Voting and Bribing in Single-Exponential Time

Dušan Knop[*1], **Martin Koutecký**[†2], **and Matthias Mnich**[‡3]

1 **Dept. of Applied Mathematics, Charles University, Prague, Czech Republic**
knop@kam.mff.cuni.cz
2 **Dept. of Applied Mathematics, Charles University, Prague, Czech Republic**
koutecky@kam.mff.cuni.cz
3 **Maastricht University, Dept. of Quantitative Economics, Maastricht, The Netherlands; and**
Universität Bonn, Institut für Informatik, Bonn, Germany
mmnich@uni-bonn.de

Abstract

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 wins the manipulated 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 for a candidate.

As our main result, we show that $\mathcal{R}$-Multi-Bribery is fixed-parameter tractable parameterized by the number of candidates for many natural voting rules $\mathcal{R}$, including Kemeny rule, all scoring protocols, maximin rule, Bucklin rule, fallback rule, SP-AV, and any C1 rule. In particular, our result resolves the parameterized of $\mathcal{R}$-Swap Bribery for all those voting rules, thereby solving a long-standing open problem and "Challenge #2" of the 9 Challenges in computational social choice by Bredereck et al.

Further, our algorithm runs in single-exponential time for arbitrary cost; it thus improves the earlier double-exponential time algorithm by Dorn and Schlotter that is restricted to the unit-cost case for all scoring protocols, the maximin rule, and Bucklin rule.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2 Discrete Mathematics, J.4 Social and Behavioral Sciences

Keywords and phrases Parameterized algorithm, swap bribery, n-fold integer programming

Digital Object Identifier 10.4230/LIPIcs.STACS.2017.46

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 set C of candidates and a set V of voters v, each of which is equipped with a total order $\prec_v$ indicating their preferences over the 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 a majority of voters win(s) – or the Borda rule, where each candidate receives $|C| - i$ points from being

* D. K. was supported by project CE-ITI P202/12/G061 of GA ČR and project 1784214 of GA UK.
† M. K. was supported by project 14-10003S of GA ČR and projects 1784214 and 338216 of GA UK.
‡ M. M. was supported by ERC Starting Grant 306465 (BeyondWorstCase).

Table 1 $\mathcal{R}$-Multi-Bribery generalizes several studied bribery problems. For Kemeny rule, no previous results are known to us. Also, XP denotes an algorithm with run time $n^{f(|C|)}$ and FPT-AS a fixed-parameter approximation scheme.

Problem	Specialization of $\mathcal{R}$-Multi-Bribery	Previous best result (except Kemeny rule)
$\mathcal{R}$-$Bribery	$\pi_i = 0, \alpha_i = \infty, a_i = d_i = \infty$	$2^{2^{O(\|C\|)}}\ n^{O(1)}$ [9]
$\mathcal{R}$-Manipulation	$\iota_i = 0$ for $i \in S$, $\iota_i = \infty$ for $i \notin S$	$2^{2^{O(\|C\|)}} \cdot n^{O(1)}$ [9]
$\mathcal{R}$-CCAV/$\mathcal{R}$-CCDV	$\iota_i = 0, \pi_i = \alpha_i = \infty$	$2^{2^{O(\|C\|)}} \cdot n^{O(1)}$ [9]
$\mathcal{R}$-Swap Bribery	$\alpha_i = \infty$, $a_i = d_i = \infty$, $\iota_i = 0$	$2^{2^{O(\|C\|)}} \cdot n^{O(1)}$, unit cost [13]
$\mathcal{R}$-Shift Bribery	$\mathcal{R}$-Swap Bribery with $\pi_i(a,b) = \infty$ for $a, b \neq c_1$	XP, arbitrary cost, FPT-AS, restricted cost [10]
$\mathcal{R}$-Support Brib.	$\pi_i = a_i = d_i = \infty, \iota_i = 0$	NP-c [26]
$\mathcal{R}$-Mixed Brib.	$a_i = d_i = \infty$, $\iota_i = 0$	NP-c [14]
$\mathcal{R}$-Extension Brib.	$\pi_i(a,b) = 0$ if $\text{rank}_i(a,b) > l_i$, else $\pi_i(a,b) = \infty$, $a_i = d_i = \infty$, $\iota_i = 0$	NP-c [2]
$\mathcal{R}$-Possible Win.	reduce to $\mathcal{R}$-Swap Bribery [14, Thm 2]	$2^{2^{O(\|C\|)}} \cdot n^{O(1)}$ [4]
Dodgson Score	Condorcet-Swap Bribery with $\pi_i = 1$	$2^{2^{O(\|C\|)}} \cdot n^{O(1)}$ [1]
Young Score	$\mathcal{R}$-CCDV with $\mathcal{R}$ = Condorcet, $d_i = 1$	$2^{2^{O(\|C\|)}} \cdot n^{O(1)}$ [27]

ranked i-th by a voter and the candidate with most points wins; and the Copeland rule, which orders candidates by their number of pairwise victories minus their number of pairwise defeats. The goal is to *manipulate* the given election (C, V) by some actions Θ in such a way that a designated candidate $c_1 \in C$ wins the perturbed election $(C, V)^\Theta$ under the fixed voting rule $\mathcal{R}$. Such manipulation problems model various real-life issues, such as actual bribery, or campaign management, or post-election checks, as in destructive bribery (known as margin of victory). Manipulation is performed by the actions of *swapping* the position of two adjacent candidates in the preference order of some voter, by *support changes* that perturb the approval count of a voter, 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. [14] where each voter may assign different prices for swapping two consecutive candidates in their preference order; this captures the notion of small changes and comprises the preferences of the voters. We additionally allow voter-individual cost for support changes and control changes. We call this 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 which problems are captured by $\mathcal{R}$-Multi-Bribery.

For instance, Faliszewski et al. [18] introduced the $\mathcal{R}$-Swap Bribery problem, where only swaps are permitted. Of particular interest has been to understand the computational complexity such problems with respect to the number $|C|$ of candidates [1, 4, 7, 8, 9, 10, 11, 13, 19]. A common outcome are *fixed-parameter algorithms* that find an optimal solution of each instance I in time $f(|C|) \cdot |I|^{O(1)}$ for some function f; for example, Dorn and Schlotter [13] show how to solve $\mathcal{R}$-Swap Bribery with unit costs in time $2^{2^{O(|C|)}} \cdot |I|^{O(1)}$[1] for so-called linearly describable voting rules $\mathcal{R}$. In general, the function f grows quite fast, often double-exponential in $|C|$ which stems from solving a certain integer linear program (ILP) at some point of the algorithm. This observation led Bredereck et al. [6] to put forward

[1] Bredereck et al. [7] pointed out that the algorithm by Dorn and Schlotter only works for unit costs.

the following "Challenge #1", as part of their "Nine Research Challenges in 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?*

Another downside of the "ILP-based approach" is that it inherently treats voters not as individuals, but as groups which 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* $\mathcal{R}$*?*

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 $\mathcal{R}$. In particular, our algorithm works for voter-dependent cost functions, and it runs in time that is only single-exponential in $|C|$.

▶ **Theorem 1.** $\mathcal{R}$-Multi-Bribery *is fixed-parameter tractable parameterized by the number of candidates, and can be solved in time*

- $2^{O(|C|^6 \log |C|)} \cdot n^3$ *if* $\mathcal{R}$ *is a scoring protocol, any C1 rule, or SP-AV,*
- $2^{O(|C|^6 \log |C|)} \cdot n^4$ *if* $\mathcal{R}$ *is the maximin, Bucklin or fallback rule, and*
- $2^{O(|C|!^6)} \cdot n^3$ *if* $\mathcal{R}$ *is the Kemeny rule.*

We argued $\mathcal{R}$-Multi-Bribery generalizes many well-studied voting and bribing problems, parameterized by the number of candidates. A direct corollary of Theorem 1 is:

▶ **Corollary 2.** *Let* $\mathcal{R}$ *be a 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 *with arbitrary cost is fixed-parameter tractable parameterized by the number* $|C|$ *of candidates.*

This solves "Challenge #2" by Bredereck et al. [6]. In particular, for scoring protocols, maximin rule and Bucklin rule, Corollary 2 extends and improves an algorithm by Dorn and Schlotter [13] that is restricted to the unit-cost case of $\mathcal{R}$-Swap Bribery, and requires double-exponential run time $2^{2^{O(|C|)}} \cdot n^{O(1)}$.

Furthermore, we avoid using Lenstra's algorithm for solving ILPs with bounded number of variables, and thereby achieve the exponential improvements over previous run times for $\mathcal{R}$-Swap Bribery. This way, we substantially contribute towards resolving "Challenge #1" by Bredereck et al.

We remark that it is unclear (cf. [19, p. 338]) if the Kemeny rule can be *described by linear inequalities* as defined by Dorn and Schlotter [13]; 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.

Another corollary of Theorem 1 is the following:

▶ **Corollary 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 [13] for unit cost, the XP-algorithm and the fixed-parameter approximation scheme for arbitrary cost by Bredereck et al. [7].

Further, we have the following:

▶ **Corollary 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. [9] who solved these problems in time that is double-exponential in $|C|$.

Our approach. Our approach to prove Theorem 1 is to formulate the $\mathcal{R}$-Multi-Bribery problem in terms of an n-fold integer program (IP). Unlike fixed-dimension ILPs, which can be handled by Lenstra's algorithm [23], n-fold IPs allow variable dimension at the expense of a more rigid block structure of the constraint matrix. We manage to encode many voting rules $\mathcal{R}$ in a constraint matrix that has this required structure. While the dimension of the IP is not bounded in terms of the number of candidates, we bound the dimension of each block by a function of $|C|$. Then we solve the n-fold IP via the fixed-parameter algorithm of Hemmecke, Onn and Romanchuk [22], parameterized by the largest coefficient and the largest dimension of each block of the IP.

We complement our positive results by a complexity lower bound for solving n-fold IPs:

▶ **Theorem 5.** *Assuming ETH, there is no algorithm solving n-fold IPs in time $a^{o(\sqrt[3]{r \cdot s \cdot t})} \cdot n^{O(1)}$, where a is the largest absolute value in the constraint matrix and r, s, t bound the dimension of each block. Further, solving n-fold IPs is* W[1]*-hard parameterized by r, s, t.*

We defer the proof of Theorem 5 to the full version of this paper.

Related work. Bribery problems in voting systems are well-studied [7, 13, 14, 19]. Bredereck et al. [7] consider shift bribery, 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 [11] 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. In a different model [10], complexity of bribery of elections admitting for multiple winners, such as when committees are formed, has been studied. Also, different cost models have been considered: Faliszewski et al. [18] 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 [17] and Faliszewski et al. [20] allow for prices that depend on the amount of change the voter is asked for by the briber. For various other bribery models that have been investigated algorithmically, cf. Rothe [3, Chapter 4.3.5].

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 [23]. Courcelle's theorem [12] 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 [21] 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.

2 Voting and Bribing Problems

We give notions for the problems we deal with; for background, cf. Brams and Fishburn [5].

Elections. An election (C, V) consists of a set C candidates and a set $V = V_a \cup V_\ell$ of voters, where V_a are *active* voters and V_ℓ are *latent* voters. Only active voters participate in an election, but through a control action (defined later) latent voters can become active or active voters can become latent. Unless specified otherwise, we assume that $V = V_a$. Each voter i is a linear order $\succ_i$ over the set C which we call a *preference order*. For distinct candidates a and b, we write $a \succ_i b$ if voter i prefers a over b. We denote by $\mathrm{rank}(c, i)$ the position of candidate $c \in C$ in the order $\succ_i$.

Swaps. Let (C, V) be an election and let $\succ_i \in V$ be a voter. A *swap* $\gamma = (a, b)_i$ in preference order $\succ_i$ means to exchange the positions of a and b in $\succ_i$; denote the resulting order by $\succ_i^\gamma$; the *cost* of $(a, b)_i$ is $\pi_i(a, b)$. A swap $\gamma = (a, b)_i$ is *admissible in* $\succ_i$ if $\mathrm{rank}(a, i) = \mathrm{rank}(b, i) - 1$. A set Γ of swaps is *admissible in* $\succ_i$ if they can be applied sequentially in $\succ_i$, one after the other, in some order, such that each one of them is admissible. Note that the obtained vote, denoted by $\succ_i^\Gamma$, is independent from the order in which the swaps of Γ are applied. We also extend this notation for applying swaps in several votes and denote it V^Γ.

Support changes. In voting rules such as SP-AV or Fallback, each voter $\succ_i$ also has an *approval count* $l_i \in \{0, \ldots, |C|\}$. For voter $\succ_i$ and $t \in \{-l_i, \ldots, |C| - l_i\}$, a *support change* t_i changes the approval count of voter i to $l_i + t$. We denote a set of support changes by Σ, and the changed set of voters by V^Σ. The cost of support change t_i is $\alpha_i(t)$; always $\alpha_i(0) = 0$. If voter i is involved in a swap or support change, a one-time *influence cost* ι_i occurs.

Control changes. The set of voters may be changed by activating some latent voters from V_ℓ or deactivating some active voters from V_a; we call this a *control change*. We denote the changed set of voters by $\overline{V}_\ell \cup \overline{V}_a$. The cost of activating voter $\succ_i \in V_\ell$ is a_i and the cost of deactivating voter $\succ_i \in V_a$ is d_i; always $a_i = 0$ for $\succ_i \in V_a$ and $d_i = 0$ for $\succ_i \in V_\ell$.

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 $\mathbf{s} = (s_1, \ldots, s_{|C|}) \in \mathbb{N}_0^{|C|}$ with $s_1 \geq \cdots \geq s_{|C|} \geq 0$. A candidate receives s_j points for each voter that ranks it as j-th best. The candidate with the maximum number of points is the winner. Examples include the Plurality rule ($\mathbf{s} = (1, 0, \ldots, 0)$), d-Approval ($\mathbf{s} = (1, \ldots, 1, 0, \ldots, 0)$ with d ones), and the Borda rule ($\mathbf{s} = (|C| - 1, |C| - 2, \ldots, 1, 0)$). Throughout, we assume that $\max_{j=1}^{|C|} s_j \leq |C|$, which is the case for the aforementioned popular rules.

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

Condorcet consistent rules. A candidate $c \in C$ is a *Condorcet winner* if any other $c' \in C \setminus \{c\}$ satisfies $|\{\succ_i \in V \mid c \succ_i c'\}| > |\{\succ_i \in V \mid c' \succ_i c\}|$. A voting rule is *Condorcet consistent* if it selects the Condorcet winner in case there is one. Such rules are classified by Brams and Fishburn [5] as C1, C2 or C3, depending on the kind of information needed to determine the winner. For $a, b \in C$ let $v(a, b) = |\{\succ_i \in V \mid a \succ_i b\}|$; we write $a <_M b$ if a beats b in a head-to-head contest, that is, if $v(a, b) > v(b, a)$.

Let $\mathcal{R}$ be a Condorcet consistent rule. We say $\mathcal{R}$ *is C1* if knowing $<_M$ suffices to determine the winner. C1 rules include the Copeland$^\alpha$ rule, the Slater rule, and others. E.g., the Copeland$^\alpha$ rule for $\alpha \in [0,1]$ specifies that for each head-to-head contest between two distinct candidates, if some candidate is preferred by a majority of voters then (s)he obtains one point and the other candidate obtains zero points, and if a tie occurs then both candidates obtain α points; the candidate with largest sum of points wins.

We say $\mathcal{R}$ *is C2* if it is not C1 and knowing $v(a,b)$ for all $a,b \in C$ is sufficient for determining the winner. The following two rules are C2:

Maximin. Declares $c \in C$ is a winner if it maximizes $v_*(c) = \min\{v(c,a) \mid c \neq a \in C\}$.

Kemeny. Declares $c \in C$ a winner if there exists a ranking of candidates $\succ_R$ such that c is first in $\succ_R$ and $\succ_R$ maximizes the total agreement with voters $\sum_{i=1}^{n} |\{(a,b) \mid ((a \succ_R b) \Leftrightarrow (a \succ_i b)) \ \forall a,b \in C\}|$ among all rankings.

We say $\mathcal{R}$ *is C3* if it is neither C2 nor C3. The following two rules are C3:

Dodgson. The *Dodgson score* of a candidate c is the minimum number of swaps needed such that c becomes the Condorcet winner. A candidate c is the *Dodgson winner* if their Dodgson score is minimum.

Young. Analogously, the *Young score* of a candidate c is the minimum number of voters that need to be deleted from an election for c to become the Condorcet winner. The candidate with the lowest Young score is the *Young winner*.

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

SP-AV. A candidate c gets a point from every voter i with $\text{rank}(c,i) \leq l_i$. The candidate with maximum number of points wins.

Fallback. Delete, for each voter $\succ_i$, the non-approved candidates (i.e., all c with $\text{rank}(c,i) > l_i$) from its order. Then, use the Bucklin rule, which might fail due to step one; in that case, use the SP-AV rule.

3 A grammar for n-fold integer programming

We now set up our main tool, a grammar for n-fold IPs. For background on n-fold IPs, we refer to the books of Onn [25] and De Loera et al. [24].

n-fold integer programs. Given nt-dimensional integer vectors $\mathbf{b}, \mathbf{u}, \mathbf{l}, \mathbf{w}$, an n-fold integer programming problem $(IP)_{E^{(n)},\mathbf{b},\mathbf{l},\mathbf{u},\mathbf{w}}$ in variable dimension nt is defined as

$$\min\left\{\mathbf{wx} : E^{(n)}\mathbf{x} = \mathbf{b}\,,\ \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}\,,\ \mathbf{x} \in \mathbb{Z}^{nt}\right\}, \quad \text{where} \quad E^{(n)} := \begin{pmatrix} D & D & \cdots & D \\ A & 0 & \cdots & 0 \\ 0 & A & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A \end{pmatrix}$$

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.

Hemmecke et al. [22] developed a dynamic program to show:

▶ **Proposition 6** ([22, Thm. 6.1]). *There is an algorithm solving $(IP)_{E^{(n)},\mathbf{b},\mathbf{l},\mathbf{u},\mathbf{w}}$ in time $a^{O(trs+t^2s)} \cdot O(n^3 L)$, where $a = \max\{\|D\|_\infty, \|A\|_\infty\}$ and L is the length of the input.*

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 j-th variable of the i-th brick with $j \in \{1,\ldots,t\}$ and $i \in \{1,\ldots,n\}$.

Also note that there are two types of constraints, given by either the row of matrices D, or by the matrix A for each brick. These types lead us to define globally and locally uniform expressions. A *globally uniform expression* for a t-tuple of coefficients $(a_1, \ldots, a_t)$ is:

$$\sum_{i=1}^{n} \sum_{j=1}^{t} a_j x_j^i \ .$$

Observe that constraints given by a row of matrices D have the form of a globally uniform expression equalling a number on the right hand side. We call them *globally uniform constraints*; typically they assure that the solution fits a budget or other global condition.

A *locally uniform expression* for $(a_1, \ldots, a_t)$ is an expression

$$\sum_{j=1}^{t} a_j x_j^i - b_i, \qquad i = 1, \ldots, n \ .$$

Notice that a locally uniform expression is in fact a set of n expressions which only differ in their additive constants. All constraints which are not globally uniform have the form of a locally uniform expression equaling some right hand side. We call them *locally uniform constraints* and typically use them to give variables within a brick their intended meaning.

As we have just seen, unlike with general IPs, n-fold IPs obey a uniform block structure. Many "integer programming tricks" are known for expressing logical connectives and other operations within IP; however, it is not obvious if they can be implemented also in n-fold IP. Our goal is to establish that all constraints constructed in a particular way are valid uniform constraints, and determine the parameters r, s, t and a in the run time of Theorem 6.

Expressions and constraints. We define an *n-fold IP grammar* as follows, where `lue` and `gue` stands for "locally uniform expression" and "globally uniform expression":

$\heartsuit ::= = | < | > | \leq | \geq$

$\diamondsuit ::= \heartsuit \ | \neq$

$\texttt{lue}_m ::= x^i \ |$ a variable per brick with l^i, u^i s.t. $\max_i u^i - \min_i l^i \leq m$

$\sum_{j=1}^{k} a_j \, \texttt{lue}_{m_j} \ |$ for k integers $a_1, \ldots, a_k$

$\texttt{lue}_m + o^i \ |$ for n integers $o^1, \ldots, o^n$

$(\texttt{lue}_m) \ |$ to clarify operator priority

λ an empty expression

$\texttt{gue} ::= \texttt{lue}_m$

The subscript $\texttt{lue}_m$ denotes an external guarantee that the result of this expression lies in $\{L, \ldots, U\}$ with $|U - L| \leq m$. Notice that in the above, one `lue` actually describes n objects, one in each brick. These objects differ from each other in exactly two ways. One, when a variable x^i is added to each brick, its lower and upper bounds l^i, u^i may be different for each $i \in \{1, \ldots, n\}$. Second, in the expression $\texttt{lue}_m + o^i$, the additive constant o^i may be different for each $i \in \{1, \ldots, n\}$.

Special attention is given to binary $\texttt{lue}$: $\texttt{lue}_b$ is $\texttt{lue}_1$ which is always in $\{0,1\}$.

$$\begin{aligned}
\texttt{lue}_b ::= {} & \texttt{lue}_b \to \texttt{lue}_b \mid \neg\, \texttt{lue}_b \mid \texttt{lue}_b \wedge \texttt{lue}_b \mid \\
& \texttt{lue}_b \vee \texttt{lue}_b \mid \texttt{lue}_b \oplus \texttt{lue}_b \mid \\
& \mathrm{bool}_m(\texttt{lue}_m) \mid && 0 \text{ if } \texttt{lue}_m \text{ is 0, else 1} \\
& \mathrm{bool}_m(\texttt{lue}_m \diamondsuit \texttt{lue}_m) && 1 \text{ if } \texttt{lue}_m \diamondsuit \texttt{lue}_m \text{, else 0} \\
\texttt{lue}_2 ::= {} & \mathrm{sign}_m(\texttt{lue}_m) && -1 \text{ if } \texttt{lue}_m \text{ neg., 1 if pos., 0 otherwise.}
\end{aligned}$$

Then, locally and globally uniform constraints ($\texttt{luc}$ and $\texttt{guc}$) are defined as:

$$\texttt{luc} ::= \texttt{lue}_m \diamondsuit \texttt{lue}_{m'} \qquad \text{and} \qquad \texttt{guc} ::= \texttt{gue} \heartsuit \texttt{gue}$$

Finally, an *n-fold IP recipe* is a tuple $(\mathcal{G}, \mathcal{L})$ where $\mathcal{G}$ and $\mathcal{L}$ are the sets of locally and globally uniform constraints, respectively. Let $t' \in \mathbb{N}$ be the number of variables introduced to each brick. Let $Sol(\mathcal{G}, \mathcal{L}) \subseteq \mathbb{Z}^{nt'}$ be the set of all vectors that satisfy all constraints in $(\mathcal{G} \cup \mathcal{L})$.

Constructing n-fold IP from recipe. Now we shall describe how to turn an n-fold IP recipe into a system of lower and upper bounds and linear equalities satisfying the n-fold format, proving this theorem:

▶ **Theorem 7.** *Let $(\mathcal{G}, \mathcal{L})$ be an n-fold IP recipe with $\mathcal{G} \cup \mathcal{L}$ build by m applications of the grammar rules such that a is an upper bound on $|a_j|$ in all coefficients and all m' appearing in $\mathrm{bool}_{m'}$ and $\mathrm{sign}_{m'}$. Then in time $O(|\mathcal{G} \cup \mathcal{L}|)$, one can compute numbers $r, s, t \in O(m)$, an n-fold matrix $E^{(n)} \in \mathbb{Z}^{nt}$, a right hand side $\mathbf{b} \in \mathbb{Z}^{r+sn}$ and lower and upper bounds $\mathbf{l}, \mathbf{u} \in \mathbb{Z}^{r+sn}$ such that $O(a)$ is an upper bound on the absolute value of the coefficients of $E^{(n)}$, and $Sol(\mathcal{G}, \mathcal{L}) = \{(\mathbf{x}^1|_{t'}, \ldots, \mathbf{x}_n|_{t'} \mid E^{(n)}(\mathbf{x}^1, \ldots, \mathbf{x}^n) = \mathbf{b}, \mathbf{l} \le \mathbf{x} \le \mathbf{u}\}$, where $\mathbf{x}^i|_{t'}$ is the restriction of $\mathbf{x}^i$ to its first t' variables.*

For the proof, to determine the parameters r, s, t of the resulting n-fold IP, we define the *s-increase of* $\texttt{lue}_m$ and *t-increase of* $\texttt{lue}_m$, denoted $\Delta s(\texttt{lue}_m)$ and $\Delta t(\texttt{lue}_m)$, and analogously for $\texttt{luc}$'s and $\texttt{gue}$'s. The s-increase $\Delta s(\texttt{lue}_m)$ and the t-increase $\Delta t(\texttt{lue}_m)$ are the number of auxiliary equalities and auxiliary variables needed to express a $\texttt{lue}_m$ or $\texttt{luc}$. We note the s- and t-increase of each rule after defining it.

Rewriting $\texttt{lue}_m$ to $\sum_{j=1}^{k} a_j x_j + \bar{b}^i$ for $i = 1, \ldots, n$. Rewriting $\texttt{lue}_m$ in some expression means replacing it with a new variable z and adding locally uniform constraints to $\mathcal{L}$. These constraints assure that the variable z will carry the desired meaning. The result is that every $\texttt{lue}_m$ is rewritten to the format $\sum_{j=1}^{t} a_j x_j + b^i$ as introduced in Sect. 3. In this phase we may still be adding constraints that are not in the n-fold format (contain inequalities etc.) as they will be dealt with later. Also note that we use the notation $\overline{\texttt{lue}}$ simply to distinguish between various $\texttt{lue}$; do not confuse this with negation.

- $\texttt{lue}_b ::= \texttt{lue}'_b \vee \overline{\texttt{lue}_b} \quad \Rightarrow \quad$ create a new binary variable s and set $2z = \texttt{lue}'_b + \overline{\texttt{lue}_b} + s$.
  $\Delta s(\texttt{lue}_b) = \Delta s(\texttt{lue}'_b) + \Delta s(\overline{\texttt{lue}_b}) + 1$, $\Delta t(\texttt{lue}_b) = \Delta t(\texttt{lue}'_b) + \Delta t(\overline{\texttt{lue}_b}) + 2$.
- $\texttt{lue}_b ::= \neg\, \texttt{lue}'_b \quad \Rightarrow \quad z = 1 - \texttt{lue}'_b$.
  $\Delta s(\texttt{lue}_b) = \Delta s(\texttt{lue}'_b) + 1$, $\Delta t(\texttt{lue}_b) = \Delta t(\texttt{lue}'_b)$.
- $\texttt{lue}_b ::= \texttt{lue}_b \to \texttt{lue}_b \mid \texttt{lue}_b \wedge \texttt{lue}_b \mid \texttt{lue}_b \oplus \texttt{lue}_b$: it is folklore [16] that all logical connectives can be constructed with $\neg$ and $\vee$.
  $\Delta s(\texttt{lue}_b) = \Delta s(\texttt{lue}'_b) + \Delta s(\overline{\texttt{lue}_b}) + O(1)$, $\Delta t(\texttt{lue}_b) = \Delta t(\texttt{lue}'_b) + \Delta t(\overline{\texttt{lue}_b}) + O(1)$

- $\texttt{lue}_b ::= \text{bool}_m(\texttt{lue}_m)$ and $\texttt{lue}_2 ::= \text{sign}_m(\texttt{lue}_m)$: We assume that $-L < \texttt{lue}_m < U$ for $L, U \in \mathbb{N}$ and that $m = \max\{L, U\}$[2]. Because we will introduce a coefficient upper bounded by m into the system, we will denote the operation $\text{bool}_m(x)$. Let v, u be two new variables such that $v = 1$ if and only if $\texttt{lue}_m \geq 0$, and $u = 1$ if and only if $\texttt{lue}_m \leq 0$:

$$v, u \in \{0, 1\}: \qquad 1 + \texttt{lue}_m \leq Uv \leq U + \texttt{lue}_m \quad \& \quad 1 - \texttt{lue}_m \leq Lu \leq L - \texttt{lue}_m$$

  Then if $\texttt{lue}_b ::= \text{bool}_m(\texttt{lue}_m)$ let $z = \neg(v \wedge u)$ and if $\texttt{lue}_b ::= \text{sign}_m(\texttt{lue}_m)$ let $z = v - u$. $\Delta s(\texttt{lue}_b) = \Delta(\texttt{lue}_m) + O(1)$, $\Delta t(\texttt{lue}_b) = \Delta t(\texttt{lue}_m) + O(1)$ and analogously for $\texttt{lue}_2$.
- $\texttt{lue}_b ::= \text{bool}_m(\texttt{lue}'_m \diamondsuit \overline{\texttt{lue}_m}) \quad \Rightarrow$
  - $\diamondsuit$ is "=": $z = \text{bool}_m(\texttt{lue}'_m - \overline{\texttt{lue}_m})$
  - $\diamondsuit$ is "$\neq$": $z = 1 - \text{bool}_m(\texttt{lue}'_m - \overline{\texttt{lue}_m})$
  - $\diamondsuit$ is "$>$": $z = \text{bool}_m(\text{sign}_m(\texttt{lue}'_m - \overline{\texttt{lue}_m}) = 1)$
  - $\diamondsuit$ is "$\geq$": $z = \text{bool}_m(\texttt{lue}'_m > \overline{\texttt{lue}_m}) \vee \text{bool}_m(\texttt{lue}'_m = \overline{\texttt{lue}_m})$

  And analogously when $\diamondsuit$ is "$<$" and "$\leq$".
  $\Delta s(\texttt{lue}_b) = \Delta s(\texttt{lue}'_b) + \Delta s(\overline{\texttt{lue}_b}) + O(1)$, $\Delta t(\texttt{lue}_b) = \Delta t(\texttt{lue}'_b) + \Delta t(\overline{\texttt{lue}_b}) + O(1)$
- $\texttt{lue}_m ::= \lambda \mid (\texttt{lue}_m) \mid \texttt{lue}_m + o^i \mid \sum_{j=1}^{k} a_j \texttt{lue}_{m_j} \mid x^i$ are rewritten in the obvious way.

Rewriting $\texttt{luc} ::= \texttt{lue}_m \diamondsuit \texttt{lue}_{m'}$ to $\texttt{lue}_{m''} = b^i$

- when $\diamondsuit$ is "=": $\texttt{lue}_m = \texttt{lue}_{m'} \quad \Rightarrow \quad \texttt{lue}_m - \texttt{lue}_{m'} = 0$.
  $\Delta s(\texttt{luc}) = \Delta s(\texttt{lue}_m) + \Delta s(\texttt{lue}_{m'})$ and $\Delta t(\texttt{luc}) = \Delta t(\texttt{lue}_m) + \Delta t(\texttt{lue}_{m'})$.
- when $\diamondsuit$ is "$\neq$": $\texttt{lue}_m \neq \texttt{lue}_{m'} \quad \Rightarrow \quad \text{bool}_{m+m'}(\texttt{lue}_m \neq \texttt{lue}_{m'}) = 1$. Then,
  $\Delta s(\texttt{luc}) = \Delta s(\text{bool}_{m+m'}(\texttt{lue}_m \neq \texttt{lue}_{m'}))$ and $\Delta t(\texttt{luc}) = \Delta t(\text{bool}_{m+m'}(\texttt{lue}_m \neq \texttt{lue}_{m'}))$.
- when $\diamondsuit$ is not "=", intuitively we want to add a slack variable:
  $\texttt{lue}_m \diamondsuit \texttt{lue}_{m'} \quad \Rightarrow \quad \texttt{lue}_m - \texttt{lue}_{m'} + \sum_{i=1}^{n} y^i = 0$ with y^i for $i = 1, \ldots, n$ being n new variables with $l^i = u^i = 0$ for $i > 1$ and with
  - $l^1 = 0$ and $u^1 = \infty$ when $\diamondsuit$ is "$\leq$",
  - $l^1 = 1$ and $u^1 = \infty$ when $\diamondsuit$ is "$<$",
  - $l^1 = -\infty$ and $u^1 = 0$ when $\diamondsuit$ is "$\geq$", and
  - $l^1 = -\infty$ and $u^1 = -1$ when $\diamondsuit$ is "$>$",

  "∞" stands for a sufficiently large number, which is usually clear from the context. Then,
  $\Delta s(\texttt{luc}) = \Delta s(\texttt{lue}_m) + \Delta s(\texttt{lue}_{m'})$ and $\Delta t(\texttt{luc}) = \Delta t(\texttt{lue}_m) + \Delta t(\texttt{lue}_{m'}) + 1$.

Finally, rewrite the globally uniform constraints in $\mathcal{G}$. Since $\texttt{gue} ::= \texttt{lue}_m$ and $\texttt{lue}_m$ gets rewritten to $\sum_{j=1}^{t} a_j x_j^i$ as required, the above rules suffice to obtain a globally uniform expression as defined in Sect. 3, and similarly for $\texttt{guc} ::= \texttt{gue} \heartsuit \texttt{gue}'$. Let $\Delta t(\texttt{guc}) = \Delta t(\texttt{gue}) + \Delta t(\texttt{gue}')$ if $\heartsuit$ is "=" and $\Delta t(\texttt{guc}) = \Delta t(\texttt{gue}) + \Delta t(\texttt{gue}') + 1$ otherwise and let $\Delta t(\texttt{gue}) = \Delta t(\texttt{lue}_m)$.

It remains to compute the parameters r, s, t. Let #variables be the number of distinct variables introduced through $\texttt{lue}_m ::= x^i$. Then,

- $t = \#\text{variables} + \sum_{\texttt{luc}_j \in \mathcal{L}} \Delta t(\texttt{luc}_j) + \sum_{\texttt{guc}_j \in \mathcal{G}} \Delta t(\texttt{guc}_j)$,
- $s = |\mathcal{L}| + \sum_{\texttt{luc}_j \in \mathcal{L}} \Delta s(\texttt{luc}_j)$,
- $r = |\mathcal{G}|$, and
- all coefficients are bounded in absolute value by $\max\{\max\{|a_j| \mid a_j \text{ in } \sum_{j=1}^{k} a_j \texttt{lue}_m\}, \max\{m \mid \text{bool}_m, \text{sign}_m \in \texttt{luc} \in \mathcal{L}\}\} \in O(a)$.

Clearly, $r, s, t \in O(m)$ and this concludes the proof of Theorem 7.

[2] This is without loss of generality: when $L < \texttt{lue}_m < U$ with $L, U \in \mathbb{N}$, $\texttt{lue}_m$ is always positive and $\text{bool}_m(\texttt{lue}_m)$ is always 1; analogously when $\texttt{lue}_m$ is always negative.

A demonstration of the rewriting process. We want m variables $x_1, \dots, x_m$ in each brick describing a permutation of $\{1, \dots, m\}$. That is equivalent to $\sum_{j=1}^{m} x_j = \binom{m+1}{2}$ and $x_j \neq x_k$ for all $j \neq k$ and $x_j \in \{1, \dots, m\}$ for all j. This is expressible by $1 + \binom{m}{2}$ `luc`'s:

$$\sum_{j=1}^{m} x_j = \binom{m+1}{2} \bigwedge x_j \neq x_k \quad \text{for all } j \neq k \ . \tag{1}$$

The first `luc` is already in the format required in Sect. 3. However, for the other `luc`'s, rewriting rules are applied. Fix j, k. The resulting n-fold IP will contain these constraints (for brevity we omit rewriting inequalities by slack variables as this is standard):

$$\begin{aligned}
w &= x_j - x_k \\
1 + w &\leq mv \leq m + w \\
1 - w &\leq mu \leq m - w \qquad \text{express } z_{\neg\,\mathrm{bool}} = \neg(v \wedge u) = \neg v \vee \neg u \\
v_\neg &= 1 - v \bigwedge u_\neg = 1 - u \qquad v, u, s \in \{0, 1\} \\
2z_\vee &= v_\neg + u_\neg + s \\
z_{\neg\,\mathrm{bool}} &= 1 - z_\vee \qquad \text{and set } \neg\,\mathrm{bool}(x_j - x_k) = 0 \\
z_{\neg\,\mathrm{bool}} &= 0 \ .
\end{aligned}$$

Further, given n permutations $o_1^i, \dots, o_m^i$ for $i = 1, \dots, n$, one for each brick, we want to compare the permutation $x_1, \dots x_m$ to $o_1, \dots, o_m$ and determine which indices are inverted, that is, $x_j < x_k \Leftrightarrow o_j > o_k$. In other words, we want to determine when the sign of $(x_j - x_k)$ equals the sign of $(o_k - o_j)$. So, for each $j \neq k$ we add a new indicator variable s_{jk} as follows:

$$s_{jk} = \mathrm{bool}_2(\mathrm{sign}_m(x_j - x_k) = \mathrm{sign}_m(o_k^i - o_j^i)) \tag{2}$$

Notice that $\mathrm{sign}_m(o_k^i - o_j^i)$ is a constant o_{kj}^i so the expression above turns to $\mathrm{bool}_2(\mathrm{sign}_m(x_j - x_k) - o_{kj}^i)$ which is now clearly a $\mathtt{lue}_b$ and is rewritten similarly as before.

An example of what is *not* expressible in the n-fold IP grammar is the (nonsensical) expression $\sum_{j=1}^{m} o_j^i x_j$. This is not a `lue` since the numbers o_j^i appear not as additive constants but as coefficients. However, coefficients are required to be identical across bricks in the grammar rule $\mathtt{lue}_m ::= \sum_{j=1}^{k} a_j\, \mathtt{lue}_{m_j}$.

▶ 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 run time in Theorem 6. One can show that such dependence is necessary (proof deferred):

▶ **Lemma 8.** *Unless* FPT = W[1]*, for any computable function f it is impossible to express the* bool() *operation consistently with the n-fold IP format while introducing only numbers bounded by $f(k)$, and introducing only $f(k)$ new variables. Moreover, solving n-fold IP parameterized only by the dimensions r, s, t (and not by the largest entry a) is* W[1]*-hard.*

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 which is common to all such rules. Thereafter, we add the parts of the formulation which depend on $\mathcal{R}$. 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, support changes, control changes) 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.

Swaps. We describe the preference order with swaps Γ applied by variables $x_1, \ldots, x_{|C|}$ with the intended meaning $x^i_j = \mathrm{rank}(c_j, i)^\Gamma$. Recall from Sect. 3 that constraints (1) enforce that $(x_1, \ldots, x_{|C|})$ is a permutation of $\{1, \ldots, |C|\}$; we add them to the program.

To express the swaps performed by Γ, for each pair of candidates $c_j, c_k \in C$ we introduce binary variables s_{jk}, s_{kj} so that $s_{jk} = 1$ if and only if c_j and c_k are swapped. We use the fact (cf. [15, Proposition 3.2]) that for two orders $\succ, \succ'$, the admissible set of swaps Γ such that $\succ' = \succ^\Gamma$ is uniquely given as the set of pairs (c_i, c_j) for which either $c_i \succ c_j \wedge c_j \succ' c_i$ or $c_j \succ c_i \wedge c_i \succ' c_j$. Thus, we only need to set constraint (2) from Sec. 3 with $o^i_j = \mathrm{rank}(c_j, i)$.

Support changes. To indicate support changes, we introduce binary variables $r_{-|C|}, \ldots, r_{|C|}$ where $r_0 = 1$ means no change, $r_j = 1$ means support change j. We set the lower and upper bounds to ensure that $r_j = 0$ for all $j \notin \{-l_i, |C| - l_i\}$. Finally, we introduce a variable $x_\alpha \in \{1, \ldots, |C|\}$ indicating the approval count after the support change:

$$\sum_{j=-|C|}^{|C|} r_j = 1, \qquad x_\alpha = l_i + \sum_{j=-|C|}^{|C|} j r_j \ .$$

Influence bit. We introduce a binary variable x_ι taking value 1 if a swap or a support change is performed, and value 0 otherwise: $x_\iota = \mathrm{bool}_{|C|^2+2|C|}\left(\sum_{j\neq k} s_{jk} + \sum_{j\neq 0} r_j\right)$.

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

Further, for each pair $c_j, c_k \in C$ of candidates we introduce a variable x_{jk} which takes value 1 if $c_j \succ^\gamma_i c_k$ and 0 otherwise. We will also frequently (and implicitly) use the following variable-splitting trick:

▶ **Lemma 9.** *Let x be an integral variable with lower bound ℓ and upper bound u and let z be a binary variable. It is possible to introduce a variable x^z and auxiliary* `lue` *constraints with $\Delta s = O(1)$ and $\Delta t = O(1)$ such that $x^z = x$ if $z = 1$ and $x^z = 0$ if $z = 0$.*

Proof. First, add $\ell z \leq x^z \leq uz$; then $z = 0$ implies $x^z = 0$ and does not influence x^z when $z = 1$. Second, add $\mathrm{bool}_b(\mathrm{bool}_b(z = 1) \to \mathrm{bool}_m(x^z = x) = 1)$ with $m = O(-\ell + u)$. ◀

Objective function. Finally, the linear objective function is given as follows:

$$w(\mathbf{x}, \mathbf{s}, \mathbf{r}) = \sum_{i=1}^{n} \left[\left(\sum_{j\neq k} \pi_i(j,k) s^i_{jk}\right) + \left(\sum_{j=-m}^{m} \alpha_i(j) r^i_j\right) + \iota_i x^i_\iota + a_i x^i_a + d_i x^i_\ell\right] \ .$$

Let us determine the values of the parameters r, s, t and a. We have introduced $O(|C|^2)$ variables and imposed $O(|C|^2)$ constraints on them, so $s = t = O(|C|^2)$. The coefficients a_j obey $\max_j |a_j| \leq O(|C|)$, and we use the bool_M operation with $M \in O(|C|^2)$, so $a = O(|C|^2)$.

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.

Scoring protocol $\mathbf{s = (s_1, \ldots, s_{|C|})}$. We introduce variables σ_c where σ_c is the number of points a voter gives candidate c. Then, an "active" copy of each new variable (denoted σ^a_i for $i = 1, \ldots, |C|$) is created such that we can disregard the contribution of latent voters. To

do this we use Lemma 9 with $x := \sigma_i$, $z := x^a$, and $x^z := \sigma_i^a$ for every $i = 1, \dots, |C|$. Then we add the following globally uniform constraints:

$$\textstyle\sum_{i=1}^{n} \sigma_j^a \;<\; \sum_{i=1}^{n} \sigma_1^a \text{ for } j = 2, \dots, |C| \;.$$

Any C1 rule $\mathcal{R}$. We guess the resulting $<_M$ such that c_1 is a winner with respect to $\mathcal{R}$; there are $O(3^{|C|^2})$ guesses. Knowing $<_M$ means that for any pair c_j, c_k of distinct candidates, we know if $v(c_j, c_k) > v(c_k, c_j)$, $v(c_k, c_j) > v(c_j, c_k)$ or $v(c_j, c_k) = v(c_j, c_k)$. We thus add:

$$\textstyle\sum_{i=1}^{n} x_{jk}^a > \sum_{i=1}^{n} x_{kj}^a (\text{if } c_j <_M c_k) \text{ and } \sum_{i=1}^{n} x_{jk}^a = \sum_{i=1}^{n} x_{kj}^a (\text{if } v(c_j, c_k) = v(c_k, c_j)) \;.$$

Maximin rule. For c_1 to be winner with the maximin rule means that there is a $B \in \{1, \dots, |V|\}$ such that $v_*(c_1) = B$, while for all $c \in C \setminus \{c_1\}$, $v_*(c_j) < B$. That, in turn, means, that for every candidate c_j there is a candidate $c_{j'}$ such that $v(c_j, c_{j'}) < B$. Guess B and $c_{j'}$ for every c_j; there are at most $n \cdot |C|^2$ guesses. Then add the following constraints:

$$\textstyle\sum_{i=1}^{n} x_{1j}^a \geq B \qquad \text{and } \sum_{i=1}^{n} x_{jj'}^a < B, \quad j = 2, \dots, |C| \;.$$

Bucklin. Guess the number $|\overline{V}_a| \in \{1, \dots, n\}$ of active voters and set $\sum_{i=1}^{n} x_a^i = |\overline{V}_a|$. Then, guess the winning round k and note that the winning score will be larger than $|\overline{V}_a|/2$. Altogether there are $O(|C||V|)$ guesses. Similarly to scoring protocols, we introduce variables σ_j (number of points for candidate j in k-approval) and $\tilde{\sigma}_j$ (number of points for candidate j in $(k-1)$-approval). We omit the details of how to split variables into active and latent:

$$\sigma_j = \text{bool}_{|C|}(x_j < k) \qquad \text{and } \tilde{\sigma}_j = \text{bool}_{|C|}(x_j < k-1), \quad j = 1, \dots, |C| \;.$$

Then, the winning condition is expressed as:

$$\sum_{i=1}^{n} \sigma_1^a > |\overline{V}_a|/2 \;\bigwedge\; \sum_{i=1}^{n} \sigma_j^a < \sum_{i=1}^{n} \sigma_1^a \text{ for } j = 2, \dots, |C| \;\bigwedge\; \sum_{i=1}^{n} \tilde{\sigma}_c^a < |\overline{V}_a|/2, c \in C \;.$$

SP-AV. In SP-AV, each candidate c receives a point if it ranks above the approval count. As before, we introduce variables σ_c for points received by a candidate c and split them into active and latent (again using Lemma 9).

$$\sigma_j = \text{bool}_{|C|}(x_j < x_\alpha) \text{ for } j = 1, \dots, |C| \;\bigwedge\; \textstyle\sum_{i=1}^{n} \sigma_j^a < \sum_{i=1}^{n} \sigma_1^a \text{ for } j = 2, \dots, |C| \;.$$

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 $|\overline{V}_a|$ of active voters; there are $O(|V| \cdot |C|)$ guesses. (SP-AV is used exactly when the winning score is less than $|\overline{V}_a|/2$.) If Bucklin is used, we need a slight modification to take support changes into account. Instead of $\sigma_j = \text{bool}_{|C|}(x_j < k)$ we have $\sigma_j = [\text{bool}_{|C|}(x_j < k) \wedge \text{bool}_{|C|}(k \leq x_\alpha)]$; similarly for $\tilde{\sigma}_j$.

As argued, for each rule we introduced $O(|C|^2)$ variables and locally uniform constraints, and $O(|C|^2)$ globally uniform constraints. Thus, by Theorem 7 we get an n-fold matrix with parameters $r = s = t = a = O(|C|^2)$. Proposition 6 is then used to solve the n-fold IP in time $2^{O(|C|^6 \log |C|)} n^3$. 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 run time is the Kemeny rule:

Kemeny. For c_1 to be a Kemeny winner, there has to be a ranking $\succ_{R^*}$ of the candidates that ranks c_1 first and $\succ_{R^*}$ maximizes the total agreement with voters $\sum_{i=1}^{|V|}\{(a,b) \mid ((a \succ_{R^*} b) \Leftrightarrow (a \succ_i b)), a, b \in C\}$ among all rankings. In other words, the number of swaps sufficient to transform every $\succ_i$ into $\succ_{R^*}$ is smaller than the number of swaps needed to transform every $\succ_i$ into any other $\succ_{R'}$ where c_1 is not first.

We guess the ranking $\succ_{R^*}$; then we introduce variables x_R^i for $R \in \{R^*\} \cup \{R' \mid c_1 \text{ is not first in } R'\}$ so that x_R^i is the number of swaps needed to transform $\succ_i^\Gamma$ into $\succ_R$, splitting them into active and latent as before. Then, we introduce the necessary constraints:

$$\begin{aligned} x_R = {} & \textstyle\sum_{j\neq k}[\mathrm{sign}_{|C|}(x_j - x_k) = \mathrm{sign}_{|C|}(\mathrm{rank}(k,R) - \mathrm{rank}(j,R)] && \text{for all } R, \\ \textstyle\sum_{i=1}^n x_R^{ai} > {} & \textstyle\sum_{i=1}^n x_{R^*}^{ai} && \text{for } R \neq R^* \ . \end{aligned}$$

This solves Kemeny-Multi-Bribery in time $2^{O(|C|!^6)}n^3$, and completes proving Theorem 1.

References

1. John J. Bartholdi III, Craig A. Tovey, and Michael A. Trick. Voting schemes for which it can be difficult to tell who won the election. *Soc. Choice Welfare*, 6(2):157–165, 1989.
2. Dorothea Baumeister, Piotr Faliszewski, Jérôme Lang, and Jörg Rothe. Campaigns for lazy voters: truncated ballots. In *Proc. AAMAS 2012*, pages 577–584, 2012.
3. Dorothea Baumeister and Jörg Rothe. Preference aggregation by voting. In Jörg Rothe, editor, *Economics and Computation. An Introduction to Algorithmic Game Theory, Computational Social Choice, and Fair Division.*, pages 197–326. Springer, 2016.
4. Nadja Betzler, Susanne Hemmann, and Rolf Niedermeier. A multivariate complexity analysis of determining possible winners given incomplete votes. In *Proc. IJCAI 2009*, pages 53–58, 2009.
5. Steven J. Brams and Peter C. Fishburn. Voting procedures. In Katora Suzumura Kenneth J. Arrow, Armatya K. Sen, editor, *Handbook of Social Choice and Welfare*, volume 19 of *Handbooks in Economics*, pages 173–236. Elsevier, 2002.
6. Robert Bredereck, Jiehua Chen, Piotr Faliszewski, Jiong Guo, Rolf Niedermeier, and Gerhard J. Woeginger. Parameterized algorithmics for computational social choice: Nine research challenges. *Tsinghua Sci. Tech.*, 19(4):358–373, 2014.
7. Robert Bredereck, Jiehua Chen, Piotr Faliszewski, André Nichterlein, and Rolf Niedermeier. Prices matter for the parameterized complexity of shift bribery. In *Proc. AAAI 2014*, pages 552–558, 2014.
8. Robert Bredereck, Jiehua Chen, Sepp Hartung, Stefan Kratsch, Rolf Niedermeier, Ondrey Suchý, and Gerhard J. Woeginger. A multivariate complexity analysis of lobbying in multiple referenda. *J. Artificial Intelligence Res.*, 50:409–446, 2014.
9. Robert Bredereck, Piotr Faliszewski, Rolf Niedermeier, Piotr Skowron, and Nimrod Talmon. Elections with few candidates: Prices, weights, and covering problems. In *Proc. ADT 2015*, volume 9346 of *Lecture Notes Comput. Sci.*, pages 414–431, 2015.
10. Robert Bredereck, Piotr Faliszewski, Rolf Niedermeier, Piotr Skowron, and Nimrod Talmon. Complexity of shift bribery in committee elections. In *Proc. AAAI 2016*, pages 2452–2458, 2016.
11. Robert Bredereck, Piotr Faliszewski, Rolf Niedermeier, and Nimrod Talmon. Large-scale election campaigns: Combinatorial shift bribery. *J. Artificial Intelligence Res.*, 55:603–652, 2016.
12. Bruno Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Inform. and Comput.*, 85(1):12–75, 1990.
13. Britta Dorn and Ildikó Schlotter. Multivariate complexity analysis of swap bribery. *Algorithmica*, 64(1):126–151, 2012.

14 Edith Elkind, Piotr Faliszewski, and Arkadii Slinko. Swap bribery. In *Proc. SAGT 2009*, volume 5814 of *Lecture Notes Comput. Sci.*, pages 299–310, 2009.

15 Edith Elkind, Piotr Faliszewski, and Arkadii Slinko. Swap bribery, 2009. URL: `https://arxiv.org/abs/0905.3885`.

16 Herbert B. Enderton. *A Mathematical Introduction to Logic*. Academic Press, 1972.

17 Piotr Faliszewski. Nonuniform bribery. In *Proc. AAMAS 2008*, pages 1569–1572, 2008.

18 Piotr Faliszewski, Edith Hemaspaandra, and Lane A. Hemaspaandra. How hard is bribery in elections? *J. Artificial Intelligence Res.*, 40:485–532, 2009.

19 Piotr Faliszewski, Edith Hemaspaandra, and Lane A. Hemaspaandra. Multimode control attacks on elections. *J. Artificial Intelligence Res.*, 40:305–351, 2011.

20 Piotr Faliszewski, Edith Hemaspaandra, Lane A. Hemaspaandra, and Jörg Rothe. Llull and copeland voting computationally resist bribery and constructive control. *J. Artificial Intelligence Res.*, 35:275–341, 2009.

21 Robert Ganian and Sebastian Ordyniak. The complexity landscape of decompositional parameters for ILP. In *Proc. AAAI 2016*, pages 710–716, 2016.

22 Raymond Hemmecke, Shmuel Onn, and Lyubov Romanchuk. n-fold integer programming in cubic time. *Math. Program.*, 137(1-2, Ser. A):325–341, 2013.

23 Hendrik W. Lenstra, Jr. Integer programming with a fixed number of variables. *Math. Oper. Res.*, 8(4):538–548, 1983.

24 Jesus A. De Loera, Raymond Hemmecke, and Matthias Köppe. *Algebraic and Geometric Ideas in the Theory of Discrete Optimization*, volume 14 of *MOS-SIAM Series on Optimization*. SIAM, 2013.

25 Shmuel Onn. Nonlinear discrete optimization. *Zurich Lectures in Advanced Mathematics, European Mathematical Society*, 2010.

26 Ildikó Schlotter, Piotr Faliszewski, and Edith Elkind. Campaign management under approval-driven voting rules. In *Proc. AAAI 2011*, pages 726–731, 2011.

27 Hobart P. Young. Extending Condorcet's rule. *J. Econ. Theory*, 16:335–353, 1977.