 0)$. Throughout, we consider only *natural* scoring protocols for which $s_1 \leq |C|$; this is the case for the aforementioned popular rules.

Bucklin. The *Bucklin winning round* is the (unique) number k such that using the k-approval rule yields a candidate with more than $\frac{n}{2}$ points, but the $(k-1)$-approval rule does not. A *Bucklin winner* is then any candidate with the maximum points (over all candidates) when the k-approval rule is applied.

Condorcet-consistent rules. A candidate $c \in C$ is a *Condorcet winner* if any other $c' \in C \setminus \{c\}$ satisfies $|\{v \in V \mid c \succ_v c'\}| > |\{v \in V \mid c' \succ_v c\}|$. A voting rule is *Condorcet-consistent* if it selects the Condorcet winner in case there is one. Fishburn [37] classified voting rules as C1, C2, or C3, depending on the kind of information needed to determine the winner.[4] For candidates $c, c' \in C$ let $v(c, c')$ be the number of voters who prefer c over c', that is, $v(c, c') = |\{v \in V \mid c \succ_v c'\}|$; we write $c <_M c'$ if c beats c' in a head-to-head contest, that is, if $v(c, c') > v(c', c)$.

C1: *$\mathcal{R}$ is C1* if knowing $<_M$ suffices to determine the winner, that is, for each pair of candidates c, c' we know whether $v(c, c') > v(c', c)$, $v(c, c') < v(c', c)$ or $v(c, c') = v(c', c)$. An example is the Copeland$^\alpha$ rule for a rational number $\alpha \in [0, 1]$, which specifies that for each head-to-head contest between two distinct candidates, if some candidate is preferred by a majority of voters, then they obtain one point and the other candidate obtains zero points, and if a

[3] See Scoring protocols for the definition.

[4] Sometimes the classification (C1, C2, C3) applies to Condorcet-consistent rules only, i.e., voting rules that guarantee to select a Condorcet winner as a winner if such a candidate exists. Here, we follow Reference [64] where this is not required.

tie occurs, then both candidates obtain α points; the candidate with largest sum of points is a winner.

C2: $\mathcal{R}$ *is C2* if it is not C1 and knowing the *exact value of* $v(c, c')$ for all $c, c' \in C$ suffices to determine the winner. Examples are the *Maximin* rule, which declares any candidate $c \in C$ a winner who maximizes $v_*(c) = \min\{v(c, c') \mid c' \in C \setminus \{c\}\}$; and the *Kemeny* rule, which declares any candidate $c \in C$ a winner for whom there exists a ranking $\succ_R$ of C that ranks c first and maximizes the total agreement with voters

$$\sum_{v \in V} \left| \{(c', c'') \mid ((c' \succ_R c'') \Leftrightarrow (c' \succ_v c'')) \ \ \forall c', c'' \in C\} \right|$$

among all rankings.

C3: $\mathcal{R}$ *is C3* if it is neither C1 nor C2. Examples are the *Dodgson* rule, which declares any candidate $c \in C$ a winner for whom a minimum number of swaps make them the Condorcet winner of the manipulated election; and the *Young* rule, which declares any candidate $c \in C$ a winner for whom removing a minimum number of voters from the election makes c the Condorcet winner of the perturbed election.

Additionally, if approval counts are given for each voter, other voting rules are possible:

Sincere-strategy preference-based approval voting (SP-AV). Each candidate c receives a point from every voter v with $\text{rank}(c, v) \leq a^v$. A candidate with maximum number of points is a winner in the election.

Fallback. Delete, for each voter $v \in V$, their unranked candidates (i.e., all c with $\text{rank}(c, v) > a^v$) from its order. Then, use the Bucklin rule, which might fail to determine a winner due to the deletion of unranked candidates; in that case, use the SP-AV rule.

At this point we can formally define the $\mathcal{R}$-Multi-Bribery problem:

$\mathcal{R}$-Multi-Bribery	Parameter: $\lvert C \rvert$
Input:	An election (C, V) with active voters V_a, latent voters V_ℓ and approval counts a^v for $v \in V$, a designated candidate $c^\star \in C$, and swap costs σ^v for $v \in V$, push action costs π^v for $v \in V$, activation costs α^v for $v \in V_\ell$ and deactivation costs δ^v for $v \in V_a$, and a one-time influence cost ι^v.
Task:	Find a set S of admissible swaps, a set P of push actions, and a control change γ of minimum cost so that $c^\star$ is a winner in the election $(C, (((V_a \cup V_\ell)^\gamma)^S)^P)$ under rule $\mathcal{R}$.

3 EXTENDED N-FOLD INTEGER PROGRAMMING

In this section, we discuss the class of n-fold IPs and show how to enhance them to obtain "extended" n-fold IPs.

3.1 n-fold Integer Programs

We begin by defining n-fold IPs. For background, we refer to the books of Onn [58] and De Loera et al. [19].

Let n, r, s, t be positive integers. Given nt-dimensional integer vectors $\mathbf{w}$, $\mathbf{b}$, $\mathbf{l}$, $\mathbf{u}$, an *n-fold IP* problem $(IP)_{E^{(n)}, \mathbf{w}, \mathbf{b}, \mathbf{l}, \mathbf{u}}$ in variable dimension nt is defined as

$$\min \left\{ \mathbf{w}\mathbf{x} \mid E^{(n)} \mathbf{x} = \mathbf{b}, \ \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}, \ \mathbf{x} \in \mathbb{Z}^{nt} \right\}, \tag{1}$$

where

$$E^{(n)} := \begin{pmatrix} D & D & \cdots & D \\ A & 0 & \cdots & 0 \\ 0 & A & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A \end{pmatrix}$$

is an $(r + ns) \times nt$-matrix, $D \in \mathbb{Z}^{r \times t}$ is an $r \times t$-matrix, and $A \in \mathbb{Z}^{s \times t}$ is an $s \times t$-matrix. For numbers, vectors and matrices, we denote by $\langle \bullet \rangle$ the binary encoding length of an object.

Hemmecke et al. [44] developed an iterative augmentation scheme combined with a dynamic program to show the following:

Proposition 3.1 ([44, Thm. 6.2]). *There is an algorithm that, given* $(IP)_{E^{(n)}, \mathbf{w}, \mathbf{b}, \mathbf{l}, \mathbf{u}}$ *with* $\mathsf{a} = \max\{\|D\|_\infty, \|A\|_\infty\}$*, in time* $\mathsf{a}^{O(trs+t^2s)} \cdot O(n^3 \langle \mathbf{w}, \mathbf{b}, \mathbf{l}, \mathbf{u} \rangle)$ *either*

(1) declares the program infeasible or unbounded or
(2) finds a minimizer of it.

The structure of $E^{(n)}$ allows us to divide the nt variables into n *bricks* of size t. We use subscripts to index within a brick and superscripts to denote the index of the brick, i.e., x^i_j is the jth variable of the ith brick with $j \in \{1, \ldots, t\}$ and $i \in \{1, \ldots, n\}$.

3.2 Extended n-fold Integer Programs

We now introduce a class of IPs that we call "extended n-fold IPs." Our motivation for this is to enhance n-fold IPs with "integer programming tricks" that are well known for general IPs; we want to show how to implement them while preserving the structure of n-fold IPs. These tricks will make the application of n-fold IPs more convenient; they include introducing inequalities using slack variables, implementing logical connectives or the bool operation (see below).

That leads us to the following definition.

Definition 3.2. Let $\mathbf{x} = (x^1_1, \ldots, x^1_t, \ldots, x^n_1, \ldots, x^n_t)$ be an nt-dimensional vector of integer variables. Let $B \in \mathbb{Z}$, $(b^1, \ldots, b^n) \in \mathbb{Z}^n$, and $(a_1, \ldots, a_t) \in \mathbb{Z}^t$. We say that

$$\sum_{i=1}^{n} \sum_{j=1}^{t} a_j x^i_j = B$$

is a *globally uniform constraint* and that

$$\sum_{j=1}^{t} a_j x^i_j = b^i, \quad i = 1, \ldots, n$$

is a *locally uniform constraint*. We stress that (in both cases) the coefficients a_j are the same regardless of the index i. To be more precise: For each locally uniform constraint there are n *invocations* (i.e., one per a brick) that have exactly the same coefficients of the left-hand side (i.e., $a^i_j = a^k_j$ for $i, k \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, t\}$) but may differ in their right-hand sides, that is, we can have $b^i \neq b^k$ for $i, k \in \{1, \ldots, n\}$.

Observe that every n-fold IP consists of box constraints (i.e., lower and upper bounds represented by vectors $\mathbf{l}$ and $\mathbf{u}$), a weight vector $\mathbf{w}$, and collections of globally and locally uniform constraints. From now on we call the problem (1) an *n-fold IP in standard form*.

Definition 3.3. An *extended n-fold IP* is a collection of locally and globally uniform constraints that are additionally allowed to contain

- inequalities $<, \leq, >, \geq$,
- negation ($\neg$) and logical disjunction $\vee$ with standard interpretation if applied to binary arguments, and undefined otherwise.

Additionally, we introduce two collections of operations $\text{bool}_m, \text{sign}_m$ for every positive integer m:

$$\text{bool}_m(x) = \begin{cases} 0, & \text{if } x = 0, \\ 1, & \text{if } x \neq 0 \text{ and } -m \leq x \leq m, \\ \text{undefined}, & \text{otherwise;} \end{cases}$$

$$\text{sign}_m(x) = \begin{cases} 0, & \text{if } x = 0, \\ 1, & \text{if } 1 \leq x \leq m, \\ -1, & \text{if } -m \leq x \leq -1, \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

Note that the constraints are still required to be uniform. Formally, define an expression recursively as follows: an *atomic expression* (or, an *atom*) is a linear combination of variables $\sum_j a_j x_j$ or a constant b, and an *expression* is formed by taking a negation, bool_m, or sign_m of an expression, or by taking a disjunction, or $\heartsuit \in \{=, <, \leq, \geq, >\}$ of two expressions. A *constraint* is of the form $f \heartsuit g$ where $\heartsuit \in \{=, <, \leq, \geq, >\}$ and f, g are expressions. (Hence, note that $x < y$ can be viewed both as an expression and a constraint; if it appears as a constraint, then it enforces $x < y$, whereas if it is an expression, then it appears as a part of some constraint, e.g., $x < y \vee x > y + 2$.) A *globally uniform expression* is one whose atoms have the form $\sum_{i=1}^{n} \sum_{j=1}^{t} a_j x_j^i$, and a *locally uniform expression* is one whose atoms have the form $\sum_{j=1}^{t} a_j x_j^i$, for all $i = 1, \ldots, n$ (as before, each locally uniform expression has *n invocations*). For technical reasons that will become clear later, for an expression x of the form $x = f \heartsuit g$ (where $\heartsuit \in \{<, \leq, =, \geq, >\}$) it holds that $x \leq m$ if and only if $f - g \leq m$; note that both f and g may be composed in a similar way and here we measure the domain of the fully expanded expression. An expression is *valid* if no expression used during the construction results in undefined). For example, if f is not a binary expression, then $f \vee g$ is undefined, and if f may attain values larger than m, then $\text{bool}_m(f)$ is undefined.

An extended n-fold IP is *valid* if the results of all of its constraints are valid. The two important parameters of a valid extended n-fold IP are

extended width is the number of inequalities, logical operations, bool_m, and sign_m operations (counting them only once for all n invocations of each locally uniform constraint or expression); and

height is the maximum integer m occurring in any of its bool_m and sign_m operations.

The notion of height applies naturally also to expressions and constraints: the *height of an expression* is the maximum m appearing in any bool_m or sign_m operation contained in it, and the *height of a constraint* is the height of the expression on the left-hand side. Observe that if $\text{bool}_m(e)$ is defined for a given expression e (i.e., $-m \leq e \leq m$), then $\text{bool}_{m+1}(e)$ is defined; note that the same holds for sign_m and sign_{m+1} as well. Thus, we may actually require all parameters m in the above definition to be the same value.

For any integer $m \in \mathbb{N}$, the expression $x \neq_m y$ is a shorthand for $\text{bool}_m(x - y)$, and the constraint $x \neq_m y$ is a shorthand for $\text{bool}_m(x - y) = 1$; the equivalence is clear. The distinction between $f \heartsuit g$ being a constraint or an expression will always be clear from the context.

Observation 1 (Folklore; see, e.g., Enderton [26]). *It is possible to use all logical connectives (i.e., $\{\wedge, \ldots\}$) with binary expressions in an extended n-fold IP.*

The following theorem deals with how many auxiliary variables and constraints are needed to convert an extended n-fold IP into standard form.

Theorem 3.4. *Let I be a valid extended n-fold IP with nt variables, r globally uniform constraints, s locally uniform constraints, largest absolute coefficient value a, extended width w, and height M. There is an algorithm that, given I, in time $O(ntw(r+s))$ constructs a standard n-fold IP I' with nt' variables, r' globally uniform constraints, s' locally uniform constraints, and largest absolute coefficient value a' such that*

- $t' = t + O(w)$,
- $r' = r$,
- $s' = s + O(w)$, *and*
- $\mathsf{a}' = \max(\mathsf{a}, M)$.

Thus, I can be solved, using the algorithm of Proposition 3.1, in time $(\mathsf{a}')^{\omega} \cdot n^3 \langle \mathbf{w}, \mathbf{b}, \mathbf{l}, \mathbf{u} \rangle$, where $\omega = O((t+w)(s+w)r + (t+w)^2(s+w))$.

Before we move to the proof of Theorem 3.4, we would like to present a rather simple example of its use.

An example of an extended n-fold IP. We focus on a single brick, that is, we work with variables $x_1, \ldots, x_m$ instead of $x^i_1, \ldots, x^i_m$. We want m variables $x_1, \ldots, x_m$ (in each brick) describing a permutation π of $\{1, \ldots, m\}$. We do this in such a way that $x_j = \pi(j)$. That is equivalent to requiring $x_j \neq x_k$ for all $j \neq k$ and $x_j \in \{1, \ldots, m\}$ for all $j \in [m]$. Note that to use a non-equality expression "$e \neq f$" we need to determine the largest possible difference $|e - f|$. In this case it is m, and thus we express the aforementioned conditions by the following $\binom{m}{2}$ locally uniform constraints

$$x_j \neq_m x_k \qquad \text{for all } j, k \in [m], j < k \tag{2}$$

and m box constraints

$$1 \leq x_j \leq m \qquad \text{for all } j \in [m]. \tag{3}$$

Note that the above expression $x_j \neq_m x_k$ is valid, since we have $-m \leq x_j - x_k \leq m$ due to the box constrains (3).

To take the advantage of handling locally uniform expression(s) we now switch our view. Assume we are given n permutations $\pi^1, \ldots, \pi^n : [m] \to [m]$, one for each brick. Now we want to compare the ith brick permutation $x^i_1, \ldots x^i_m$ with $\pi^i(1), \ldots, \pi^i(m)$ and determine which indices are inverted, that is, when it happens that $x^i_j < x^i_k$ and $\pi^i(j) > \pi^i(k)$. In other words, we want to determine when the sign of $(x^i_j - x^i_k)$ equals the sign of $(\pi^i(k) - \pi^i(j))$. So, for each $j < k$ we add a new binary indicator variable $y^i_{j,k}$ as follows:

$$y^i_{j,k} = \mathrm{bool}_2\left(\mathrm{sign}_m\left(x^i_j - x^i_k\right) = \mathrm{sign}_m\left(\pi^i(k) - \pi^i(j)\right)\right). \tag{4}$$

Notice that $\mathrm{sign}_m(\pi^i(k) - \pi^i(j))$ is a constant, we denote it $\pi^i_{k,j}$, so the right-hand side of Equation (4) translates to

$$\mathrm{bool}_2\left(\mathrm{sign}_m(x^i_j - x^i_k) - \pi^i_{k,j}\right);$$

assuming that an expression is locally uniform if it only differs across bricks in its constants (as we will show later), this is indeed a locally uniform expression.

3.3 Proof of Theorem 3.4

Proof of Theorem 3.4. We will prove Theorem 3.4 by exhaustively applying a set of rewriting rules. These rewriting rules are applied to expressions and constraints containing logical operations, the bool_m and the sign_m operations, and inequalities. We will use the letters e, f, and g for expressions, and analogously $f \heartsuit g$ with $\heartsuit \in \{=, <, \leq, >, \geq\}$ for expressions. Furthermore, we always create an auxiliary variable x_e and assign to it the value an expression e takes by adding an auxiliary locally uniform constraint "$x_e = e$." A rewriting rule can be applied anytime all its operands are already represented by such a variable.

To determine the parameters r', s', and t' of the resulting n-fold IP in standard form, we consider the "s-increase" $\Delta s(e)$ of an expression e, which is the number of auxiliary equations required to rewrite e into the standard form. Similarly, the "t-increase" $\Delta t(e)$ of e is the number of auxiliary variables needed to express e, and analogously for globally uniform constraints and $\Delta r(e)$. We note the s- and t-increase of each rule after defining it.

Rewriting a locally uniform expression e to the standard format. Rewriting a locally uniform expression e in some locally uniform constraint(s) means replacing it with a new variable x_e and adding auxiliary locally uniform constraints (and possibly some new auxiliary variables). These constraints assure that the variable x_e will carry the desired meaning (i.e., the value of e). The result is that every expression e is rewritten to the standard format. In this phase we may still be adding constraints that are not in the standard format (e.g., contain inequalities) as they will be dealt with by subsequent applications of the rewriting rules.

- $e ::= f \vee g$: We assume (since I is valid) that we have two binary variables x_f and x_g such that $x_f = 1$ if the expression f holds (i.e., if it evaluates to true) and $x_f = 0$ otherwise; the same holds for the variable x_g and the expression g. Now, we introduce a new auxiliary binary variable y_e (i.e., $0 \leq y_e \leq 1$) and the constraint

$$2x_e = x_f + x_g + y_e. \tag{5}$$

  We claim that Equation (5) ensures that $x_e = 1$ if and only if $x_f = 1$ or $x_g = 1$ or equivalently that $x_e = 1$ if and only if $x_f + x_g \geq 1$. Assume that $x_f + x_g \in \{1, 2\}$ and define $y_e = 2 - x_f - x_g$; note that in this case $0 \leq y_e \leq 1$, i.e., y_e fulfills its box constraints. It is straightforward to verify that in this case $x_e = 1$. However, if $x_f + x_g = 0$, then, since both x_e and y_e are binary variables, to satisfy Equation (5) we must set $x_e = y_e = 0$.

  Dealing with uniformity. Note that in the above discussion we focused solely on a single brick. Formally, we should have discussed that, since the expression $e ::= f \vee g$ is locally uniform, we have binary variables x_f^i and x_g^i for every $i = 1, \ldots, n$. We introduce auxiliary binary variables $y_e^1, \ldots, y_e^n$ and binary variables $x_e^1, \ldots, x_e^n$ (i.e., pair of these for each brick). Then, we add the locally uniform constraint

$$2x_e^i = x_f^i + x_g^i + y_e^i, \qquad i = 1, \ldots, n.$$

  Now, it is straightforward to verify that we have maintained the local uniformity, since all the constraints "look the same." The key property here is that we have a uniform approach to every brick; thus, it is sufficient if all our rewriting rules do not depend on a particular brick. We believe that the first rewriting approach is easier to follow and, since the later can be obtained in a straightforward way from it, we only present the first for all the remaining rewriting rules.

  Parameter changes. Consider the parameters $\Delta s(e)$ and $\Delta t(e)$. We added one auxiliary variable y_e, and the variable x_e to store the result of $f \vee g$. Hence, $\Delta t(e) = \Delta t(f) + \Delta t(g) + 2$. Similarly, $\Delta s(e) = \Delta s(f) + \Delta s(g) + 1$, as the only new condition

is condition (5), since $0 \leq y_e \leq 1$ is a box constraint and thus we do not count it here (same for $0 \leq x_e \leq 1$).

- $e ::= \neg f$ is an expression equivalent to the locally uniform constraint $x_e = 1 - x_f$, where x_e is a new binary variable; note that we again use the binary variable x_f.
  Then $\Delta s(e) = \Delta s(f) + 1$, as there is one new constraint $x_e = 1 - x_f$, and $\Delta t(e) = \Delta t(f) + 1$, as we have added a new variable x_e.
- $e ::= \text{bool}_m(f)$ and $e ::= \text{sign}_m(f)$ (see also a demonstration in Section 3.4 below): We first compute the smallest possible positive integers $L, U \in \mathbb{N}$ such that $-L < f < U$ holds. Note that we have $\max\{L, U\} \leq m$, since the given instance I is valid. The subscript m in $\text{bool}_m(f)$ signifies that we need to introduce coefficients (upperbounded by m) into the (new) system. Let $y_e, z_e \in \{0, 1\}$ be two new auxiliary binary variables. We first add some constraints so that $y_e = 1$ if and only if $x_f \geq 0$, and $z_e = 1$ if and only if $x_f \leq 0$ (note that m is a constant and thus all constraints below are linear and remain uniform):

$$1 + x_f \leq m \cdot y_e \leq m + x_f, \tag{6}$$

$$1 - x_f \leq m \cdot z_e \leq m - x_f. \tag{7}$$

  Note that Equations (6) and (7) do not depend on L, U, and thus we again obtain locally uniform constraints. To see that the above holds we distinguish two cases; first for y_e:

  $x_f \geq 0$ From the definition of m it follows that $1 \leq 1 + x_f \leq U \leq m$. Thus, to satisfy the first (i.e., the leftmost) inequality in Equation (6) we must set $y_e = 1$. Furthermore, we have that $m + x_f \geq m = m \cdot y_e$ and thus Equation (6) is satisfied.

  $x_f < 0$ From the definition of m it follows that $0 \geq 1 + x_f > -m$; thus if $y_e \in \{0, 1\}$, then the first inequality of Equation (6) is satisfied. Thus, to satisfy the second inequality in (6) we must set $y_e = 0$, since $m + x_f < m$; furthermore, we have $m + x_f > 0 = m \cdot y_e$ and thus this setting is valid. All in all $y_e = 0$ is the only solution to Equation (6).

  Now, we again distinguish two cases for z_e:

  $x_f \leq 0$ We have $1 - x_f \geq 1$. Thus, to satisfy the first inequality in Equation (7) we have to set $z_e = 1$. Now, since $m - x_f \geq m = m \cdot z_e$, we have satisfied (the second inequality in) Equation (7).

  $x_f > 0$ We have $1 - x_f \leq 0$ and thus the first inequality holds for any $z_e \in \{0, 1\}$. However, we have $m - x_f < m$, and thus only $z_e = 1$ satisfies the second inequality in (7); note that $m - x_f > 0$.

  Now if $e ::= \text{bool}_m(f)$, then we additionally to the above locally uniform constraints add the locally uniform expression $x_e = \neg(y_e \wedge z_e)$. Note that this additional locally uniform expression can be directly rewritten to the standard form using the above rules. If $e ::= \text{sign}_m(f)$, then we additionally to the above locally uniform constraints add the locally uniform constraint $x_e = y_e - z_e$.
  Then (in both cases) we have $\Delta s(e) = \Delta(f) + O(1)$ and $\Delta t(e) = \Delta t(f) + O(1)$, as we only add constantly many new (and auxiliary) variables and constraints.
- $e ::= \text{bool}_m(f \heartsuit g) \quad \Rightarrow$
  — $\heartsuit$ is "=": $x_e = \neg\text{bool}_m(x_f - x_g)$ (which can be rewritten using the above rewriting rules). Clearly, $\text{bool}_m(x_f - x_g) = 1$ if and only if $(x_f - x_g) = 0$ if and only if $x_f = x_g$. Since I is valid, we have $-m < x_f - x_g < m$ and thus the expression $x_e = \neg\text{bool}_m(x_f - x_g)$ is valid.
  — $\heartsuit$ is ">": $x_e = \text{bool}_2(\text{sign}_m(x_f - x_g) = 1)$, which, using similar arguments as above, is equivalent and valid.
  — $\heartsuit$ is "$\geq$": $x_e = \text{bool}_m(x_f > x_g) \vee \text{bool}_m(x_f = x_g)$; which follows directly from the above.
  And analogously when $\heartsuit$ is "<" or "$\leq$".

Then $\Delta s(e) = \Delta s(f) + \Delta s(g) + O(1)$ and $\Delta t(e) = \Delta t(f) + \Delta t(g) + O(1)$, since the overhead of the used operations is only $O(1)$.

Rewriting a locally uniform constraint $e ::= f \heartsuit g$ to standard format.

- when $\heartsuit$ is "=": $f = g \quad \Rightarrow \quad x_f - x_g = 0$.
  Then $\Delta s(e) = \Delta s(f = g) = \Delta s(f) + \Delta s(g)$ and $\Delta t(e) = \Delta t(f = g) = \Delta t(f) + \Delta t(g)$.
- when $\heartsuit$ is not "=", intuitively we want to add a slack variable and rewrite $f \heartsuit g \Rightarrow$ into a locally uniform constraint $x_f - x_g + y_e = 0$ with the variable y_e having an upper bound u_{y_s} and a lower bound l_{y_s} set as follows:
  $-l_{y_s} = 0$ and $u_{y_s} = Q_e$ when $\heartsuit$ is "$\leq$",
  $-l_{y_s} = 1$ and $u_{y_s} = Q_e$ when $\heartsuit$ is "$<$",
  $-l_{y_s} = -Q_e$ and $u_{y_s} = 0$ when $\heartsuit$ is "$\geq$", and
  $-l_{y_s} = -Q_e$ and $u_{y_s} = -1$ when $\heartsuit$ is "$>$";
  where $Q_e = \max\{\|\mathbf{l}\|_\infty, \|\mathbf{u}\|_\infty\} \cdot \mathsf{a} \cdot nt$ stands for a sufficiently large number.
  Then, $\Delta s(e) = \Delta s(f) + \Delta s(g)$ and $\Delta t(e) = \Delta t(f) + \Delta t(g) + 1$.

Rewriting a globally uniform constraint $e ::= f \heartsuit g$ to standard format. First, we rewrite any logical operations and bool_m and sign_m operations in e and f using the same rules as above, that is, by adding auxiliary variables and locally uniform constraints. Then, what remains is to deal with the inequalities $\heartsuit \in \{<, \leq, >, \geq\}$. We use slack variables as before, but since we cannot add just one variable without breaking the n-fold format, we instead add n new variables and "disable" all but one of them using the lower and upper bounds:
$e ::= f \heartsuit g \quad \Rightarrow \quad x_f - x_g + \sum_{i=1}^{n} s_e^i = 0$ with s_e^i for $i = 1, \ldots, n$ being n new auxiliary variables with lower and upper bounds $l_{s_e^i} = u_{s_e^i} = 0$ for $1 < i \leq n$ and with

- $l_{s_e^1} = 0$ and $u_{s_e^1} = \infty$ when $\heartsuit$ is "$\leq$",
- $l_{s_e^1} = 1$ and $u_{s_e^1} = \infty$ when $\heartsuit$ is "$<$",
- $l_{s_e^1} = -\infty$ and $u_{s_e^1} = 0$ when $\heartsuit$ is "$\geq$", and
- $l_{s_e^1} = -\infty$ and $u_{s_e^1} = -1$ when $\heartsuit$ is "$>$".

Then, $\Delta t(e) = \Delta t(f) + \Delta t(g)$ if $\heartsuit$ is "=" (since clearly this can be done even without introducing the auxiliary variables) and $\Delta t(e) = \Delta t(f) + \Delta t(g) + 1$, otherwise.

Finishing the proof. It remains to compute the parameters r', s', t', and a' of the instance I' we have created by exhaustive application of rewriting rules to the given instance I. Let $\mathcal{L}$ be the set of locally uniform constraints of I' and $\mathcal{G}$ be the set of globally uniform constraints of I'. Then

- $t' = t + \sum_{e \in \mathcal{L}} \Delta t(e) + \sum_{e \in \mathcal{G}} \Delta t(e) = t + O(w)$,
- $s' = s + \sum_{e \in \mathcal{L}} \Delta s(e) = s + O(w)$,
- $r' = r$, since we have merely added slack variables into globally uniform constraint of I,
- all coefficients are bounded in absolute value by $\mathsf{a}' = \max\{\mathsf{a}, M\}$, since we have only introduced new large coefficients via the locally uniform expressions bool_m and sign_m and those are upper bounded by M.

This concludes the proof of Theorem 3.4. □

3.4 A Demonstration of the Rewriting Process

In this section, we demonstrate the rewriting process as presented in the proof of Theorem 3.4 on the constraints (2) (while keeping the box constraints (3) in mind). To that end, our goal is to rewrite

$$x_j \neq_m x_k \qquad \text{for all } j, k \in [m], j < k,$$

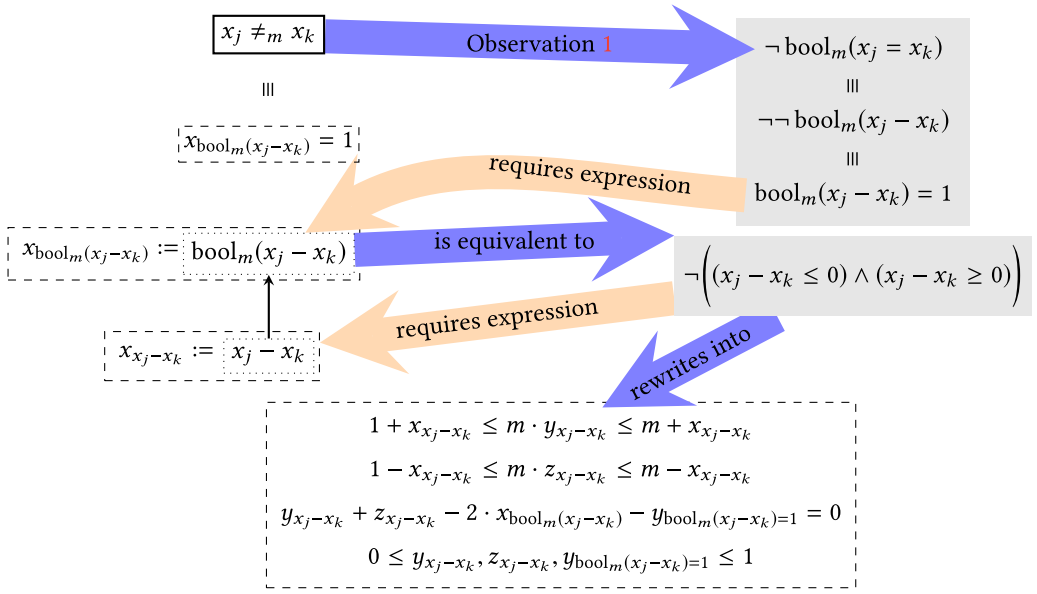

Fig. 1. An illustration of the rewriting process for Equation (3). The thick box $x_j \neq_m x_k$ represents the start of the rewriting process. The dashed boxes represent the result of the rewriting process, i.e., the locally uniform condition.

which, as we have already seen, is equivalent to

$$\mathrm{bool}_m(x_j - x_k) = 1 \qquad \text{for all } j, k \in [m], j < k.$$

We refer to Figure 1 for an illustration. We fix j and k and apply the rewriting rules (i.e., follow the above proof). The resulting standard n-fold IP will contain the following constraints[5] (for brevity we omit rewriting inequalities by slack variables as this is standard):

$$\begin{aligned}
x_{x_j - x_k} &= x_j - x_k \\
1 + x_{x_j - x_k} &\leq m \cdot y_{x_j - x_k} \leq m + x_{x_j - x_k} \\
1 - x_{x_j - x_k} &\leq m \cdot z_{x_j - x_k} \leq m - x_{x_j - x_k} \\
0 &\leq y_{x_j - x_k}, z_{x_j - x_k} \leq 1 \\
2x_{\mathrm{bool}_m(x_j - x_k)} &= y_{x_j - x_k} + z_{x_j - x_k} - y_{\mathrm{bool}_m(x_j - x_k)=1} \\
x_{\mathrm{bool}_m(x_j - x_k)} &= 1.
\end{aligned}$$

Remark. Naturally, we ask if the bool() operation can be implemented *without* introducing a number a depending on the lower and upper bounds, as a becomes the base of the runtime in Theorem 3.1. One can show that such dependence is necessary (the proof is deferred to Section 5.1):

Lemma 3.5. *Unless* FPT = W[1], *the* bool() *operation cannot be expressed in n-fold IP format by introducing only $f(k)$ new variables and numbers bounded by $f(k)$, for any computable function f and $k = \max\{r, s, t\}$. Moreover, binary n-fold IP is weakly* NP*-hard even when $r = t = 1$ and $s = 0$.*

[5]Note that the fourth line of inequalities represents box constrains for the newly added variables.

4 SINGLE-EXPONENTIAL ALGORITHMS FOR VOTING AND BRIBING

We now establish a formulation of $\mathcal{R}$-Multi-Bribery as an n-fold IP, for various rules $\mathcal{R}$. To this end, we first describe the part of the IP that is common to all such rules, in Section 4.1. Thereafter, in Section 4.2 we add the parts of the formulation that depend on $\mathcal{R}$.

4.1 General Setup

Given an instance (C, V) of $\mathcal{R}$-Multi-Bribery, we construct an n-fold IP whose variables describe the situation after bribery actions (swaps, push actions, control changes; if allowed) were performed. From these variables we also derive new variables to express the cost function. In the following we always describe the variables and constraints added per voter, and there is one brick per voter. So in what follows we fix a voter $v \in V$.

Swaps. We describe the preference order with swaps S applied by variables x_c^v for $c \in C$ with the intended meaning $x_c^v = \mathrm{rank}(c, v)^S$. We stress here that the ranking according to values of x_c^v is the one in the altered elections. Recall that constraints (2) and (3) enforce that $(x_1^v, \ldots, x_{|C|}^v)$ is a permutation of C; we add them to the program (we set $m = |C|$).

To express the swaps performed by S, for each pair of candidates $c, c' \in C$ we introduce binary variables $s_{\{c,c'\}}^v$ so that $s_{\{c,c'\}}^v = 1$ if and only if c and c' are swapped. We need an observation that follows from a result of Elkind et al. [25, Proposition 1].

Observation 2. *For complete preference orders $\succ, \succ'$, the admissible set S of swaps such that $\succ' = \succ^S$ is uniquely given as the set of pairs (c, c') for which either $c \succ c' \wedge c' \succ' c$, or $c' \succ c \wedge c \succ' c'$.*

Thus, we only need to set constraint (4) from Section 3.2 with $\pi_c^v = \mathrm{rank}(c, v)$. Further, for each pair $c, c' \in C$ of candidates we introduce a variable $x_{(c,c')}^v$ that takes value 1 if $c \succ_v^S c'$, and value 0 otherwise. To that end, we add the following local uniform expressions (i.e., for each $v \in V$)

$$x_{(c,c')}^v = \mathrm{bool}_{|C|}\left(x_c^v < x_{c'}^v\right), \qquad c, c' \in C, \tag{8}$$

$$s_{\{c,c'\}}^v = \mathrm{bool}_2\left(x_{(c,c')}^v = \pi_{(c,c')}^v\right), \qquad \forall c, c' \in C, \tag{9}$$

where $\pi_{(c,c')}^v$ is a constant equal to 1 if and only if $c \succ_v c'$; and both are 0, otherwise. Observe that the above conditions are equivalent to the intended meaning of the variables $x_{(c,c')}^v$ and $s_{\{c,c'\}}^v$. Note that the above is locally uniform expression, since all the above differ only in the right-hand sides of the local uniform constraints

$$x_{x_{(c,c')}^v = \pi_{(c,c')}^v}^v = x_{(c,c')}^v - \pi_{(c,c')}^v$$

needed to compute the above values. Furthermore, both Equations (8) and (9) are valid. To see this observe that $-|C| < x_c^v - x_{c'}^v < |C|$, since we have $1 \leq x_c^v, x_{c'}^v \leq |C|$, and $-2 < x_{(c,c')}^v - \pi_{(c,c')}^v < 2$, since $x_{(c,c')}^v$ is a binary variable and $\pi_{(c,c')}^v \in \{0, 1\}$, i.e., $x_{x_{(c,c')}^v \pi_{(c,c')}^v}^v \in \{-1, 0, 1\}$.

Push actions. To indicate push actions, we introduce binary variables $p_{-|C|}^v, \ldots, p_{|C|}^v$, where $p_0^v = 1$ means no change, $p_j^v = 1$ means push action $p^v = j$. We set the lower and upper bounds to ensure that $p_j^v = 0$ for all $j \notin \{-a^v, |C| - a^v\}$. Finally, we introduce a variable $x_\alpha^v \in \{1, \ldots, |C|\}$

indicating v's approval count after the push action:

$$\sum_{j=-|C|}^{|C|} p_j^v = 1,$$

$$x_\alpha^v = a^v + \sum_{j=-|C|}^{|C|} j p_j^v.$$

Influence bit. To model certain variants of the problem, such as $\mathcal{R}$-Manipulation, we need an auxiliary "influence bit." We introduce a binary variable x_ι taking value 1 if a swap or a push action is performed, and value 0 otherwise:

$$x_\iota^v = \text{bool}_{|C|^2}\left(\sum_{c,c' \in C} s_{\{c,c'\}}^v + \sum_{j \neq 0} p_j^v\right).$$

The above condition is valid for nontrivial instances with $|C| \geq 2$, since $\sum_{c,c' \in C} s_{\{c,c'\}}^v \leq |C|(|C|-1)$, $\sum_{j \neq 0} p_j^v \leq \sum_{j=-|C|}^{|C|} p_j^v = 1$ and $|C|(|C|-1)+1 < |C|^2$ (if $|C| \geq 2$).

Control changes. We introduce two binary variables x_a^v and x_ℓ^v such that $x_a^v = 1$ and $x_\ell^v = 0$ if voter v is active, and $x_a^v = 0$ and $x_\ell^v = 1$ if voter v latent:

$$x_a^v + x_\ell^v = 1.$$

We will also frequently use the following variable-splitting trick:

Lemma 4.1. *Let x be an integral variable with lower bound ℓ and upper bound u and let z be a binary variable. Then one can introduce a variable x^z, an auxiliary variable, and three locally uniform constraints of height at most $m = u - \ell$ such that $x^z = x$ if $z = 1$ and $x^z = 0$ if $z = 0$.*

Proof. Assume, without loss of generality, that x is normalized, that is, $\ell = 0$. If this is not the case, we replace x with $(\tilde{x} + \ell)$ in all constraints and set box constraints for $\tilde{x}$ to $0 \leq \tilde{x} \leq u - \ell$. Clearly, this substitution yields an equivalent integer program (and maintains uniformity of the IP).

We first add constrains $0 \leq x^z \leq uz$ (note that the later condition is *not* a box condition, since z is a variable). Now, if $z = 0$, then $x^z = 0$, and otherwise ($z = 1$) we have $\ell \leq x^z \leq u$. We now essentially repeat this trick with a negation of z for a new variable $x^{\neg z}$, that is, we add constraints $0 \leq x^{\neg z} \leq u(1-z)$. Finally, we add the constraint $x^z + x^{\neg z} = x$, which finishes the construction, since by the above discussion we know one of the variables x^z or $x^{\neg z}$ must be set to 0 and thus the other must be set to the same value as x. □

Objective function. Finally, *collectively for all voters* the linear objective function is as follows:

$$w(\mathbf{x}, \mathbf{s}, \mathbf{p}) = \sum_{v \in V}\left[\left(\sum_{c,c' \in C} \sigma^v(c,c') s_{(c,c')}^v\right) + \left(\sum_{j=-|C|}^{|C|} \pi^v(j) p_j^v\right) + \iota^v x_\iota^v + \alpha^v x_a^v + \delta^v x_\ell^v\right].$$

Observe that the maximum coefficient of the objective function, i.e., $\|\mathbf{w}\|_\infty$, is the maximum of all the cost functions σ^v, π^v, ι^v, α^v, and δ^v across all their arguments. So far we have introduced $O(|C|^2)$ variables and imposed $O(|C|^2)$ constraints on them (per brick). The largest coefficient introduced in a constraint is $|C|$, and we have already used the bool_M operation with $M = |C|^2$.

4.2 Voting Rules

Now we describe the part specific to the voting rules. A voting rule $\mathcal{R}$ is incorporated in the IP in two steps. First, optionally, new variables are derived using locally uniform constraints. Then, globally uniform constraints are imposed.

Often we can only set up the IP knowing certain facts about how the winning condition is satisfied. We guess those facts, construct the IP, solve it and remember the objective value. Finally, we choose the minimum over all guesses.

(R1) Scoring protocol $\mathbf{s} = (s_1, \ldots, s_{|C|})$. We introduce variables τ_c^v for the number of points that voter v gives candidate c:

$$\tau_c^v = \sum_{k=1}^{|C|} s_k \mathrm{bool}_{|C|}(x_c^v = k).$$

Again, the above is valid, since $|C| < x_c^v - k < |C|$ as we have $1 \leq x_c^v, k \leq m$. Then, an "active" copy τ_c^{va} of each variable τ_c^v is created such that we can disregard the contribution of latent voters. To do this we use Lemma 4.1 with $x := \tau_c^v$, $z := x^a$, and $x^z := \tau_c^{va}$ for every $c \in C$. Then we add the following globally uniform constraints specifying that the score received by $c^\star$ is greater than the score received by any other candidate:

$$\sum_{v \in V} \tau_c^{va} \; < \; \sum_{v \in V} \tau_{c^\star}^{va} \text{ for } c \in C \setminus \{c^\star\}.$$

(R2) Any C1 rule $\mathcal{R}$. We guess the resulting $<_M$ such that $c^\star$ is a winner with respect to $\mathcal{R}$; there are $O(3^{|C|^2})$ guesses. From $<_M$ we can infer for any pair c, c' of distinct candidates, whether $v(c, c') > v(c', c)$, $v(c', c) > v(c, c')$, or $v(c, c') = v(c', c')$. With this knowledge, we add the following constraints, where again variables with an a in the superscript stand for the active parts:

$$\sum_{v \in V} x_{(c,c')}^{va} > \sum_{v \in V} x_{(c',c)}^{va} \qquad \text{if } c <_M c',$$

$$\sum_{v \in V} x_{(c,c')}^{va} = \sum_{v \in V} x_{(c',c)}^{va} \qquad \text{if } v(c, c') = v(c', c).$$

(R3) Maximin rule. For $c^\star$ to be a winner with the maximin rule means that there is a number $B \in \{0, 1, \ldots, |V|\}$ such that $v_*(c^\star) = B$, while for all $c \in C \setminus \{c^\star\}$, $v_*(c) < B$. That, in turn, means, that for every candidate $c \neq c^\star$ there is a candidate c' such that $v(c, c') < B$. (Recall that $v_*(c) = \min\{v(c, c') \mid c' \in C \setminus \{c\}\}$.) Guess B and c' for every c; there are at most $n \cdot |C|^2$ guesses. This implies that for every candidate $c \neq c^\star$ there is a candidate $d(c)$ (the defeater of c) such that $v(c, d(c)) < B$. All in all B and $d(c)$ for every $c \in C \setminus \{c^\star\}$, that is, we guess a mapping $d : C \setminus \{c^\star\} \to C$; there are at most $n \cdot (|C| - 1)^{|C|}$ guesses. Then add the following constraints:

$$\sum_{v \in V} x_{(c^\star,c)}^{va} \geq B \qquad c \in C \setminus \{c^\star\},$$

$$\sum_{v \in V} x_{(c,d(c))}^{va} < B \qquad c \in C \setminus \{c^\star\}.$$

(R4) Bucklin. To determine the control action γ, we guess the number $\left|V_a^\gamma\right| \in \{1, \ldots, |V|\}$ of active voters and set

$$\sum_{v \in V} x_a^v = \left|V_a^\gamma\right|.$$

Then, we guess the winning round k and observe that the winning score will be larger than $|V_a^\gamma|/2$. Altogether, there are $O(|C||V|)$ guesses. Similarly to scoring protocols, we introduce variables τ_c^v (number of points for a candidate c in k-approval) and $\tilde{\tau}_c^v$ (number of points for a candidate c in $(k-1)$-approval). Again, we consider only the active parts τ_c^{va} of τ_c^v and $\tilde{\tau}_c^{va}$ of $\tilde{\tau}_c^v$:

$$\begin{aligned} \tau_c^v &= \mathrm{bool}_{|C|}\,(x_c^v < k) && c \in C, \\ \tilde{\tau}_c^v &= \mathrm{bool}_{|C|}\,(x_c^v < k-1) && c \in C. \end{aligned}$$

As before, the validity of this expression follows directly from $1 \le x_c^v, k \le |C|$. Then, the winning condition is expressed as:

$$\begin{aligned} &\sum_{v\in V} \tau_{c^\star}^{va} > \left|V_a^\gamma\right|/2 && \\ &\sum_{v\in V} \tau_c^{va} \le \sum_{v\in V} \tau_{c^\star}^{va} && c \in C \setminus \{c^\star\} \\ &\sum_{v\in V} \tilde{\tau}_c^{va} \le \left|V_a^\gamma\right|/2 && c \in C. \end{aligned}$$

(R5) SP-AV. In SP-AV, each candidate c receives a point if it ranks above the approval count. As before, we introduce variables τ_c^v for points received by a candidate c and split them into active and latent (again using Lemma 4.1). We add the following constraints:

$$\begin{aligned} \tau_c^v &= \mathrm{bool}_{|C|}(x_c^v \le x_\alpha^v), && c \in C, \\ \sum_{v\in V} \tau_c^{va} &\le \sum_{v\in V} \tau_{c^\star}^{va} && c \in C \setminus \{c^\star\}. \end{aligned}$$

Those expressions are valid, as $1 \le x_c^v, x_\alpha^v \le |C|$.

(R6) Fallback. In the Fallback rule, the non-approved candidates are discarded, the Bucklin rule is applied and if it fails to select a winner, the SP-AV rule is applied. We guess the Bucklin winning round k or if SP-AV is used and the number $|V_a^\gamma|$ of active voters; there are $O(|V| \cdot |C|)$ guesses. (SP-AV is used exactly when the winning score is less than $|V_a^\gamma|/2$.) If Bucklin is used, then we need a slight modification to take push actions into account. Instead of $\tau_c^{va} = \mathrm{bool}_{|C|}(x_c < k)$ we have

$$\tau_c^v = \mathrm{bool}_{|C|}(x_c^v < k) \wedge \mathrm{bool}_{|C|}(k \le x_\alpha^v);$$

and similarly for $\tilde{\tau}_c^{va}$. Observe that both conditions are valid.

For each of the rules (R1)–(R6), as argued, we have constructed an extended n-fold IP with $O(|C|^2)$ variables and locally uniform constraints per brick, and $O(|C|^2)$ globally uniform constraints. The largest coefficient is $|C|$, the height is $O(|C|^2)$, and the extended width is also $O(|C|^2)$. Thus, by Theorem 3.4 we can compute an n-fold matrix with parameters $r = s = t = \mathsf{a} = O(|C|^2)$. Proposition 3.1 is then used to solve this n-fold IP in time $2^{O(|C|^6 \log |C|)} n^3 \langle \mathbf{w} \rangle$. Also, $O(3^{|C|^2})$ guesses suffice for each rule except Maximin, Bucklin, and Fallback, where $O(|C|^2|V|)$ guesses suffice.

An exception to this runtime is the Kemeny rule:

(R7) Kemeny. For $c^\star$ to be a Kemeny winner, there has to be a ranking $\succ_{R^*}$ of the candidates that ranks $c^\star$ first and maximizes the total agreement with voters

$$\sum_{v\in V} \left|\{(c,c') \in C \times C \mid ((c \succ_{R^*} c') \Leftrightarrow (c \succ_v c'))\}\right|$$

among all rankings. In other words, the number of swaps sufficient to transform all $\succ_i$ into $\succ_{R^*}$ is smaller than the number of swaps needed to transform all $\succ_v$ into any other $\succ_{R'}$, where $c^\star$ is not first.

We guess the ranking $\succ_{R^*}$ over all rankings of C; then we introduce variables x_R^v for $R \in \{R^*\} \cup \{R' \mid c^\star \text{ is not first in } R'\}$ so that x_R^v is the number of swaps needed to transform $\succ_v^S$ into $\succ_R$ (recall that S is the bribery described by the variables x_c^v). We again split variables x_R^v and in the score comparison consider their active parts x_R^{av} only. Then, we introduce the necessary constraints:

$$x_R^v = \sum_{c,c' \in C, c \neq c'} \text{bool}_2\left(\text{sign}_{|C|}\left(x_c^v - x_{c'}^v\right) = \text{sign}_{|C|}(\text{rank}(c', R) - \text{rank}(c, R))\right) \qquad \text{for all } R,$$

$$\sum_{v \in V} x_R^{av} > \sum_{v \in V} x_{R^*}^{av}, \qquad R \neq R^*.$$

Those are valid, as we have $1 \leq x_c^v, x_{c'}^v, \text{rank}(c', R), \text{rank}(c, R) \leq |C|$ and because the result of the sign operation is guaranteed to be in $\{0, 1\}$. This solves Kemeny-Multi-Bribery in time $|C|^{O(|C|!^6)}n^3$, and completes the proof of Theorem 1.1.

5 LOWER BOUNDS AND HARDNESS FOR N-FOLD IPS

Here we provide the proof of Lemma 3.5, which shows that the bool() operation cannot be implemented in n-fold IPs without introducing large numbers into the system and that solving n-fold IPs becomes W[1]-hard when parameterized only by (r, s, t).

5.1 bool() Inexpressibility: Unary Bin Packing

The Unary Bin Packing problem takes as input n items of integer sizes $o_1, \ldots, o_n$ as well as two integers k, B, and asks if the items can be packed into k bins each of which has capacity B. Here by packing we mean an assignment of items to bins $\sigma : \{1, \ldots, n\} \to \{1, \ldots, k\}$ such that the packing is *admissible*, that is, $\sum_{i \in \sigma^{-1}(j)} o_i \leq B$ for every $j \in \{1, \ldots, k\}$.

Unary Bin Packing	Parameter: k
Input: Positive integers k, B encoded in unary and item sizes $o_1, \ldots, o_n$ for every $i \in \{1, \ldots, n\}$.	
Task: Find an admissible packing of the items to k bins.	

Jansen et al. [45] prove that this problem, parameterized by k, is W[1]-hard.

Lemma 5.1. *Unless* FPT = W[1], *the* bool() *operation cannot be expressed in n-fold IP format by introducing only $f(k)$ new variables and numbers bounded by $f(k)$, for any computable function f and $k = \max\{r, s, t\}$.*

Proof. Given an instance $(o_1, \ldots, o_n, k, B)$ of Unary Bin Packing, we create an n-fold IP as follows. We create a brick for each item o_i, and introduce k variables $x_1^i, \ldots, x_k^i$ for $i = 1, \ldots, n$ and the following locally uniform constraints:

$$\sum_{j=1}^{k} x_j^i = o_i, \qquad \text{(AssignItem)}$$

$$\sum_{j=1}^{k} \text{bool}_n(x_j^i) = 1. \qquad \text{(OneBin)}$$

Clearly, (AssignItem) and (OneBin) together force $x_j^i = o_i$ for exactly one j, which we then denote $j(i)$, and $x_{j'}^i = 0$ for all $j' \neq j(i)$. Intuitively, this means that an item i belongs to the bin $j(i)$.

Using globally uniform constraints we enforce that no bin overflows:

$$\sum_i x_j^i \leq B, \qquad j = 1, \ldots, k.$$

It is clear that this IP is feasible if and only if $(o_1, \ldots, o_n, k, B)$ is a "yes" instance.

Now suppose it is possible to express the bool_n using $f(k)$ additional local variables and $f(k)$ locally uniform constraints all of which use numbers bounded by $f(k)$ in absolute value. Replacing the condition (OneBin) with this expression and invoking Theorem 3.4 on the thus altered IP model we obtain an n-fold IP with $r, s, t, \mathsf{a} = f(k)$. Finally, this would yield an $f'(k)n^{O(1)}B^{O(1)}$-time algorithm for the Unary Bin Packing problem, where f' is a computable function independent of n. Consequently, FPT = W[1]. □

5.2 Largest Coefficient Matters: Subset Sum

The Subset Sum problem is a well-known (weakly) NP-hard problem and is defined as follows. Given n positive integers $w_1, \ldots, w_n$ and the target value T the task is to find a set $I \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in I} w_i = T$.

Lemma 5.2. *Unless* P = NP*, there is no algorithm solving n-fold IPs of encoding length L in time $f(r, s, t) \cdot (nL)^{O(1)}$ for any computable function f.*

Proof. We formulate the Subset Sum problem straightforwardly as an n-fold IP with n bricks and exactly one global condition:

$$\sum_{i=1}^{n} w_i x_1^i = T, \tag{10}$$

where $x_1^i \in \{0, 1\}$ are binary variables. Observe that the parameters of the above n-fold IP are $r = t = 1$ and $s = 0$.

Suppose now we have an algorithm that solves n-fold IP in the claimed time $f(r, s, t) \cdot (nL)^{O(1)}$. Since all of r, s, t are bounded by a constant, it follows that $f(r, s, t)$ is a constant. Let $c = f(1, 0, 1)$. Using such a hypothetical algorithm we can decide Subset Sum in time $c \cdot (nL)^{O(1)}$, where L is the length of binary encoding of the vector $(w_1, \ldots, w_n, T)$. Consequently, P = NP. □

Proof of Lemma 3.5. The lemma is a direct consequence of Lemma 5.1 and Lemma 5.2. □

6 CONCLUSIONS AND OPEN PROBLEMS

We introduced a general voting and bribing problem, $\mathcal{R}$-Multi-Bribery, which allows for swaps, push actions, and control changes. For several classical voting rules $\mathcal{R}$, we provided formulations of $\mathcal{R}$-Multi-Bribery in terms of (extended) n-fold integer programs; those formulations lead

- to the first fixed-parameter algorithms for some of those problems and
- to the first single-exponential algorithms for others.

Our approach is also natural in handling situations where each voter has different pricing functions, which was previously not possible in most cases. In particular, we provide the first fixed-parameter algorithm for $\mathcal{R}$-Swap-Bribery with arbitrary cost functions, for many natural voting rules $\mathcal{R}$.

While our result covers many classical and well-studied voting rules, we are convinced that many other rules are covered by our framework. What would be highly desirable though is to obtain sufficient conditions on easy-to-check properties of rules $\mathcal{R}$ for which $\mathcal{R}$-Multi-Bribery is fixed-parameter tractable parameterized by the number of candidates, or for which it admits a single-exponential time algorithm.

It would also be desirable to complement our algorithmic upper bounds with matching lower bounds based on the Exponential Time Hypothesis, by either improving the runtimes of the provided algorithms or providing appropriate hardness results.

Partial orders. For simplicity, we have focused our attention on handling elections in which votes are given as total orders. Some generalizations are clearly possible: for example, the Possible Winner problem, where voters are given as partial orders, reduces to Swap Bribery. If voters are given as weak orders (or bucket orders), the situation becomes somewhat less clear: It is obvious that there should be no cost of swapping candidates in the same group, but it is not clear how winner determination should work in such a setting; for example, it seems counter-intuitive if the Borda rule would give different numbers of points to candidates in a single group. This means that our outlined approach of defining swap costs to be zero within groups corresponds to a bribery action that first extends a weak order to a total order arbitrarily for free, and then performs "regular" swap bribery. We can imagine different models that would also make sense. Last, we have considered truncated orders and shown some results for them. Some work has been already done toward answering related questions, see References [38, 56, 57].

We believe it is interesting to study in more detail how Swap Bribery can be sensibly generalized to partial orders. However, we are confident that essentially any possible generalization will be captured by the Minimum Move framework [51] in which one defines voter types and arbitrary costs for moving a voter from one type to another. Specifically, one can define a type for each possible partial order over $|C|$ candidates, of which there is at most $2^{|C|^2}$ many, then define the cost of bribing a voter to move from one type to another, and then define how each type interacts with the voting rule.

APPENDIX

A PROBLEM DEFINITIONS

We provide the formal definitions of the voting and bribing problems covered by our result for $\mathcal{R}$-Multi-Bribery. In these definitions, for each problem $\mathcal{R}$-Problem, by "such that $c^\star$ is a winner in the election" we mean that $c^\star$ is a winner in the election under voting rule $\mathcal{R}$.

The $\mathcal{R}$-$Bribery problem was introduced by Faliszewski et al. [29], and further studied by Bredereck et al. [13]:

$\mathcal{R}$-$Bribery	Parameter: $\|C\|$
Input: A complete election (C, V), a designated candidate $c^\star \in C$, and a price vector $(p_v)_{v \in V}$.	
Task: Select a subset $S \subseteq V$ minimizing $\sum_{v \in S} p_v$ such that changing their preference orders $\succ_v$ for $v \in S$ arbitrarily to preference orders $\succ'_v$ makes $c^\star$ win the election $\left(C, (\succ'_v)_{v \in S} \cup (\succ_v)_{v \in V \setminus S}\right)$.	

The $\mathcal{R}$-Manipulation problem was introduced by Faliszewski et al. [32], who studied it for the Copeland$^\alpha$ rule:

$\mathcal{R}$-Manipulation	Parameter: $\|C\|$
Input: An election (C, V), a designated candidate $c^\star \in C$, a set $M \subseteq V$ of manipulators whose preference lists are empty (all candidates tied) and complete preferences for all $V \setminus M$.	
Task: Determine complete preference orders $\succ'_v$ for all voters $v \in M$ such that $c^\star$ is a winner in the election $(C, (\succ'_v)_{v \in M} \cup (\succ_v)_{v \in V \setminus M})$.	

The $\mathcal{R}$-Swap-Bribery problem was introduced by Elkind et al. [25] (therein referred to as $\mathcal{E}$-Swap-Bribery):

$\mathcal{R}$-Swap-Bribery Parameter: $|C|$

Input: A complete election (C, V), a designated candidate $c^\star \in C$, and swap cost functions σ^v for $v \in V$.

Task: Find a set S of admissible swaps with minimum cost such that $c^\star$ is a winner in the election $(C, V)^S$.

$\mathcal{R}$-Shift-Bribery Parameter: $|C|$

Input: An election (C, V), a designated candidate $c^\star \in C$, and swap cost functions σ^v for $v \in V$.

Task: Find a set S of admissible shifts of smallest cost such that $c^\star$ is a winner in the election $(C, V)^S$.

$\mathcal{R}$-Support-Bribery Parameter: $|C|$

Input: An election (C, V) with approval counts a^v for $v \in V$, a designated candidate $c^\star \in C$, and push action cost functions π^v for $v \in V$.

Task: Find a set P of push actions with minimum cost such that $c^\star$ is a winner in the election $(C, V)^P$.

$\mathcal{R}$-Mixed-Bribery Parameter: $|C|$

Input: An election (C, V) with approval counts a^v for $v \in V$, a designated candidate $c^\star \in C$, swap cost functions σ^v for $v \in V$ and push action cost functions π^v for $v \in V$.

Task: Find a set S of admissible swaps and a set P of push actions with minimum cost such that $c^\star$ is a winner in the election $((C, V)^S)^P$.

$\mathcal{R}$-Extension-Bribery Parameter: $|C|$

Input: An election (C, V) with approval counts a^v for $v \in V$ and a^v-top-truncated preference orders, and push action cost functions π_v for $v \in V$.

Task: Extend the approved part of each voter by $k_v \in \mathbb{N}$ previously disapproved candidates such that $c^\star$ becomes the winner, minimizing $\sum_{v \in V} c_v(k_v)$.

$\mathcal{R}$-Possible Winner Parameter: $|C|$

Input: An election (C, V) and a designated candidate $c^\star \in C$.

Task: Decide if the partial orders $\prec_v, v \in V$ can be extended to linear orders $\prec'_v$ such that $c^\star$ is a winner in the election $(C, \{\prec'_v\}_{v \in V})$.

$\mathcal{R}$-CCAV/CCDV Parameter: $|C|$

Input: An election $(C, V = V_a \uplus V_\ell)$, a designated candidate $c^\star \in C$, activation costs α^v for $v \in V_\ell$ and deactivation costs δ^v for $v \in V_a$.

Task: Find a set of voter (de)activations with minimum cost so that $c^\star$ is a winner in the perturbed election.

Dodgson Score	Parameter: $\|C\|$
Input: An election (C, V) and a designated candidate $c^\star \in C$.	
Task: Find a smallest set S of admissible swaps such that $c^\star$ becomes the Condorcet winner of the election $(C, V)^S$.	

Young Score	Parameter: $\|C\|$
Input: An election (C, V) and a designated candidate $c^\star \in C$.	
Task: Find a smallest set $S \subseteq V$ of voters that need to be removed such that $c^\star$ becomes the Condorcet winner of the election $(C, V \setminus S)$.	

ACKNOWLEDGMENTS

We are grateful to the anonymous reviewers of a preliminary version of our article, which appeared in the proceedings of STACS 2017 [50], for their helpful comments, which led to a considerably improved presentation of our results here. We also express our gratitude to Ildikó Schlotter and Piotr Faliszewski for many helpful remarks and discussion. Last, we thank the anonymous reviewers of TEAC whose work led to further improvements in presentation as well as to more coherent write-up.

REFERENCES

[1] Kateřina Altmanová, Dušan Knop, and Martin Koutecký. 2019. Evaluating and tuning n-fold integer programming. *ACM J. Exp. Algor.* 24, 1 (2019), 1–22.

[2] John J. Bartholdi III, Craig A. Tovey, and Michael A. Trick. 1989. Voting schemes for which it can be difficult to tell who won the election. *Soc. Choice Welf.* 6, 2 (1989), 157–165.

[3] Dorothea Baumeister, Gábor Erdélyi, Olivia Johanna Erdélyi, and Jörg Rothe. 2015. Complexity of manipulation and bribery in judgment aggregation for uniform premise-based quota rules. *Math. Social Sci.* 76 (2015), 19–30.

[4] Dorothea Baumeister, Piotr Faliszewski, Jérôme Lang, and Jörg Rothe. 2012. Campaigns for lazy voters: Truncated ballots. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS'12)*. 577–584.

[5] Dorothea Baumeister and Jörg Rothe. 2016. Preference aggregation by voting. In *Economics and Computation. An Introduction to Algorithmic Game Theory, Computational Social Choice, and Fair Division*, Jörg Rothe (Ed.). Springer, 197–326.

[6] Nadja Betzler, Susanne Hemmann, and Rolf Niedermeier. 2009. A multivariate complexity analysis of determining possible winners given incomplete votes. In *Proceedings of the International Joint Conferences on Artificial Intelligence (IJCAI'09)*. 53–58.

[7] Nadja Betzler, Rolf Niedermeier, and Gerhard Woeginger. 2011. Unweighted coalitional manipulation under the Borda rule is NP-hard. In *Proceedings of the International Joint Conferences on Artificial Intelligence (IJCAI'11)*. 55–60.

[8] Steven J. Brams and Peter C. Fishburn. 2002. Voting procedures. In *Handbook of Social Choice and Welfare*, Katora Suzumura Kenneth J. Arrow, Armatya K. Sen (Ed.). Handbooks in Economics, Vol. 19. Elsevier, 173–236.

[9] Robert Bredereck, Jiehua Chen, Piotr Faliszewski, Jiong Guo, Rolf Niedermeier, and Gerhard J. Woeginger. 2014. Parameterized algorithmics for computational social choice: Nine research challenges. *Tsinghua Sci. Tech.* 19, 4 (2014), 358–373.

[10] Robert Bredereck, Jiehua Chen, Piotr Faliszewski, André Nichterlein, and Rolf Niedermeier. 2014. Prices matter for the parameterized complexity of shift bribery. In *Proceedings of the Association for the Advancement of Artificial Intelligence (AAAI'14)*. 552–558.

[11] Robert Bredereck, Jiehua Chen, Sepp Hartung, Stefan Kratsch, Rolf Niedermeier, Ondřej Suchý, and Gerhard J. Woeginger. 2014. A multivariate complexity analysis of lobbying in multiple referenda. *J. Artif. Intell. Res.* 50 (2014), 409–446.

[12] Robert Bredereck, Piotr Faliszewski, Andrzej Kaczmarczyk, Rolf Niedermeier, Piotr Skowron, and Nimrod Talmon. 2017. Robustness among multiwinner voting rules. In *Proceedings of the International Symposium on Algorithmic Game Theory (SAGT'17)*, Lecture Notes Computer Science, Vol. 10504. 80–92.

[13] Robert Bredereck, Piotr Faliszewski, Rolf Niedermeier, Piotr Skowron, and Nimrod Talmon. 2015. Elections with few candidates: Prices, weights, and covering problems. In *Proceedings of the Conference on Algorithmic Decision Theory (ADT'15)*, Lecture Notes Computer Science, Vol. 9346. 414–431.

[14] Robert Bredereck, Piotr Faliszewski, Rolf Niedermeier, Piotr Skowron, and Nimrod Talmon. 2016. Complexity of shift bribery in committee elections. In *Proceedings of the Association for the Advancement of Artificial Intelligence (AAAI'16)*. 2452–2458.

[15] Robert Bredereck, Piotr Faliszewski, Rolf Niedermeier, and Nimrod Talmon. 2016. Large-scale election campaigns: Combinatorial shift bribery. *J. Artif. Intell. Res.* 55 (2016), 603–652.

[16] David Cary. 2011. Estimating the margin of victory for instant-runoff voting. In *Proceedings of the 2011 Electronic Voting Technology Workshop / Workshop on Trustworthy Elections*.

[17] Bruno Courcelle. 1990. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Inform. Comput.* 85, 1 (1990), 12–75.

[18] Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. 2015. *Parameterized Algorithms*. Springer.

[19] Jesús A. De Loera, Raymond Hemmecke, and Matthias Köppe. 2013. *Algebraic and Geometric Ideas in the Theory of Discrete Optimization*. MOS-SIAM Series on Optimization, Vol. 14. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA.

[20] Palash Dey, Neeldhara Misra, and Y. Narahari. 2015. Detecting possible manipulators in elections. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS'15)*. 1441–1450.

[21] Palash Dey, Neeldhara Misra, and Y. Narahari. 2017. Frugal bribery in voting. *Theoret. Comput. Sci.* 676 (2017), 15–32.

[22] Britta Dorn and Ildikó Schlotter. 2012. Multivariate complexity analysis of swap bribery. *Algorithmica* 64, 1 (2012), 126–151.

[23] Friedrich Eisenbrand, Christoph Hunkenschröder, Kim-Manuel Klein, Martin Koutecký, Asaf Levin, and Shmuel Onn. 2019. *An Algorithmic Theory of Integer Programming*. Technical Report. https://arxiv.org/abs/1904.01361.

[24] Edith Elkind and Piotr Faliszewski. 2010. Approximation algorithms for campaign management. In *Proceedings of the Conference on Web and Internet Economics (WINE'10)*, Lecture Notes Computer Science, Vol. 6484. 473–482.

[25] Edith Elkind, Piotr Faliszewski, and Arkadii Slinko. 2009. Swap bribery. In *Proceedings of the International Symposium on Algorithmic Game Theory (SAGT'09)*, Lecture Notes Computer Science, Vol. 5814. 299–310.

[26] Herbert B. Enderton. 2001. *A Mathematical Introduction to Logic* (2nd ed.). Harcourt/Academic Press, Burlington, MA.

[27] Piotr Faliszewski. 2008. Nonuniform bribery. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS'08)*. 1569–1572.

[28] Piotr Faliszewski, Rica Gonen, Martin Koutecký, and Nimrod Talmon. 2018. Opinion diffusion and campaigning on society graphs. In *Proceedings of the International Joint Conferences on Artificial Intelligence (IJCAI'18)*. 219–225.

[29] Piotr Faliszewski, Edith Hemaspaandra, and Lane A. Hemaspaandra. 2009. How hard is bribery in elections? *J. Artif. Intell. Res.* 40 (2009), 485–532.

[30] Piotr Faliszewski, Edith Hemaspaandra, and Lane A. Hemaspaandra. 2011. Multimode control attacks on elections. *J. Artif. Intell. Res.* 40, 1 (2011), 305–351.

[31] Piotr Faliszewski, Edith Hemaspaandra, Lane A. Hemaspaandra, and Jörg Rothe. 2009. Llull and Copeland voting computationally resist bribery and constructive control. *J. Artif. Intell. Res.* 35 (2009), 275–341.

[32] Piotr Faliszewski, Edith Hemaspaandra, and Henning Schnoor. 2008. Copeland voting: Ties matter. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS'08)*. 983–990.

[33] Piotr Faliszewski, Pasin Manurangsi, and Krzysztof Sornat. 2019. Approximation and hardness of shift-bribery. In *Proceedings of the Association for the Advancement of Artificial Intelligence (AAAI'19)*. 1901–1908.

[34] Piotr Faliszewski, Yannick Reisch, Jörg Rothe, and Lena Schend. 2015. Complexity of manipulation, bribery, and campaign management in Bucklin and fallback voting. *Auton. Agent Multi-Agent Syst.* 29, 6 (2015), 1091–1124.

[35] Piotr Faliszewski and Jörg Rothe. 2016. Control and bribery in voting. In *Handbook of Computational Social Choice*, Felix Brandt, Vincent Conitzer, Ulle Endriss, Jérôme Lang, and Ariel D. Procaccia (Eds.). Cambridge University Press, 146–168.

[36] Piotr Faliszewski, Piotr Skowron, and Nimrod Talmon. 2017. Bribery as a measure of candidate success: Complexity results for approval-based multiwinner rules. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS'17)*. 6–14.

[37] Peter C. Fishburn. 1977. Condorcet social choice functions. *SIAM J. Appl. Math.* 33, 3 (1977), 469–489.

[38] Zack Fitzsimmons and Edith Hemaspaandra. 2015. Complexity of manipulative actions when voting with ties. In *Proceedings of the Conference on Algorithmic Decision Theory (ADT'15)*, Lecture Notes Computer Science, Vol. 9346. 103–119.

[39] J. Flum and M. Grohe. 2006. *Parameterized Complexity Theory*. Springer-Verlag, Berlin.

[40] Eugene C. Freuder. 1990. Complexity of K-tree structured constraint satisfaction problems. In *Proceedings of the Association for the Advancement of Artificial Intelligence (AAAI'90)*. 4–9.
[41] Robert Ganian and Sebastian Ordyniak. 2016. The complexity landscape of decompositional parameters for ILP. In *Proceedings of the Association for the Advancement of Artificial Intelligence (AAAI'16)*. 710–716.
[42] Grzegorz Gawron and Piotr Faliszewski. 2019. Robustness of approval-based multiwinner voting rules. In *Proceedings of the Conference on Algorithmic Decision Theory (ADT'19)*, Lecture Notes Computer Science, Vol. 11834. 17–31.
[43] Noam Hazon, Raz Lin, and Sarit Kraus. 2013. How to change a group's collective decision? In *Proceedings of the International Joint Conferences on Artificial Intelligence (IJCAI'13)*. 198–205.
[44] Raymond Hemmecke, Shmuel Onn, and Lyubov Romanchuk. 2013. n-fold integer programming in cubic time. *Math. Program.* 137, 1-2, Ser. A (2013), 325–341.
[45] Klaus Jansen, Stefan Kratsch, Dániel Marx, and Ildikó Schlotter. 2013. Bin packing with fixed number of bins revisited. *J. Comput. System Sci.* 79, 1 (2013), 39–49.
[46] Andrzej Kaczmarczyk and Piotr Faliszewski. 2019. Algorithms for destructive shift bribery. *Auton. Agent Multi-Agent Syst.* 33, 3 (2019), 275–297.
[47] Ravi Kannan. 1983. Improved algorithms for integer programming and related lattice problems. In *Proceedings of the Annual ACM Symposium on Theory of Computing (STOC'83)*. 193–206.
[48] Orgad Keller, Avinatan Hassidim, and Noam Hazon. 2019. Approximating weighted and priced bribery in scoring rules. *J. Artif. Intell. Res.* 66 (2019), 1057–1098.
[49] Orgad Keller, Avinatan Hassidim, and Noam Hazon. 2019. New approximations for coalitional manipulation in scoring rules. *J. Artif. Intell. Res.* 64 (2019), 109–145.
[50] Dušan Knop, Martin Koutecký, and Matthias Mnich. 2017. Voting and bribing in single-exponential time. In *Proceedings of the Symposium on Theoretical Aspects of Computer Science (STACS'17)*, Leibniz Int. Proc. Informatics, Vol. 66. 46:1–46:14.
[51] Dušan Knop, Martin Koutecký, and Matthias Mnich. 2018. A unifying framework for manipulation problems. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS'18)*. 256–264.
[52] Dusan Knop, Martin Koutecký, and Matthias Mnich. 2019. Combinatorial n-fold integer programming and applications. *Mathematical Programmin*. https://doi.org/10.1007/s10107-019-01402-2.
[53] Hendrik W. Lenstra, Jr. 1983. Integer programming with a fixed number of variables. *Math. Operat. Res.* 8, 4 (1983), 538–548.
[54] Thomas R. Magrino, Ronald L. Rivest, and Emily Shen. 2011. Computing the margin of victory in IRV elections. In *Proceedings of the 2011 Electronic Voting Technology Workshop/Workshop on Trustworthy Elections*.
[55] Cynthia Maushagen, Marc Neveling, Jörg Rothe, and Ann-Kathrin Selker. 2018. Complexity of shift bribery in iterative elections. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS'18)*. 1567–1575.
[56] Vijay Menon and Kate Larson. 2017. Computational aspects of strategic behaviour in elections with top-truncated ballots. *Auton. Agents Multi-Agent Syst.* 31, 6 (2017), 1506–1547.
[57] Nina Narodytska and Toby Walsh. 2014. The computational impact of partial votes on strategic voting. In *Proceedings of the European Conference on Artificial Intelligence (ECAI'14)*, Frontiers Artificial Intelligence Appl., Vol. 263. 657–662.
[58] Shmuel Onn. 2010. *Nonlinear Discrete Optimization*. European Mathematical Society. 147 pages.
[59] Anja Rey, Jörg Rothe, and Adrian Marple. 2017. Path-disruption games: Bribery and a probabilistic model. *Theory Comput. Syst.* 60, 2 (2017), 222–252.
[60] Ildikó Schlotter, Piotr Faliszewski, and Edith Elkind. 2017. Campaign management under approval-driven voting rules. *Algorithmica* 77, 1 (2017), 84–115.
[61] Dmitry Shiryaev, Lan Yu, and Edith Elkind. 2013. On elections with robust winners. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS'13)*. 415–422.
[62] Lirong Xia. 2012. Computing the margin of victory for various voting rules. In *Proceedings of the ACM Conference on Economics and Computation (EC'12)*. 982–999.
[63] Hobart P. Young. 1977. Extending Condorcet's rule. *J. Econ. Theory* 16, 2 (1977), 335–353.
[64] William S. Zwicker. 2016. Introduction to the theory of voting. In *Handbook of Computational Social Choice*, Felix Brandt, Vincent Conitzer, Ulle Endriss, Jérôme Lang, and Ariel D. Procaccia (Eds.). Cambridge University Press, 23–56.

Received November 2018; revised March 2020; accepted March 2020