Multi-Party Campaigning

Martin Koutecký,[1] **Nimrod Talmon**[2]
[1] Computer Science Institute, Charles University, Prague, Czech Republic,
[2] Ben-Gurion Univesity, Be'er Sheva, Israel
koutecky@iuuk.mff.cuni.cz, talmonn@bgu.ac.il

Abstract

We study a social choice setting of manipulation in elections and extend the usual model in two major ways: first, instead of considering a single manipulating agent, in our setting there are several, possibly competing ones; second, instead of evaluating an election after the first manipulative action, we allow several back-and-forth rounds to take place. We show that in certain situations, such as in elections with only a few candidates, optimal strategies for each of the manipulating agents can be computed efficiently. Our algorithmic results rely on formulating the problem of finding an optimal strategy as sentences of Presburger arithmetic that are short and only involve small coefficients, which we show is fixed-parameter tractable – indeed, one of our contributions is a general result regarding fixed-parameter tractability of Presburger arithmetic that might be useful in other settings. Following our general theorem, we design quite general algorithms; in particular, we describe how to design efficient algorithms for various settings, including settings in which we model diffusion of opinions in a social network, complex budgeting schemes available to the manipulating agents, and various realistic restrictions on adversary actions.

Introduction

Within computational social choice, the study of external agents wishing to rig a given election has received extensive study, most notably by studying problems of election control and bribery in elections (Faliszewski and Rothe 2015): In these problems, an external agent aims at altering the result of a given election; in election control, such an agent can change the structure of the election, usually by adding/removing voters/candidates, while in bribery problems, such an agent can change the way certain voters vote.

Existing literature concentrates only on cases in which just one external agent (e.g., one briber) exists (Faliszewski, Hemaspaandra, and Hemaspaandra 2009). Some papers take a more game-theoretic approach, but from a cooperative perspective (Bachrach, Elkind, and Faliszewski 2011), while we are concerned with a non-cooperative perspective. Furthermore, all existing work (to the best of our knowledge) only deals with one action of bribery or control which is followed (immediately, or potentially with an additional opinion diffusion step (Faliszewski et al. 2018)) by evaluating the resulting election with respect to the agent's initial goal.

Here we study a very rich and realistic model, in which several competing agents, each agent with its own objective, alter the election in multiple rounds, responding to each other's actions. In particular, our model captures the setting of multi-party campaigning, in which several parties are campaigning over the same election. In the most basic model we consider k bribers operating on an election with m alternatives and n voters. There is a process, running for ℓ turns, in which, in each turn, each briber can change certain votes of their choice, respecting some budget limit. At the end of the process, a winner of the resulting election is chosen according to some voting rule $\mathcal{R}$. Our goal is to compute optimal bribery strategies for each agent.

To this end, we construct a Presburger arithmetic formula (for background see a recent guide (Haase 2018)) that is true if and only if a given agent has a winning strategy which respects the budget; indeed, there is a procedure to recover this strategy if it exists. Furthermore, by ensuring that this formula is short and contains only small coefficients, we are able to compute such optimal strategies efficiently; this follows by combining a careful analysis of the algorithm of Cooper (1972) for deciding Presburger arithmetic formulas together with algorithms for convex integer optimization in small dimension (Grötschel, Lovász, and Schrijver 1993; Dadush, Peikert, and Vempala 2011).

Our contribution here is a complexity analysis of deciding Presburger arithmetic formulas, taking into account multiple parameters, showing that only some of them undergo a (necessary) combinatorial explosion. In particular, we infer that deciding Presburger arithmetic and even optimizing a convex function over its satisfying assignments is fixed-parameter tractable parameterized the length of the given formula and its largest coefficient. A problem Π is *fixed-parameter tractable (*FPT*) parameterized by* k if it has an algorithm running in time $f(k)\operatorname{poly}(n)$ for any instance of size n. Classifying a problem as FPT gives a formal way of saying that a special class of instances (those with small values of the parameter k) are easy to solve for a problem which is hard in general.

In essence, Presburger arithmetic contains logical formulas over linear constraints (this stands in contrast with Peano arithmetic, which also permits multiplication of variables).

Thus, it is possible to design the formulas described above in a modular way incorporating "hooks" within the basic model described above, which allow future extensions by plugging in more complex formulas; we later demonstrate several such extensions using this framework.

In particular, we show that our basic model can be drastically extended to include settings in which **(1)** voter types represent complex voter differences; **(2)** the budgets available to the agents change between turns, perhaps depending on the intermediate election at each turn, representing campaign polling and fundraising interaction; **(3)** settings in which voters are embedded in a social network and a diffusion process causes voters to update their votes; and **(4)** settings in which various restrictions on the strategies available to the agent are present, allowing for more realistic modeling. Importantly, for these extensions, computational efficiency follows similarly to the basic model.

Our Contributions

Our main contributions are as follows:

1. We suggest a useful model that captures the natural setting of multi-party campaigning, including rich model settings, incorporating opinion diffusion in social networks and involved budgeting schemes.
2. We devise efficient algorithms for finding optimal strategies for agents in these models, by a reduction to optimization over satisfying assignments of formulas in Presburger arithmetic.
3. We show that Presburger arithmetic convex minimization is fixed-parameter tractable wrt. the length of the given formula and the absolute value of its largest coefficient. Indeed, this result might be useful in other contexts.

Related Work

There is vast literature on control and bribery (Faliszewski and Rothe 2015); we mention the paper of Elkind et al. (2009), who proposed the notion of swap bribery, in which the external agent (i.e., the briber) pays for each swap it causes in a voter's preference order, and that of Faliszewski et al. (2009), who further studied the complexity of various bribery problems. To the best of our knowledge, there are no works dealing with multiple external agents manipulating a given election. (We do mention recent work that consider multiple bribers (Zhou et al. 2019; Grandi and Turrini 2016; Grandi, Stewart, and Turrini 2018), however in the different setting of a ranking system; in particular, our technical contribution differs largely from these works.)

Our algorithmic techniques allow us to show fixed-parameter tractability (FPT) of our problems. FPT algorithms for bribery problems have been studied quite extensively (Bredereck et al. 2016; Dorn and Schlotter 2012), also for more complex social choice settings such as multiwinner elections (Faliszewski, Skowron, and Talmon 2017). We differ greatly from these works as we consider the more general case containing several manipulating agents and multiple back-and-forth rounds. As we show later, our model can accommodate certain diffusion processes that occur, e.g., when agents live in a social network. In this context, we mention the work of Bredereck and Elkind (2017) and Faliszewski et al. (2018) on the interchange between bribery actions and opinion diffusion in social networks.

Our algorithmic framework is presented for the representation of elections as society vectors, following the concept used by Knop et al. (2018a) for studying bribery problems and also used later by Faliszewski et al. (2018). This representation is very general and serves as a unifying framework to describe extensions to our basic model.

Algorithmically, our approach is based on formulating combinatorial problems as (linear) minimization over satisfying assignments of Presburger arithmetic formulas. In this context, the overall idea is quite similar to the use of Lenstra's celebrated result (Lenstra 1983) implying that deciding ILP is FPT wrt. the number of integer variables. Other FPT results originating from the theory of integer programming that have found use in computational social choice are algorithms for n-fold IP (Eisenbrand et al. 2019) and parametric ILP (Eisenbrand and Shmonin 2008). Some recent work in computational social choice that uses some of these tools and share the same general flavor (Bredereck et al. 2020; Knop, Koutecký, and Mnich 2018b, 2017).

Presburger Arithmetic has been introduced by Presburger and Tarski in 1929 (Presburger 1929) and we built on Cooper's algorithm from 1972 (Cooper 1972). Nguyen and Pak (2019) recently resolved a major open problem by proving that even deciding short PA formulas is NP-hard (and beyond) if they are allowed to contain large coefficients. Klaedtke (2008) gave more refined bounds for the automata approach to deciding PA formulas, however, even his bounds do not make the distinction between coefficients and constants which is necessary to obtain our algorithmic result.

PA is FPT

Let m, n be integers. We define $[m, n] := \{m, m+1, \ldots, n\}$ and $[n] := [1, n]$. Throughout, we reserve bold face letters (e.g., $\mathbf{x}, \mathbf{y}$) for vectors. For a vector $\mathbf{x}$ its i-th coordinate is x_i. We write $\mathbf{ax}$ for the dot product of vectors $\mathbf{a}$ and $\mathbf{x}$.

We wish to develop efficient algorithms that find optimal strategies for agents that are manipulating a given election. Our approach is to write a formula in Presburger arithmetic (here we shorten to PA; this is not to be confused with Peano arithmetic) with a vector $\mathbf{m}$ of free variables such that the satisfying assignments are bribery actions corresponding to a first move in a winning strategy. Here we first provide a brief introduction to PA, and later show that optimizing over the satisfying assignments of a PA formula can be done efficiently if some of its parameters are bounded, as will be the case for the formulas modeling winning strategies.

PA is a useful logic to reason about numbers. Intuitively, PA can be viewed as Integer Linear Programming (ILP) enriched with logical connectives and quantifiers. For two formulas Φ and Ψ, we denote their equivalence by $\Phi \cong \Psi$.

Definition 1 (Extended Presburger Arithmetic (PA))**.** *An* atom *(or* atomic formula*) is a linear inequality* $\mathbf{ax} \leq b$ *or a congruence* $\mathbf{ax} \equiv b \mod p$*, with* $\mathbf{a} \in \mathbb{Z}^n$ *and* $b, p \in \mathbb{Z}$*. We call* $\mathbf{t} \equiv \mathbf{ax} + b$ *for some* $\mathbf{a}$ *and* b *a* term*. A* formula

is obtained by taking Boolean combinations of atoms using the standard logical connectives ($\wedge, \vee, \implies, \neg$, etc.) and by existential and general quantifiers $\exists, \forall$, respectively. Denote by PA *the set of all PA formulas. A* literal *is an atom or its negation. A variable is* bound *in a formula φ if it appears in a quantifier, and it is* free *otherwise. If $\mathbf{x}$ is a vector of the free variables of a formula φ, we write $\varphi(\mathbf{x})$.*

Remark 1. *The term "extended" in the definition above refers to the congruence atoms that are not present in the original language as defined by Presburger (Presburger 1929); however, it is typical to speak of PA as this extended language because it allows for quantifier elimination, unlike PA without congruence atoms. For detailed definitions see Klaedtke (2008).*

We provide some further useful notation below. For $\varphi \in$ PA we define $\mathsf{T}(\varphi)$ to be the set of all atoms of φ of the form $\mathbf{ax} \leq b$, $\mathsf{D}(\varphi)$ to be the set of all atoms of the form $\mathbf{ax} \equiv b \mod p$, $\mathsf{L}(\varphi)$ to be the number of symbols of φ (i.e., the number of atoms, logical connectives, and quantifiers),[1] the *maximum coefficient* $\alpha(\varphi)$ to be the maximum $\|\mathbf{a}\|_\infty$ and p contained in any of its atoms, and the *maximum constant* $\beta(\varphi)$ to be the largest right hand side b in any of its atoms.

In this section, we aim at solving the following problem.

PRESBURGER ARITHMETIC MINIMIZATION	
Input:	A PA formula $\varphi(\mathbf{x})$ with d free variables, and a function $f : \mathbb{R}^d \to \mathbb{R}$.
Find:	Find an assignment $\mathbf{x} \in \mathbb{Z}^d$ satisfying $\varphi(\mathbf{x})$ and minimizing $f(\mathbf{x})$ (among satisfying assignments), or reports unsatisfiability.

The main algorithmic result regarding Presburger arithmetic we prove here is the following.

Theorem 1. PRESBURGER ARITHMETIC MINIMIZATION *is fixed-parameter tractable parameterized by $\mathsf{L}(\varphi)+\alpha(\varphi)$ for any convex function f.*

Remark 2. *The full proof is deferred to the supplementary material; here we highlight the main ideas used within. The proof combines two elements. First, we perform a careful analysis of Cooper's algorithm (Cooper 1972) which uses quantifier elimination to find a quantifier-free formula ψ equivalent to φ. We show that $\mathsf{L}(\psi), \alpha(\psi) \leq g(\mathsf{L}(\varphi), \alpha(\varphi))$ and $\beta(\psi) \leq g(\mathsf{L}(\varphi), \alpha(\varphi)) \cdot \beta(\varphi)$ for some function g. While Cooper's algorithm is by now textbook material (Bradley and Manna 2007), it has not yet been recognized as a fixed-parameter algorithm, and the complexity bound we provide is not explicitly stated (as far as we know) in any existing work.*

Second, ψ is transformed into disjunctive normal form (DNF), yielding a formula $\psi_1 \vee \psi_2 \vee \cdots \vee \psi_K$ such that an assignment $\mathbf{x}$ satisfies ψ if and only if it satisfies some ψ_i, $i \in [K]$, with each ψ_i being a conjunction of linear inequalities or congruences. Such a conjunction can be then turned into a system of only linear constraints (by linearizing the congruences) and then one can apply any FPT *algorithm for convex minimization in small dimension (Grötschel, Lovász, and Schrijver 1993; Dadush, Peikert, and Vempala 2011) to each of these systems and return the best result among all.*

[1] Note that this definition is different than the standard definition of the length of a formula, which uses unary encoding of numbers.

A Basic Model with Multiple Bribers

Here we describe a basic model of an election with multiple manipulating agents. Informally, we wish to model a situation containing a set of voters where each manipulating agent wishes to make its preferred candidate win in an election that occurs eventually, and, to this end, can, at a certain cost, alter the opinions of several voters. We first provide some preliminaries and then describe our formal model.

Elections. An (ordinal) 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, represented via a *preference order* $\succ_v$ which is a total order over C. We often identify a voter v with her preference order $\succ_v$. Denote by $\mathrm{rank}(c, v)$ the rank of candidate c in $\succ_v$; v's most preferred candidate has rank 1 and her least preferred candidate has rank $|C|$. For distinct candidates $c, c' \in C$, write $c \succ_v c'$ if voter v prefers c over c' (i.e., v ranks c higher than she ranks c').

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*. Many voting rules have been considered in the social choice literature. Perhaps the simplest voting rule is Plurality, where the winner is an alternative which is ranked first by the largest number of voters. As another example, the *Borda rule* selects as a winner a candidate whose average ranking over the voters is the highest, that is, a candidate c gets $m - \mathrm{rank}(c, v)$ from each candidate v, and the candidate with the most points wins.

Swaps and Swap Bribery. Let (C, V) be an election and let $\succ_v \in V$ be a voter. For candidates $c, c' \in C$, a *swap* $s = (c, c')_v$ corresponds to an exchange between 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 $\mathrm{rank}(c, v) = \mathrm{rank}(c', v) - 1$ (note that this is indeed a swap of consecutive candidates). A set S of swaps is *admissible in* $\succ_v$ if each swap in S 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 extend this notation for applying swaps in several votes and denote it V^S (note that, in particular, the cost of V^S equals the sum of costs of $s \in S$). We specify v's cost of swaps by a function $\sigma^v : C \times C \to \mathbb{Z}$.

In the Swap Bribery problem, which we generalize here, we are given an election (C, V), a designated candidate $c^\star \in C$, and swap costs $\sigma^v : C \times C \to \mathbb{Z}$ for $v \in V$. The goal is to identify a set S of admissible swaps of minimum cost so that $c^\star$ wins the election (C, V^S) under the rule $\mathcal{R}$.

Societies and Moves. It will be useful to view an electorate not simply as a set of votes, but bundled by voter types. Let $\tau \in \mathbb{N}$ be the number of *types of voters*; note that $\tau \leq n$, and it can be significantly smaller. E.g., if voters are only distinguished by their preference orders, then $\tau \leq m!$ which might be much smaller than n. A *society* is a non-negative τ-dimensional integer vector $\mathbf{s} = (s_1, \ldots, s_\tau)$,

where s_j, $j \in [\tau]$, corresponds to the number of voters of type j in the society. In most problems, we are interested in modifying a society by moving people among types. A *move* is a vector $\mathbf{m} = (m_{1,1}, \ldots, m_{\tau,\tau}) \in \mathbb{Z}^{\tau^2}$. Intuitively, $m_{i,j}$ is the number of people of type i turning type j.

Definition 2. *A* change *is a vector $\boldsymbol{\Delta} = (\Delta_1, \ldots, \Delta_\tau) \in \mathbb{Z}^\tau$ whose elements sum up to 0. We say that $\boldsymbol{\Delta}$* is the change associated with a move $\mathbf{m}$ *if, for all $i \in [\tau]$, $\Delta_i = \sum_{j=1}^{\tau} m_{j,i} - m_{i,j}$, and we write $\boldsymbol{\Delta} = \Delta(\mathbf{m})$. A change $\boldsymbol{\Delta}$ is* feasible wrt. society $\mathbf{s}$ *if $\mathbf{s} + \boldsymbol{\Delta} \geq \mathbf{0}$, i.e., if applying the change $\boldsymbol{\Delta}$ to $\mathbf{s}$ results in a society (i.e., as long as there are enough voters from each type to move to other types).*

A useful notion is the *move costs vector*, which is a vector $\mathbf{c} = (c_{1,1}, \ldots, c_{\tau,\tau})$ in $(\mathbb{N} \cup \{+\infty\})^{\tau^2}$ satisfying the triangle inequality, i.e., $c_{i,k} \leq c_{i,j} + c_{j,k}$ for all distinct i, j, k.

Remark 3. *In this work we focus on moves that correspond to swap bribery actions. However, the actions of bribery, manipulation, and control that are expressible as moves in societies are much more general, as shown by Knop et al. (2018a). E.g., one may create, for each voter type $t \in [\tau]$, an "inactive" variant t', and moving a voter from t to t' corresponds to deleting this voter while moving a voter from t' to t corresponds to adding it – which are the actions considered in constructive control by adding/deleting voters. Hence, we encourage the reader to keep in mind that whenever we talk about swaps and bribery, many other types of actions may be substituted or added in that place.*

Our Basic Model

In our formal model we consider a society with n voters over m candidates. Bundling voters into types according to their preferences, we have that the number of voter types equals the number of preference orders existing in the society (thus, in particular, upper bounded by $m!$). We consider k bribers, $A_1, \ldots, A_k$, and a process of ℓ turns, such that, in each turn, each briber is given a budget of B to be used to change the society; in each turn, the bribers bribe the society in a round-robin fashion – A_1 first, then A_2, and so on. The bribery operations are all unit-cost swap bribery operations; in our context this means that, in each turn, each briber can cause at most B swaps of consecutive candidates. At the end of the ℓ-th round, we apply the Borda voting rule on the society to identify a winning candidate.

We are interested in an optimal strategy for A_1; we refer to A_1 as *our briber* where, w.l.o.g., the preferred candidate of A_1 is p.

Remark 4. *It is also possible to model the situation when a different agent plays the first turn, i.e., where they play and we respond, and so forth. In that case, we are able to verify, with the same complexity as all the results below, whether a winning strategy of our briber exists within a certain cost, but since the move that should be made will depend on the previous moves of the other bribers, there is no answer to be output other than "yes/no". After a move by the other bribers is realized, we are in the original setting, because it is our turn, and we can compute the optimal response.*

Optimal Strategies via Presburger Arithmetic

Here we wish to formulate the problem of finding optimal strategies for our briber in the model described above using formulas in PA; this will be useful algorithmically, as Theorem 1 then implies efficient algorithms if the relevant parameters are kept small.

First, we note that here we are interested in a worst-case adversarial model, so we assume that the only aim of the other bribers (i.e., $A_2, \ldots, A_k$) is to interfere with our briber; thus, while A_1 uses his bribery budget to try to make p win the election under $\mathcal{R}$ after the ℓ turns, all other bribers use their bribery budget to prevent alternative p from winning. Thus, in particular, the other bribers can collude and join forces (possibly also transferring money between themselves) to prevent p from winning. So, in fact, the number of other bribers does not make a difference in our current worst-case model; in particular, $k-1$ other bribers, each with a budget of B for each turn, are equal, from the point of view of our briber, to a single briber with a budget of $(k-1)B$ for each turn. Thus, below we assume that there is only one other briber, which we refer to as *the other briber*.

Formulating Optimal Strategies

Below we describe a formula Φ in PA; later we argue that Φ is satisfiable iff there is a strategy for our briber guaranteeing that our candidate wins.

We first describe some ingredients of Φ. First, $\mathbf{s}_1^1$ is the initial society. Then, $\mathbf{c}$ is the cost vector corresponding to unit-cost swap-briberies (i.e., $c_{i,j}$ is the cost of swapping from type i to type j, which is the number of inversions between the corresponding two permutations). We need the following auxiliary predicates:

- PossibleMove$(\mathbf{m}, \mathbf{s})$ is true for a move $\mathbf{m}$ and a society $\mathbf{s}$ if the resulting vector is a society, that is, PossibleMove$(\mathbf{m}, \mathbf{s}) \equiv \mathbf{s} + \boldsymbol{\Delta}(\mathbf{m}) \geq \mathbf{0}$.
- FeasibleMove$(\mathbf{m}, B)$ is true for a move $\mathbf{m}$ and an integer B if the number of swaps in move $\mathbf{m}$ is at most B that is, FeasibleMove$(\mathbf{m}, B) \equiv \mathbf{cm} \leq B$.
- ApplyMove$(\mathbf{m}, \mathbf{s}, \mathbf{s}')$ is true for a move $\mathbf{m}$ and two societies $\mathbf{s}, \mathbf{s}'$ if $\mathbf{s}'$ is the result of applying $\mathbf{m}$ to $\mathbf{s}$, that is, ApplyMove$(\mathbf{m}, \mathbf{s}, \mathbf{s}') \equiv \mathbf{s}' = \mathbf{s} + \boldsymbol{\Delta}(\mathbf{m})$. Since we prefer to view this as a function, we write $\mathbf{s}' =$ ApplyMove$(\mathbf{m}, \mathbf{s})$.
- BordaWinner$(p, \mathbf{s})$ is true for a society $\mathbf{s}$ if p is the Borda winner in it. This is encoded by observing that the Borda score of candidate c is $S_c = \sum_{t \in [\tau]} (m - \mathrm{rank}(c, t))$ (where $\mathrm{rank}(c, t)$ is the rank of c for voters of type t), and p is the unique Borda winner if $S_p > S_c$ for every candidate $c \neq p$, i.e., BordaWinner$(p, \mathbf{s}) \equiv \bigwedge_{c \in C \setminus \{p\}} S_c < S_p$. Note that S_c is just a shorthand for the aforementioned sum, so we need not introduce any new variables.

Figure 1 shows the general structure of Φ. In Φ, $\mathbf{m}_i^j$ represents the i-th move of our briber, if $j = 1$, or the other briber, if $j = 2$. $\mathbf{s}_i^j$ represents the society just before the i-th turn of our briber, if $j = 1$, or the other briber, if $j = 2$. Finally, $\mathbf{s}'$ represents the eventual society.

$$\Phi(\mathbf{m}_1^1) \equiv \text{PossibleMove}(\mathbf{m}_1^1, \mathbf{s}_1^1) \wedge \text{FeasibleMove}(\mathbf{m}_1^1, B) \wedge \mathbf{s}_1^2 = \text{ApplyMove}(\mathbf{m}_1^1, \mathbf{s}_1^1)$$
$$\forall \mathbf{m}_1^2 : \text{PossibleMove}(\mathbf{m}_1^2, \mathbf{s}_1^2) \wedge \text{FeasibleMove}(\mathbf{m}_1^2, B) \wedge \mathbf{s}_2^1 = \text{ApplyMove}(\mathbf{m}_1^2, \mathbf{s}_1^2)$$
$$\vdots$$
$$\exists \mathbf{m}_\ell^1 : \text{PossibleMove}(\mathbf{m}_\ell^1, \mathbf{s}_\ell^1) \wedge \text{FeasibleMove}(\mathbf{m}_\ell^1, B) \wedge \mathbf{s}_\ell^2 = \text{ApplyMove}(\mathbf{m}_\ell^1, \mathbf{s}_\ell^1)$$
$$\forall \mathbf{m}_\ell^2 : \text{PossibleMove}(\mathbf{m}_\ell^2, \mathbf{s}_\ell^2) \wedge \text{FeasibleMove}(\mathbf{m}_\ell^2, B) \wedge \mathbf{s}' = \text{ApplyMove}(\mathbf{m}_\ell^2, \mathbf{s}_\ell^2)$$
$$\wedge\, \text{BordaWinner}(p, \mathbf{s}')$$

Figure 1: General Structure of ϕ.

Note that the lines beginning with "$\exists$" are for our briber while the lines beginning with "$\forall$" are for the other briber, as we care whether there are bribery operations for our briber to choose that would be winning for any bribery operations that the other briber might choose. Moreover, the PossibleMove predicates make sure that we only consider possible moves, the FeasibleMove predicates make sure that we only consider feasible moves, and we make sure to update the current society by applying the ApplyMove function.

Finding Optimal Strategies

Given the formula Φ as defined above and an initial society $\mathbf{s}_1^1$, Theorem 1 computes an initial move $\mathbf{m}_1^1$ of a winning strategy of our briber if one exists, and otherwise reports that there is no winning strategy. (In fact, the first predicate FeasibleMove$(\mathbf{m}_1^1, B)$ may be removed and instead a more involved cost function may be minimized.)

To bound the complexity, we examine $\mathsf{L}(\Phi)$, $\alpha(\Phi)$, and $\beta(\Phi)$. The length $\mathsf{L}(\Phi)$ is bounded by a polynomial in the number of variables, which is $\mathcal{O}(\ell \cdot \tau^2)$, since for each round we have a constant number of society and move vectors, which are of dimensions τ and τ^2, respectively, and the number of rounds is $O(\ell)$. The largest coefficient $\alpha(\Phi)$ is bounded by $\|\mathbf{c}\|_\infty$, which is the largest swap distance between two permutations, which is $\mathcal{O}(m^2)$. It is crucial to note that the large input data, which is B and $\mathbf{s}_1^1$, only ever appear as constants (right hand sides), hence $\beta(\Phi) \leq \|B, \mathbf{s}_1^1\|_\infty$. Recall that Theorem 1 implies an FPT algorithm for parameters τ and ℓ if $\mathsf{L}(\Phi)$ and $\alpha(\Phi)$ are bounded by a function of these, which indeed is the case here.

Enriched Models

Above we described an efficient algorithm for our basic model. This basic model is, however, quite restricted. Luckily, our approach is very robust, thus rendering our algorithms quite general; indeed, as we show next, we have efficient algorithms in many cases for other, much less restricted models.

The basic observation is that, as we use PA, we can in fact design our formulas (in particular, Φ) in a modular way; then, we can adapt each module separately to accommodate for various model generalizations and variants. Figure 2 shows a modular and slightly more general version of Φ.

Specifically, for the basic model, PreConditions merely checks both PossibleMove and FeasibleMove, and the PostProcessing step leaves the society and budgets intact. Below we discuss various enrichments to the basic model presented above and describe how to define PreConditions and PostProcessing to formulate them, thus to allow for our algorithmic approach to operate on them as well.

Complex Budgeting Schemes

In the basic model, each briber had a fixed budget to be used separately for each turn. We discuss other options below.

Initial Budgets. We can allow the budget to be fixed at the beginning of the process. This would correspond to a campaign manager setting aside some amount to be used during the whole campaign. This can be formulated in the PostProcessing step, by decreasing from B_{i-1}^j the amount just used, i.e., setting $B_i^j = B_{i-1}^j - \mathbf{cm}_{i-1}^j$.

Individual Budgets. We can also easily make it so that our briber has a different budget than the other briber, by simply plugging this info into each B_i^j, $i \in [\ell], j \in \{1, 2\}$.

Chunked Budgets. We can allow the budget to be given in chunks. This would correspond to, say, a campaign manager assigning some amount to be used for each month (if one turn means one month).

This can be formulated in the PostProcessing step, by decreasing from B the amount just used, in addition to adding to B some amount, i.e., if C_i^j is the contribution to briber j in round i, then we would add the constraint $B_i^j = B_{i-1}^j - \mathbf{cm}_{i-1}^j + C_i^j$ to the PostProcessing formula.

Adaptive Budgets. We can even allow the budget to be adaptive; for example, that in each turn, the increase in the budget is a linear function that depends on the Borda score of the preferred candidate p_i (alternatively, on p for our briber and on the inverse Borda score of p for the other briber), i.e., $B_i^j = B_{i-1}^j - \mathbf{cm}_{i-1}^j + S_p(\mathbf{s}_{i-1}^j)$, where $S_p(\mathbf{s}_{i-1}^j)$ is the Borda score of p in the society $\mathbf{s}_{i-1}^j$. This would correspond to voters donating to the campaign as it runs depending on the intermediate poll results.

More generally, the budget for each briber is decided based on the society just before he plays. The dependence might be arbitrary, as long as it can be formulated in PA.

$$\Phi(\mathbf{m}_1^1) \equiv \text{PreConditions}(\mathbf{m}_1^1, \mathbf{s}_1^1, B_1^1) \wedge \mathbf{s}_1^2 = \text{ApplyMove}(\mathbf{m}_1^1, \mathbf{s}_1^1) \wedge (\bar{\mathbf{s}}_1^2, B_2^1) = \text{PostProcessing}(\mathbf{s}_1^2, \mathbf{m}_1^1, B_1^1)$$
$$\forall \mathbf{m}_1^2 : \text{PreConditions}(\mathbf{m}_1^2, \bar{\mathbf{s}}_1^2, B_1^2) \wedge \mathbf{s}_2^1 = \text{ApplyMove}(\mathbf{m}_1^2, \bar{\mathbf{s}}_1^2) \wedge (\bar{\mathbf{s}}_2^1, B_2^2) = \text{PostProcessing}(\mathbf{s}_2^1, \mathbf{m}_1^2, B_1^2)$$
$$\vdots$$
$$\exists \mathbf{m}_\ell^1 : \text{PreConditions}(\mathbf{m}_\ell^1, \bar{\mathbf{s}}_\ell^1, B_\ell^1) \wedge \mathbf{s}_\ell^2 = \text{ApplyMove}(\mathbf{m}_\ell^1, \bar{\mathbf{s}}_\ell^1) \wedge (\bar{\mathbf{s}}_\ell^2, B_{\ell+1}^1) = \text{PostProcessing}(\mathbf{s}_\ell^2, \mathbf{m}_\ell^1, B_\ell^1)$$
$$\forall \mathbf{m}_\ell^2 : \text{PreConditions}(\mathbf{m}_\ell^2, \bar{\mathbf{s}}_\ell^2, B_\ell^2) \wedge \mathbf{s}' = \text{ApplyMove}(\mathbf{m}_\ell^2, \bar{\mathbf{s}}_\ell^2) \wedge (\bar{\mathbf{s}}', B_{\ell+1}^2) = \text{PostProcessing}(\mathbf{s}', \mathbf{m}_\ell^2, B_\ell^2)$$
$$\text{WinningConditions}(p, \bar{\mathbf{s}}')$$

Figure 2: A modular and slightly more general version of Φ.

Voter Behavior

We can model several scenarios w.r.t. how voters respond to bribery. This is allowed by the fact that if we start with a society with a small number of types, the following refinements do not increase the number of types too much. When we speak of different cost functions, these would be a part of the PreConditions check, and they need to be encoded by a linear function with small coefficients in order for Theorem 1 to give a fixed-parameter algorithm.

Loyal Voters. One possibility is that, whenever a voter is bribed for the first time, then she stays loyal and would never be bribed again. This can be encoded by, for each type $t \in [\tau]$, introducing a new type t' which has the same preference order but the cost of moving it is infinite for every type except for itself. Thus, the resulting society has 2τ types.

Semiloyal Voters. A more relaxed notion of loyalty would be that, each time a voter is bribed, her price for being bribed again goes up. This is encoded by, for each type $t \in [\tau]$, introducing $2\ell + 1$ new types $t_0, t_1, \ldots, t_{2\ell}$, where a voter is of type t_j if their preference order corresponds to type t and they have been bribed j times in the process. The cost of moving a voter of type t_j to type t'_{j+1} can be set as needed to model the increase in cost, and the cost of moving from t_j to $t'_{j'}$ for any $j' \neq j + 1$ is ∞, unless $t = t'$ and $j = j'$, where the cost is zero. (Infinite costs are *not* modeled using a large enough integer as usual in some contexts, but rather by upper bounding by zero the corresponding coordinate of a move vector.)

Greedy Voters. A different option is that, in each turn, each voter remembers the bribing offer from our briber as well as the bribing offer from the other briber, and simply goes with the higher offer. Note that for this to be formulated we shall add the possibility of offering more money to bribe a certain voter. To model this, we need a list of all possible offers a voter can get, which is bounded by our assumption that the cost functions are bounded. Then, for each original voter type $t \in [\tau]$ and for each possible offer $o \in \mathbb{N}$, we create a new voter type t_o. A voter is of type t_o if their preference order is t and if their last accepted bribe offer was o. The cost function of moving from t_o to $t'_{o'}$ can be set as needed to model the observed behavior, and is ∞ for every $o' \leq o$ unless $t = t'$. (To be precise, this models the behavior where a briber is aware that paying o is not enough to move a voter; if we wanted to model that a briber decides to spend money but it has no effect, we could replace PreConditions with a function altering the move and the budget, so that we would charge the briber for the move he attempted to make, but only execute the move accepted by the voters. This would still give a fixed-parameter algorithm, but would lead to more clutter in the formula.)

Pay-off Functions and Winning Conditions

In the basic model we considered the single-winner Borda voting rule as defining the winning condition, and we implicitly considered a 0/1 pay-off function, getting a 0 if our candidate loses and getting a 1 if he wins. We can allow for more complex winning conditions and pay-off functions, corresponding also to more advanced social choice settings. Below, when discussing different pay-offs, we still encode this as a decision problem: is there a strategy (or what is the cheapest strategy) achieving a given pay-off. In particular, in the supplementary material we discuss encoding any ILP-definable single-winner voting rule (including all scoring rules, and others); encoding a setting in which the candidates, as well as the bribers, are embedded on a real line; and encoding multiwinner elections, e.g., the winning condition that requires p to be in a winning committee.

Adversarial Constraints

In the basic model we considered a worst-case assumption on the side of our briber in which the other bribers were only concerned with interfering with the goal of our briber. It is also natural to assume that each of the other bribers A_i, $i \in [2, k]$, has its own preferred alternative p_i, and the goal of briber A_i is to make p_i the winner of the election. This goal may be expressed (in broad terms) in A_i never considering moves that harm them, which means that A_1 does not need to ensure they will be able to respond to such moves. Note that this is significantly different from the collusion case considered before in that A_1 has more "maneuvering space" by not having to be overly pessimistic with respect to his opponent's moves.

In the supplementary material we provide details on how to encode a few specific possibilities, which depend on the social choice setting; note that now we cannot simply reduce the setting with several other bribers (i.e., with $k > 2$) to the setting with only one briber (i.e., with $k = 2$); thus, the length of the formula now also depends on k.

The specific settings we describe in the supplementary material consider an adversarial constraint that corresponds to "do not decrease the margin of victory of p_i"; and an adversarial constraint for multiwinner elections that corresponds to "do not decrease the ranking of p in the ranking of the alternatives induced by the score given to each alternative by the voting rule $\mathcal{R}$." Here we provide details on other specific adversarial constraints.

Agent Far-sightedness The purpose of this section is in getting away from the overly pessimistic assumption that other agents are somehow willing to harm themselves. However, perhaps *not* accounting for any moves of opponents by which they harm themselves is overly naïve and optimistic, because there may exist moves that cause harm in the short term but are in fact beneficial in the long run. To account for this, we introduce the notion of z-far-sightedness: an agent is *z-far-sighted*, $z \in \mathbb{N}$, if they are only willing to perform moves which are guaranteed (no matter what strategy other agents choose) to benefit them after z rounds of the campaigning game or by the time the game ends (i.e., by the time the election is evaluated), whichever comes first.

With the knowledge of the total number of rounds ℓ, we can implement z-far-sightedness in our model as part of the PreConditions check. We only sketch the approach here because giving a full description would be notationally cumbersome. So, we construct a formula $\Phi_{i,z,\ell'}$ expressing the z-far-sightedness condition for agent i in round ℓ'; the structure of $\Phi_{i,z,\ell'}$ is analogous to Φ but only considers $\min\{z, \ell - \ell'\}$ rounds, and it is constructed from the perspective of the considered agent.

However, we need to account for the fact that other agents are also far-sighted. This means that we have to construct $\Phi_{i,z,\ell'}$ recursively. A possible issue is where the recursion terminates: this is where we need the foreknowledge that the game ends in ℓ rounds. Specifically, in order to construct $\Phi_{i,z,\ell'}$, we need to have constructed $\Phi_{i',z,\ell''}$, where i' is "the next agent" $i' = i + 1 \mod k$, and ℓ'' is either ℓ' if i was not the last agent to play in round ℓ', and $\ell' + 1$ otherwise. Hence, the recursion depth in the construction of $\Phi_{i,z,\ell'}$ is at most $k \cdot \ell$, and the length of the resulting formula is bounded by a function of τ, k, and ℓ, again yielding an FPT algorithm when parameterized by τ, k, and ℓ.

Diffusion Processes

In the basic model the only way by which votes have changed is through bribery operations. Very naturally, we can accommodate other ways to change votes, most notably diffusion processes, in which voters are assumed to reside on a social network and update their votes based on the votes of their neighbors (Grandi 2017). Faliszewski et al. (2018) have defined a very general class of opinion diffusion processes, called ILP-definable processes, which do not consider each voter individually but group them by types (note that, like in our models, the types may be more refined than just by preference orders).

Any ILP-definable diffusion process can be encapsulated within the PostProcessing step. In particular, we need to specify a number of diffusion steps to happen after each bribery operation, which translates into the length of the resulting PostProcessing formula. Note that some ILP-definable diffusion processes have been shown to converge in a number of steps bounded by the number of voter types, hence if such process is considered, we may allow full convergence to happen after each bribery step. Otherwise, a time-bound between bribery steps perhaps translates into a small number of diffusion steps.

Discussion and Outlook

We developed an algorithmic framework for situations in which several bribers aim at rigging a given election, and showed that in many cases, finding optimal strategies can be done in time which depends super-polynomially only on the number of rounds and the number of alternatives in the election or, more generally, only on the number of voter types in the given election. Our framework is versatile and incorporates many features which make the resulting model highly realistic, in particular we can efficiently solve situations in which there are **(1)** complex voter types; **(2)** different budgets, including adaptive budgets, for each agent; **(3)** voters are embedded in a network and are affected by certain diffusion processes; **(4)** various realistic restrictions apply to the manipulating agents.

Next we discuss some avenues for future research.

Unbounded number of bribery rounds. Our results are parameterized by the number of rounds of the bribery-counterbribery exchange. It is known that deciding Presburger arithmetic sentences (a problem easier than PRESBURGER ARITHMETIC MINIMIZATION) has super-exponential complexity unless the length of the formula is bounded (Fischer and Rabin 1974), and even with bounded length but large coefficients, the complexity grows increasingly higher in the polynomial hierarchy with increasing number of quantifier alternations (Nguyen and Pak 2019). However, it remains open whether any of the proposed campaigning models are also hard when the number of rounds is not bounded.

We conjecture that the most basic model with unit-cost swap actions and identical budgets for each round is FPT for a constant number of bribers and parameterized by the number of candidates, because we believe that the identical budget will enforce some sort of repetitive structure in the strategies of bribing agents. However, we also conjecture that already when we allow the budgets to vary from round to round, the problem is NP-hard for constantly many bribers and candidates.

Presburger arithmetic and AI. Finally, Theorem 1 which we rely on, and particularly Cooper's algorithm for Presburger arithmetic, is a general and widely applicable tool. We are curious to see more applications in computational social choice, and, more generally, in the field of Artificial Intelligence. This is particularly interesting because Theorem 1 falls into the larger context of identifying fixed-parameter tractability in problems above NP in the polynomial hierarchy (of which PA is an example) which has only recently begun to be explored, and which has great relevance to AI (de Haan 2016).

Acknowledgements

Koutecký was partially supported by Charles University project UNCE/SCI/004 and by the project 19-27871X of GA ČR. Talmon was supported by the Israel Science Foundation (ISF; Grant No. 630/19).

References

Bachrach, Y.; Elkind, E.; and Faliszewski, P. 2011. Coalitional voting manipulation: A game-theoretic perspective. In *Proceedings of IJCAI '11*.

Bradley, A. R.; and Manna, Z. 2007. *The calculus of computation: decision procedures with applications to verification*. Springer Science & Business Media.

Bredereck, R.; and Elkind, E. 2017. Manipulating Opinion Diffusion in Social Networks. In *Proceedings of IJCAI '17*, 894–900.

Bredereck, R.; Faliszewski, P.; Niedermeier, R.; Skowron, P.; and Talmon, N. 2020. Mixed integer programming with convex/concave constraints: Fixed-parameter tractability and applications to multicovering and voting. *Theoretical Computer Science* .

Bredereck, R.; Faliszewski, P.; Niedermeier, R.; and Talmon, N. 2016. Complexity of shift bribery in committee elections. In *Proceedings of AAAI '16*.

Cooper, D. C. 1972. Theorem proving in arithmetic without multiplication. *Machine intelligence* 7(91-99): 300.

Dadush, D.; Peikert, C.; and Vempala, S. 2011. Enumerative lattice algorithms in any norm via M-ellipsoid coverings. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, 580–589. IEEE.

de Haan, R. 2016. *Parameterized Complexity in the Polynomial Hierarchy*. Ph.D. thesis, Springer.

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

Eisenbrand, F.; Hunkenschröder, C.; Klein, K.; Koutecký, M.; Levin, A.; and Onn, S. 2019. An Algorithmic Theory of Integer Programming. http://arxiv.org/abs/1904.01361.

Eisenbrand, F.; and Shmonin, G. 2008. Parametric integer programming in fixed dimension. *Mathematics of Operations Research* 33(4): 839–850.

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

Faliszewski, P.; Gonen, R.; Koutecký, M.; and Talmon, N. 2018. Opinion Diffusion and Campaigning on Society Graphs. In *Proceedings of IJCAI '18*, 219–225.

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

Faliszewski, P.; and Rothe, J. 2015. Control and Bribery in Voting. In Brandt, F.; Conitzer, V.; Endriss, U.; Lang, J.; and Procaccia, A. D., eds., *Handbook of Computational Social Choice*, chapter 7. Cambridge University Press.

Faliszewski, P.; Skowron, P.; and Talmon, N. 2017. Bribery as a measure of candidate success: Complexity results for approval-based multiwinner rules. In *Proceedings of AAMAS '17*, 6–14.

Fischer, M. J.; and Rabin, M. O. 1974. Super-exponential complexity of Presburger arithmetic. In Karp, R. M., ed., *comp*, 27–42.

Grandi, U. 2017. Social choice and social networks. In Endriss, U., ed., *Trends in Computational Social Choice*. AI Access.

Grandi, U.; Stewart, J.; and Turrini, P. 2018. The complexity of bribery in network-based rating systems. In *Proceedings of AAAI '18*.

Grandi, U.; and Turrini, P. 2016. A network-based rating system and its resistance to bribery. *arXiv preprint arXiv:1602.01258* .

Grötschel, M.; Lovász, L.; and Schrijver, A. 1993. *Geometric algorithms and combinatorial optimization*, volume 2 of *Algorithms and Combinatorics*. Springer-Verlag, Berlin, second edition.

Haase, C. 2018. A survival guide to presburger arithmetic. *ACM SIGLOG News* 5(3): 67–82.

Klaedtke, F. 2008. Bounds on the automata size for Presburger arithmetic. *ACM Transactions on Computational Logic* 9(2): 11.

Knop, D.; Koutecký, M.; and Mnich, M. 2017. Combinatorial n-fold integer programming and applications. *Mathematical Programming* 1–34.

Knop, D.; Koutecký, M.; and Mnich, M. 2018a. A unifying framework for manipulation problems. In *Proceedings of AAMAS '18*, 256–264.

Knop, D.; Koutecký, M.; and Mnich, M. 2018b. Voting and bribing in single-exponential time. *arXiv preprint arXiv:1812.01852* .

Lenstra, H. W. 1983. Integer programming with a fixed number of variables. *Mathematics of operations research* 8(4): 538–548.

Nguyen, D.; and Pak, I. 2019. Short Presburger arithmetic is hard. *SIAM Journal on Computing* 0: STOC17–1.

Presburger, M. 1929. Uber die vollstandigkeiteines gewissen systems der arithmetik ganzer zahlen, in welchen die addition als einzige operation hervortritt. In *Comptes-Rendus du ler Congres des Mathematiciens des Pays Slavs*.

Zhou, X.; Matsubara, S.; Liu, Q.; Liu, Y.; and Huang, G. 2019. Bribery in Rating System: A Game-Theoretic Perspective. *arXiv preprint arXiv:1911.10014* .