Journal of Artificial Intelligence Research 64 (2019) 109-145 Submitted 05/18; published 01/19

New Approximations for Coalitional Manipulation in Scoring Rules

Orgad Keller ORGAD.KELLER@GMAIL.COM
Department of Computer Science
Bar-Ilan University, Israel

Avinatan Hassidim AVINATAN@CS.BIU.AC.IL
Department of Computer Science
Bar-Ilan University, Israel

Noam Hazon NOAMH@ARIEL.AC.IL
Department of Computer Science
Ariel University, Israel

Abstract

We study the problem of *coalitional manipulation*—where k manipulators try to manipulate an election on m candidates—for any scoring rule, with focus on the Borda protocol. We do so in both the weighted and unweighted settings. For these problems, recent approximation approaches have tried to minimize k, the number of manipulators needed to make some preferred candidate p win (thus assuming that the number of manipulators is not limited in advance). In contrast, we focus on minimizing the *score margin* of p which is the difference between the maximum score of a candidate and the score of p.

We provide algorithms that approximate the optimum score margin, which are applicable to any scoring rule. For the specific case of the Borda protocol in the unweighted setting, our algorithm provides a superior approximation factor for lower values of k.

Our methods are novel and adapt techniques from multiprocessor scheduling by carefully rounding an exponentially-large configuration linear program that is solved by using the ellipsoid method with an efficient separation oracle. We believe that such methods could be beneficial in other social choice settings as well.

1. Introduction

The concept of *elections* lies at the heart of democratic societies, where it is the main tool for reaching a joint decision when considering several alternatives. While being so commonplace in human societies, it has also played a major role in multiagent systems, where it enables decisions by groups of intelligent agents (Ephrati & Rosenschein, 1993). In its essence, an election consists of n agents (also called voters) who need to decide on a winning candidate among m candidates. In order to do so—focusing on the well-studied preferential model—each voter reveals a ranking of the candidates according to his preference and the winner is then decided according to some protocol.

Ideally in voting, the voters should be truthful, that is, their reported ranking of the candidates shall be their true one. However, almost all voting rules are prone to manipulation: Gibbard (1973) and Satterthwaite (1975) show that for any reasonable preference-based voting system with at least 3 candidates, voters might benefit from reporting a ranking different from their true one in order to make sure that the candidate they prefer the most

wins. Furthermore, several voters might decide to collude and coordinate their votes in such a way that a specific candidate p (hereafter the *preferred candidate*) will prevail. Such a setting is reasonable especially when the voters are agents that are operated by one party of interest.

For some time, the hope of making voting protocols resistant to manipulation at least *in practice* relied on computational assumptions (Bartholdi, Tovey, & Trick, 1989). For several common voting protocols, it was shown that computing a successful voting strategy for the manipulators is NP-hard, see surveys by Faliszewski and Procaccia (2010) and by Conitzer and Walsh (2016). However, approximation algorithms and heuristics were devised in order to overcome the NP-hardness albeit with some compromises on the quality of the resulting strategy. This paper falls within this scheme.

We study the following two problems. In *(constructive) unweighted coalitional manipulation (UCM)*[1], we assume that k additional voters (hereafter the manipulators), all of them preferring a specific candidate p, can be added to the voting system, thus forming a coalition. We also assume that all n original voters (hereafter the non-manipulators) voted first (or equivalently, that the non-manipulators are truthful and that their preferences are known). Find a strategy for the manipulators telling each one how to vote so that p wins, if such a strategy exists. We call such a strategy a *p-winning strategy*. In the *weighted* variant *(WCM)*, the manipulators are weighted; essentially this means that the ballot of a manipulator having weight w is counted as if it was replaced with w unweighted copies of it. A related problem is *Bribery*, in which an interested party is willing to pay k (of the given n) voters to change their vote into a given ballot, such that p will prevail. Likewise, Bribery can be easily extended into a weighted variant as well.

While UCM and Bribery are usually defined as search problems in which we are asked to find a strategy, they can also be seen as optimization problems in which the goal is to find the minimum k enabling p to win. In this context, they can be treated as a measure of how far candidate p is from winning the election, or equivalently, they define the notion of a *margin* by which she might lose. Moreover, these problems also have destructive variants, in which the sole purpose of the manipulating party-of-interest is to prevent the currently leading candidate from winning. Likewise, the destructive variants serve as a measure of how robust the victory of the currently leading candidate is. Specifically, when considering the destructive variant of Bribery, this measure is known as the *margin of victory* and was extensively studied, e.g. by Magrino, Rivest, and Shen (2011), Cary (2011), and Xia (2012).

We focus on one of the most important families of voting rules, known as *(positional) scoring rules*. For a given *score vector* $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_m)$, where $\alpha_1 \geq \cdots \geq \alpha_m$, a scoring rule $\mathcal{R}_{\boldsymbol{\alpha}}$ is a voting rule where each voter awards $\alpha_1, \ldots, \alpha_m$ points to the candidates he ranked in places $1, 2, \ldots, m$, respectively. The winners are the candidates with the maximum aggregate score. Popular cases of scoring rules are the Plurality, Veto, and Borda voting rules.

The computational hardness of manipulating various scoring rules depends on the setting. Specifically, we distinguish between three cases in an increasing level of generality: UCM, WCM with polynomially-bounded integer weights (or weights that can be normalized to be such), and WCM with no restriction on the weights. Focusing on the Borda scoring

1. The problem was called CCUM, for "constructive coalitional unweighted manipulation", by Zuckerman, Procaccia, and Rosenschein (2009).

rule, it is NP-hard even for the UCM case (Davies, Katsirelos, Narodytska, & Walsh, 2011; Betzler, Niedermeier, & Woeginger, 2011). Please refer to Section 6 for a more elaborate overview of other scoring rules and previous work. For similar results related to Bribery, see a survey by Faliszewski and Rothe (2016).

In order to overcome the hardness, several lines of research ensued. Most notably, a string of approximation results for both manipulation (Zuckerman et al., 2009; Xia, Conitzer, & Procaccia, 2010; Brelsford, Faliszewski, Hemaspaandra, Schnoor, & Schnoor, 2008) and (various models of) Bribery (Brelsford et al., 2008; Faliszewski, 2008; Elkind, Faliszewski, & Slinko, 2009; Elkind & Faliszewski, 2010) has followed. In addition, other research venues focused on simple, practical heuristics for UCM, in particular for Borda-UCM (Davies, Katsirelos, Narodytska, Walsh, & Xia, 2014), or on analyzing the hardness of UCM in the average case (Procaccia & Rosenschein, 2007b, 2007a).

For UCM and WCM, recent research (Zuckerman et al., 2009; Xia et al., 2010) has focused on approximating the minimum k enabling p to win. For Borda-UCM, they showed that if a p-winning strategy for k manipulators exists, then they will find a p-winning strategy with at most one additional manipulator – in addition to the k given in the problem definition. A counterpart for Borda-WCM—this time approximating the overall manipulator weight required—was also provided.

This kind of approximation might seem a bit problematic. Firstly, the ability to add a manipulator is a strong operation, perhaps too strong; for instance, for Borda-UCM, whereby adding a manipulator adds $\Omega(m)$ to the difference between p and her highest-scoring competitor (hereafter we shall refer to a non-preferred candidate as a *competitor*). Secondly, while in some cases it might be reasonable that the party behind the manipulators can add another manipulator to the system, in many cases we do not expect this to be true. Furthermore, in the weighted case assumptions are needed to be made on the weight of the additional manipulators – also a problematic aspect. Instead, it is interesting to question what we can assert—assuming that the number of manipulators *cannot be changed*—on the ability to promote a specific candidate p given the votes of the non-manipulators and the value k (or equivalently, the length-k vector of manipulator weights).

We provide positive results of the following type:

> *If a manipulation strategy enabling p to win by a large-enough margin exists, we efficiently find a successful manipulation strategy that will cause p to win.*

Let the *score margin* be the minimum difference obtainable between the final score of the highest-scoring competitor and of p's. Now take the unweighted case as an example: assume that we can provide, for some function $f(k, m)$, an additive $f(k, m)$-approximation to the score margin. Then, if there exists a p-winning strategy such that this difference is at most $-f(k, m)$ (we can also say that "p can win by a margin of at least $f(k, m)$ points"), we can rest-assured that the algorithm will find a p-winning strategy.

This, in turn, boils down to approximating an upper-bound to the score of the highest-scoring competitor. Earlier research of this type focused only on cases where the number of candidates is bounded: for $\mathcal{R}_{\boldsymbol{\alpha}}$-WCM, Brelsford et al. (2008) provide a fully polynomial-time approximation scheme (FPTAS) for this upper bound. Compared to this line of work, we do not limit ourselves to a bounded number of candidates.

Approximating the score of the highest-ranked competitor had found another important use-case: in another work (Keller, Hassidim, & Hazon, 2018), the current authors showed that approximating the score of the highest-ranked competitor translates to an approximation to the number of bribed voters in the related Bribery problem.

1.1 Our Results and Contributions

We shall now describe the nature of our contribution. Given an election under a scoring rule $\mathcal{R}_{\boldsymbol{\alpha}}$ (see Section 2.1 for formal definitions), the goal is to determine the manipulator strategies S, such that the score margin $\max_{c\in\mathsf{C}\setminus\{p\}} S(c) - S(p)$ is minimized, where C is the candidate set and $S(c)$ is c's final score according to the strategy S. This boils down to minimizing $T = \max_{c\in\mathsf{C}\setminus\{p\}} S(c)$, i.e., the score of the highest-ranked competitor. Let T^{OPT} denote the minimum obtainable T, and let $g(\boldsymbol{\alpha})$ be the maximal difference between two values in $\boldsymbol{\alpha}$ that are $\widetilde{O}(\sqrt{m})$ entries away[2] ($g(\boldsymbol{\alpha})$ will be precisely defined later). Then:

- We provide a randomized algorithm for $\mathcal{R}_{\boldsymbol{\alpha}}$-UCM which finds a strategy that obtains a bound $T \leq T^{\mathsf{OPT}} + k \cdot g(\boldsymbol{\alpha})$ with a high probability (Theorem 8).
- For unary weights—or equivalently weights that can be normalized to become integers bounded by a polynomial in m, k—we provide a randomized algorithm for $\mathcal{R}_{\boldsymbol{\alpha}}$-WCM which finds a strategy that obtains a bound $T \leq T^{\mathsf{OPT}} + W \cdot g(\boldsymbol{\alpha})$ with a high probability, where W is the sum of manipulator weights (Theorem 15).
- For a general weight vector, we provide a randomized algorithm for $\mathcal{R}_{\boldsymbol{\alpha}}$-WCM which finds a strategy that obtains a bound $T \leq (1+\epsilon)T^{\mathsf{OPT}} + W \cdot g(\boldsymbol{\alpha})$, for any constant $\epsilon > 0$, with a high probability (Theorem 17).

In the above, notice that the first two results are additive approximations and thus translate directly into an additive approximation on the optimal score margin. The third one does not necessarily, but it does present somewhat of a trade-off w.r.t. Brelsford et al.'s (2008) FPTAS (see Lemma 3 therein): there, the authors supported only a constant number of candidates; while we remove this limitation, we add an additive $Wg(\boldsymbol{\alpha})$-factor to the approximation. It should be noted that the second result continues the line of research focusing on weighted settings where weights are not too large, or equivalently, are assumed to be encoded by a unary encoding. Such assumptions were previously made by Brelsford et al. (2008) and Faliszewski, Hemaspaandra, and Hemaspaandra (2009).

Taking Borda-UCM as an example, if there exists a p-winning strategy that will enable p to win by a margin of $\Omega(k\sqrt{m \log m})$ compared to the score of the highest-scoring competitor, our method will find a p-winning strategy (albeit with a possibly smaller margin) (Corollary 9). Similar guarantees apply in the weighted settings as well (Corollary 16).

Our methods have the following advantages over previous methods:

- We provide *provable* guarantees – as opposed to the heuristics made by Davies et al. (2014).

2. The $\widetilde{O}(\cdot)$ notation suppresses poly-logarithmic factors.

- Our algorithm generalizes to the weighted case – as opposed to the heuristics made by Davies et al. (2014), wherein it is not clear how to extend their algorithm to the weighted case.

- We provide an approximation based on a more granular metric – compared to the greedy algorithm of Zuckerman et al. (2009)—also called the Reverse algorithm[3]—for Borda-UCM. They provide an additive +1-approximation to k, the number of required manipulators, and though adding only a single extra manipulator seems like a minor operation, it is not. As mentioned, an extra manipulator implies the addition of $\Omega(m)$ points to the difference between p and her highest-scoring competitor. Focusing on the score margin instead, provides the ability to distinguish between cases where p loses by a small score margin, to cases where she loses by a large one.

- We provide a better approximation to the score margin – compared to the implicit score margin approximation provided by Reverse. Specifically, consider the Reverse algorithm for Borda-UCM, and now assume that adding extra manipulators is not allowed. We show (Claim 1) that their method implies no better than an additive $\Omega(m)$-approximation to the score of the highest-scoring competitor. Our approximation is therefore superior when k is $o(\sqrt{m/\log m})$.

- Our methods generalize to all scoring rules – as opposed to the work by Zuckerman et al. (2009). It should be noted that in Davies et al.'s (2014) work, the provided heuristics are empirically analyzed only w.r.t. Borda and two other voting rules (Nanson's and Baldwin's) which are not pure scoring rules.

- Our methods are not limited to a bounded number of candidates – as opposed to the work of Brelsford et al. (2008).

- Our results take into consideration the entire input at once. Compared to previous methods, our results are linear-programming-based, and not greedy. Thus, they make a decision based on the entire input, as opposed to repeatedly making a decision based on a greedy estimate w.r.t. some subset of the input.

- Our method provides a polynomially-computable, non-trivial lower bound to the value of T^{OPT} in the form of a solution to the linear program it uses as the starting-point to the approximation. In some cases the value T returned by our algorithm did not increase w.r.t. this linear program solution. In such cases, the linear program can be seen as an oracle proving that the algorithm did indeed find the optimal solution.

Focusing again on Borda, it is worth noting that while Borda is *resistant* to manipulation in the sense that Borda-UCM is NP-hard, it can be argued that empirically-speaking it is manipulable. This is due to Davies et al. (2014) showing that in the vast majority of experiments over Borda-UCM, where non-manipulator votes were taken from two distributions, their Average Fit heuristic finds an optimal strategy. However, our methods still provide a deeper, more theoretical insight into UCM for Borda as well as all other scoring rules. As mentioned, they also hold for WCM where Average Fit has no counterpart.

3. This name was given by Davies et al. (2011).

Our algorithms do not rely on any assumption on the distribution of voter ballots and weights and thus supply the guaranteed approximation factors regardless of such distributions. Specifically, they do not rely on the votes being independent of one another.

Our techniques are novel in that they employ the use of *configuration linear programs* (C-LPs), a method which is also used in the scheduling literature, namely for two well-studied problems: *scheduling on unrelated machines* (Svensson, 2012), and the so-called *Santa Claus problem* (Bansal & Sviridenko, 2006). See Section 6 for a discussion on these problems. C-LPs are used for the generation of an initial, invalid strategy, which is later modified to become valid. They are unique in the sense that they are linear programs that might have an exponential number of variables. This is an issue which we solve by converting them to an LP with an exponential number of constraints (the LP dual) and using the ellipsoid method with a polynomially-computable separation oracle (Khachiyan, 1980; Grötschel, Lovász, & Schrijver, 1981). Essentially these techniques solve an LP without being given all the constraints as input, but by requesting violated constraints from the oracle in an 'on demand' fashion. We also implemented our algorithm: as a result of not finding a library that would enable solving an LP this way, we simulated this by an iterative use of a general LP-solving library, each time adding a violated constraint based on running the separation oracle externally.

The current work subsumes a previous conference paper (Keller, Hassidim, & Hazon, 2017). In particular, the conference paper had studied only Borda-UCM. The current work generalizes this to any scoring rule, and to the weighted setting.

1.2 Outline of the Paper

We begin with preliminaries. Section 3 is dedicated to our $\mathcal{R}_{\boldsymbol{\alpha}}$-UCM algorithm where we first provide the motivation for this work by showing that for Borda, the Reverse algorithm offers no better than an $\Omega(m)$-approximation factor with respect to the score margin. We then present our $\mathcal{R}_{\boldsymbol{\alpha}}$-UCM algorithm which provides a better approximation for small values of k. This section also serves as a warm-up for the $\mathcal{R}_{\boldsymbol{\alpha}}$-WCM algorithm presented later, as the $\mathcal{R}_{\boldsymbol{\alpha}}$-UCM counterpart is simpler (both provide a similar approximation factor when applied to $\mathcal{R}_{\boldsymbol{\alpha}}$-UCM instances). In Section 4 we present our $\mathcal{R}_{\boldsymbol{\alpha}}$-WCM algorithm. We begin by handling the case of weights polynomial in the input size, and then show how to modify our methods in order to remove this limitation. In Section 5 we discuss our implementation of the algorithms, and present empirical results for both cases. Finally, in Section 6 we overview some related work in order to present ours in the correct context.

2. Preliminaries

In this section we provide basic definitions, problem definitions, and some prerequisites.

2.1 Definitions

Candidate Set. With a slight change of notation, let $\mathsf{C} = \{c_0, c_1, \ldots, c_m\}$ be a candidate set consisting of the preferred candidate $p = c_0$ and the other m candidates $c_1, \ldots, c_m$. Note that we changed the notation so that the overall number of candidates is $m + 1$; this will help streamline the writing.

Election. An election $E = (\mathsf{C}, V)$ is defined by a candidate set C and a set $N = \{1, \ldots, n\}$ of n voters where each voter submits a *preference order*, i.e., a ranking of the candidates according to his preference. Formally, $V = \langle v_1, \ldots, v_n \rangle$ is the *preference profile*, that is a list of the preference orders v_ℓ for each voter $\ell \in N$. For example, $v_\ell = c_1 \succ p \succ c_2$ is one such possible order if $\mathsf{C} = \{p, c_1, c_2\}$.

Given $E = (\mathsf{C}, V)$, some *decision rule* $\mathcal{R}$ is applied in order to decide on the winner(s); formally $\mathcal{R}(E) \subseteq \mathsf{C}$ is the set of winners of the elections.

Weighted Election. A weighted election $E = (\mathsf{C}, V, \mathbf{w}')$ is defined similarly to an election, with the following twist: a positive-valued weight-vector $\mathbf{w}'$ of dimension n is also included in the input. $\mathbf{w}'$ represents the weights of the n voters, with the meaning that if a voter ℓ has a weight w'_ℓ, then any score awarded to some candidate c as a result of the ballot v_ℓ, is multiplied by w'_ℓ. Some results assume that the weights are encoded by a unary encoding or equivalently that they are representable by a word of length $O(\log \mathcal{N})$ bits in fixed-point arithmetic, where $\mathcal{N}$ is the overall input size. Another equivalent assumption is that they can be normalized to integers that are bounded by a polynomial in the input size. Our most general result will not depend on such an assumption.

Positional Scoring Rules. A scoring rule $\mathcal{R}_{\boldsymbol{\alpha}}$ is usually described by a vector $\boldsymbol{\alpha} = (\alpha_0, \alpha_1, \ldots, \alpha_m)$ for which $\alpha_0 \geq \alpha_1 \geq \cdots \geq \alpha_m$, and α_0 is polynomial in m, used as follows: each voter awards the candidate he ranked i-th the score α_{i-1}. Finally, the winning candidate is the one with the highest aggregated score. In the specific case of the *Borda* scoring rule, $\boldsymbol{\alpha} = (m, m-1, \ldots, 1, 0)$. In *$t$-approval*, $\boldsymbol{\alpha} = (\mathbf{1}^t; \mathbf{0}^{m+1-t})$ where $\mathbf{0}^{t'}$ (resp. $\mathbf{1}^{t'}$) is 0 (resp. 1) concatenated t' times. *Plurality* (resp. *Veto*) is the specific case of 1-approval (resp. m-approval). We refer to an index j representing the score α_j as a *score-type*.

Example 1. As a running example, let $\mathsf{C} = \{p, c_1, c_2, c_3, c_4, c_5\}$ and $n = 2$. Notice that $m = 5$ and thus $|\mathsf{C}| = m+1 = 6$. We define the two voters to have preferences $v_1 = c_5 \succ c_1 \succ c_3 \succ c_2 \succ c_4 \succ p$ and $v_2 = c_4 \succ c_2 \succ c_3 \succ c_5 \succ c_1 \succ p$, respectively. If $\boldsymbol{\alpha} = (5, 4, 3, 2, 1, 0)$ (i.e., Borda), then the scores of $p, c_1, c_2, c_3, c_4, c_5$ following the election are $0, 5, 6, 6, 6, 7$, respectively. We denote this score profile as $\boldsymbol{\sigma}$, and thus $\boldsymbol{\sigma} = (0, 5, 6, 6, 6, 7)$.

WCM and UCM. In the $\mathcal{R}_{\boldsymbol{\alpha}}$-(constructive) weighted coalitional manipulation ($\mathcal{R}_{\boldsymbol{\alpha}}$-WCM) problem, the following input is given:

- A score profile vector $\boldsymbol{\sigma} = (\sigma_0, \sigma_1, \ldots, \sigma_m)$ representing the aggregated scores given hitherto to each candidate in C by the original voters in a weighted election E under the rule $\mathcal{R}_{\boldsymbol{\alpha}}$.
- A weight-vector $\mathbf{w} = (w_1, \ldots, w_k)$ representing the weights of k manipulators to be added to the election.

It should be determined if when adding k manipulators $u_1, \ldots, u_k$ to the election, there exist a voting strategy for the manipulators under $\mathcal{R}_{\boldsymbol{\alpha}}$ in which p wins. If such a strategy exists, the algorithm should find it. $\mathcal{R}_{\boldsymbol{\alpha}}$-(constructive) unweighted coalitional manipulation ($\mathcal{R}_{\boldsymbol{\alpha}}$-UCM) is the specific case where $\mathbf{w}$ is the all-ones vector and therefore can be replaced in the input by the integer k.

We define the input size of the problem as $\mathcal{N} = m + k$. Notice that the input to the problem as we defined it does not include E, but instead the score profile $\boldsymbol{\sigma}$. Indeed the inclusion of $\boldsymbol{\sigma}$ eliminates the need to obtain E in the input: $\boldsymbol{\sigma}$ is a sufficient representation of the outcome of non-manipulator votes, and once the manipulator strategies are computed, we have enough information in order to determine the winner. Such a representation is called a *compilation* of the truthful voters' votes. The complexity of sufficient compilations has already been studied, e.g., by Chevaleyre, Lang, Maudet, and Ravilly-Abadie (2009) and by Xia and Conitzer (2010). If in some scenario, E is preferred in the input (instead of $\boldsymbol{\sigma}$), then the input size becomes $\Theta(nm + k)$. Our choice to have the input size $m + k$ is stricter, as clearly, any algorithm (such as ours) which is polynomial in $\mathcal{N} = m + k$ is also polynomial in $nm + k$.

A careful reader might ask whether the use of $\boldsymbol{\sigma}$ will make the problem more difficult to solve by forcing us to support values of $\boldsymbol{\sigma}$ which are invalid in the sense that they cannot be obtained as a compilation of the non-manipulator votes. However, Davies et al. (2014, see Lemma 1) show that this is not the case. In the context of Borda, for any given non-negative integer vector $\boldsymbol{\sigma} = (\sigma_0, \sigma_1, \ldots, \sigma_m)$, we can define a set of non-manipulators (along with their preferences) and an additional candidate c_{m+1} that will induce an initial score profile $\boldsymbol{\sigma'} = (\tau + \sigma_1, \ldots, \tau + \sigma_m, y)$ for some values τ and $y < \tau$. Such an additive translation and addition of a new candidate will have no effect on the winner (and on the difference between each two candidates' final scores). This is due to the fact that any manipulation strategy on $\boldsymbol{\sigma}$ can be translated into an equivalent manipulation strategy on $\boldsymbol{\sigma'}$ while preserving the differences between candidate scores: for each manipulator, rank c_{m+1} last and keep the order of the other (original) candidates unchanged. As the differences between candidate scores are maintained, and as our results concern an *additive* approximation, we cannot effectively limit the nature of values in $\boldsymbol{\sigma}$.

In our examples we assume the non-unique-winner/co-winner model where p is considered a winner even if she is not the only winner. This assumption is irrelevant to the results themselves as they focus on additively approximating the score margin by an order-of-magnitude expression, where the difference between p being a non-unique-winner to becoming a unique-winner is one point.

Example 2. Continuing our running example, recall that $\boldsymbol{\sigma} = (0, 5, 6, 6, 6, 7)$ and assume we have $k = 2$ manipulators at our disposal. A p-winning strategy exists by assigning them the preferences $u_1 = p \succ c_4 \succ c_2 \succ c_3 \succ c_1 \succ c_5$ and $u_2 = p \succ c_1 \succ c_5 \succ c_3 \succ c_2 \succ c_4$, respectively. At the end of the process, each candidate has a score of 10, and by the co-winner assumption, p wins.

Score Margin. Our objective is to minimize the score margin $\max_{c' \in \mathsf{C} \setminus \{p\}} S(c') - S(p)$, where $S(c)$ is c's score according to the manipulation strategy S. It should be noted that the value $S(c)$ is dependent on $\mathcal{R}_{\boldsymbol{\alpha}}$, $\boldsymbol{\sigma}$, and $\mathbf{w}$, and therefore a more precise notation should be $S(c; \mathcal{R}_{\boldsymbol{\alpha}}, \boldsymbol{\sigma}, \mathbf{w})$. However, since $\mathcal{R}_{\boldsymbol{\alpha}}$, $\boldsymbol{\sigma}$, and $\mathbf{w}$ will always be clear from the context, we discard them for brevity.

Manipulation Matrices. If a p-winning strategy exists, the output is a manipulation strategy S which can be represented as a $k \times (m + 1)$ matrix $\mathbf{S}$ in which the entry $S_{\ell,i} = j$ if manipulator u_ℓ awarded a score of α_j to candidate c_i. Therefore, each row of $\mathbf{S}$ is a

	p	c_1	c_2	c_3	c_4	c_5
$\boldsymbol{\sigma}$	0	5	6	6	6	7
u_1	0 ($\alpha_0 = 5$)	4 ($\alpha_4 = 1$)	2 ($\alpha_2 = 3$)	3 ($\alpha_3 = 2$)	1 ($\alpha_1 = 4$)	5 ($\alpha_5 = 0$)
u_2	0 ($\alpha_0 = 5$)	1 ($\alpha_1 = 4$)	4 ($\alpha_4 = 1$)	3 ($\alpha_3 = 2$)	5 ($\alpha_5 = 0$)	2 ($\alpha_2 = 3$)
S	10	10	10	10	10	10

Table 1: Manipulation matrix for manipulators u_1 and u_2 with strategic ballots $p \succ c_4 \succ c_2 \succ c_3 \succ c_1 \succ c_5$ and $p \succ c_1 \succ c_5 \succ c_3 \succ c_2 \succ c_4$, respectively. The aggregate scores given by the non-manipulators (the vector $\boldsymbol{\sigma}$) are given in the row following the header and the final scores are given in the last row.

	p	c_1	c_2	c_3	c_4	c_5
$\boldsymbol{\sigma}$	0	5	6	6	6	7
–	0	1	2	3	1	2
–	0	4	4	3	5	5
S	10	10	10	10	10	10

Table 2: An example of a relaxed manipulation matrix that can be re-arranged to become the manipulation matrix in Table 1.

permutation of $\{0, \ldots, m\}$ (representing a permutation on $\boldsymbol{\alpha}$). Such a representation is also called a *manipulation matrix*. We can relax the requirement that each row of $\mathbf{S}$ is a permutation, and replace it by the requirement that each score-type j is repeated exactly k times in $\mathbf{S}$. Such a matrix is called a *relaxed manipulation matrix*. We can perform this relaxation as Davies et al. (2014, see Theorem 7) have shown that each relaxed manipulation matrix can be rearranged to become a valid manipulation matrix while preserving each candidate's final score.

Example 3. The (valid) manipulation matrix corresponding to the manipulation strategy in Example 2 is illustrated in Table 1. For clarity, and to prevent confusion between the score-type j and the value α_j it represents, we also indicate the actual value of α_j in parentheses. An example of a relaxed manipulation matrix is presented in Table 2. Notice that every score-type is repeated exactly twice, and that this relaxed manipulation matrix can be rearranged to become the valid manipulation matrix mentioned above, without affecting candidate scores.

Greedy Methods. As a point of reference in our experimental results, we will use both Reverse and Average Fit. In Reverse, the algorithm finds a manipulation strategy as follows: it iterates over the manipulators. Each manipulator awards his maximum score α_0 to p, and then awards all of his other scores to the other candidates according to the reverse order of their *current* aggregate scores. Therefore, the candidate with the current maximum score will be awarded α_m, etc.

Average Fit is defined for UCM only. We first pool all manipulator scores together, so that we have k copies of α_0, k copies of α_1, etc. We then award all the α_0 scores to p. Let $S(p) = \sigma_0 + k\alpha_0$ be p's score following this. We then perform km iterations (i.e., the

	p	c_1	c_2	c_3	c_4	c_5
$\boldsymbol{\sigma}$	0	5	6	6	6	7
u_1	0 ($\alpha_0 = 5$)	1 ($\alpha_1 = 4$)	2 ($\alpha_2 = 3$)	3 ($\alpha_3 = 2$)	4 ($\alpha_4 = 1$)	5 ($\alpha_5 = 0$)
u_2	0 ($\alpha_0 = 5$)	5 ($\alpha_5 = 0$)	4 ($\alpha_4 = 1$)	3 ($\alpha_3 = 2$)	2 ($\alpha_2 = 3$)	1 ($\alpha_1 = 4$)
S	10	9	10	10	10	11

Table 3: The manipulation matrix resulting from applying Reverse to our running example.

number of scores remaining to be awarded) where in each iteration, we visit the candidate c_i with the maximum ratio $d(c_i)/n(c_i)$ where $d(c_i)$ is the gap between $S(p)$ and c_i's current score, and $n(c_i)$ is the number of scores we still need to award to c_i. We award her the maximum score which *fits* in the aforementioned gap.

Example 4. The manipulation matrix resulting from applying Reverse to our running examples can be found in Table 3. As can be easily seen, p loses to c_5 with scores 10 and 11, respectively.

A step-by-step demonstration of an application of Average Fit on our running example can be found in Table 4. The resulting relaxed manipulation matrix can be found in Table 5. Notice two interesting issues: after iteration 5, there is still a copy of α_3 to distribute, however all candidates have gap of at most 1. Therefore at this point it is clear that at least one candidate will have a final score of at least 11. However, at this point the heuristic aggravates the situation even further because it distributes 2 copies of α_4 and 2 copies of α_5 between the four candidates with a ratio of 1/1 (we chose an arbitrary order: c_1, c_3, c_4, c_5), and at this point, it has to give the remaining α_3 to c_2, resulting in her having a final score of 12, compared to p's 10.

High Probability. Throughout the paper, when we use the term 'with a high probability', we are referring to an arbitrarily-chosen polynomially-small failure probability, i.e., a success probability in the form of $1 - m^{-d}$ where $d \geq 1$ is a constant that can be chosen without affecting the asymptotic running time. 'Failure' refers to the event that the algorithm does not provide the desired approximation guarantee.

In this paper we will use various forms of the Hoeffding inequalities, which are variants of the Chernoff inequalities:

Hoeffding's (1963, Theorem 2) Generalized Inequality. Let $X_1, \ldots, X_m$ be independent random variables where each X_i is bounded by the interval $[a_i, b_i]$, respectively. Let $X = \sum_{i=1}^{m} X_i$ and $\bar{X} = X/m$. Then

$$\Pr[\bar{X} - \mathbb{E}[\bar{X}] \geq t] \leq \exp\left(-\frac{2m^2t^2}{\sum_{i=1}^{m}(b_i - a_i)^2}\right) .$$

By redefining the above inequality in terms of the sum X (instead of the mean $\bar{X}$) and defining $\lambda = tm$ we derive the following equivalent formulation:

$$\Pr[X - \mathbb{E}[X] \geq \lambda] \leq \exp\left(-\frac{2\lambda^2}{\sum_{i=1}^{m}(b_i - a_i)^2}\right) .$$

	p	c_1	c_2	c_3	c_4	c_5
$\boldsymbol{\sigma}$	0	5	6	6	6	7
—	$\alpha_0 = 5$					
—	$\alpha_0 = 5$					
1		$\alpha_1 = 4$ (5/2)				
2			$\alpha_1 = 4$ (4/2)			
3				$\alpha_2 = 3$ (4/2)		
4					$\alpha_2 = 3$ (4/2)	
5						$\alpha_3 = 2$ (3/2)
6		$\alpha_4 = 1$ (1/1)				
7				$\alpha_4 = 1$ (1/1)		
8					$\alpha_5 = 0$ (1/1)	
9						$\alpha_5 = 0$ (1/1)
10			$\alpha_3 = 2$ (0/1)			
S	10	10	12	10	9	9

Table 4: A step-by-step demonstration of an application of Average Fit on our running example. The numbers in the left column are the iteration numbers. Each score is followed by a ratio, indicated in parentheses. It represents the value $d(c_i)/n(c_i)$ just before awarding this score.

	p	c_1	c_2	c_3	c_4	c_5
$\boldsymbol{\sigma}$	0	5	6	6	6	7
—	0	1	1	2	2	3
—	0	4	3	4	5	5
S	10	10	12	10	9	9

Table 5: The relaxed manipulation matrix resulting from applying Average Fit to our running example, as depicted in Table 4.

Specifically, when $X_i \in \{0, 1\}$, we obtain the following 'classic' Hoeffding inequality, which is equivalent to Theorem 1 in Hoeffding's (1963) paper:

$$\Pr[X - \mathbb{E}[X] \geq \lambda] \leq \exp\left(-\frac{2\lambda^2}{m}\right) .$$

The same bounds also hold for the symmetric scenarios, i.e., for $\Pr[\mathbb{E}[X] - X \geq \lambda]$ and $\Pr[\mathbb{E}[\bar{X}] - \bar{X} \geq t]$.

2.2 Reduction to a Min-Max Problem

As previously indicated, since we know what the final score of p will be (non-manipulator votes are known and each manipulator will award p the maximum score possible), we can effectively discard p and treat our problem as a minimization problem on the final scores of $c_1, \ldots, c_m$ only. In other words, we can focus on finding the value $T^{\mathsf{OPT}} =$

$\min_S \max_{c \in \mathsf{C} \setminus \{p\}} S(c) = \min_S \max_{i=1,\ldots,m} S(c_i)$, i.e., the minimum possible score (ranging over all possible manipulation strategies S) of the highest-scoring competitor. Following this argument the output strategy S in fact can be represented as a $k \times m$ relaxed manipulation matrix.

3. Unweighted Coalitional Manipulation

In this section we present our algorithm for $\mathcal{R}_{\boldsymbol{\alpha}}$-UCM. We note that the more general algorithm for $\mathcal{R}_{\boldsymbol{\alpha}}$-WCM—that we present in Section 4—obtains a similar approximation factor when applied to an unweighted instance. However, the $\mathcal{R}_{\boldsymbol{\alpha}}$-UCM is simpler, and thus serves as a warm-up for the $\mathcal{R}_{\boldsymbol{\alpha}}$-WCM one, while incorporating most of its fundamental algorithmic ideas and tools. First we start with some motivation for our methods: consider Borda-UCM, for which previous theoretical results were given by Zuckerman et al. (2009), in the form of an additive +1-approximation guarantee on the number of required manipulators. Does it provides an approximation to the score margin as well? We show that such a guarantee can only be $\Omega(m)$. This motivates our algorithm, which obtains a better factor for $k = o(\sqrt{m/\log m})$.

3.1 Lower Bound for REVERSE

We begin by showing that there are cases in which the Reverse algorithm for Borda provides only an additive $\Omega(m)$-approximation to the minimum final score of the highest-scoring competitor. The following claim analyzes Reverse according to this metric.

Claim 1. *There are infinitely many values for m, such that when the addition of more than k extra manipulators is not allowed, the optimal strategy enables p to win by a margin of at least $m/3$, but* Reverse *fails to find a p-winning strategy.*

Proof. We provide an infinite family of cases where the claim holds.

Let $k = 3$ and let $m = 3t$ for any positive integer t. Consider the case where after the non-manipulators voted, all candidates (except p) have the same score $\sigma_i = s$ for all i. Effectively this can be normalized to $(\sigma_1, \ldots, \sigma_m) = \mathbf{0}$.

According to the Reverse algorithm, the first manipulator can award $c_1, \ldots, c_m$ with $0, \ldots, m-1$, respectively, after which the second manipulator will be obliged to award $c_1, \ldots, c_m$ with $m-1, \ldots, 0$, respectively. W.l.o.g. we assume that the third manipulator votes like the first. It can be verified that c_m will end up with the maximal score of $\lceil k/2 \rceil (m-1) = 2(m-1)$.

Conversely, as an upper bound for an optimal solution, consider the following strategy. Place all scores to be given in a descending sequence, that is, the sequence $\langle m-1, m-1, m-1, m-2, m-2, m-2, \ldots, 0, 0, 0 \rangle$. Give the first m scores in the sequence to $c_1, \ldots, c_m$, respectively, the next m to $c_m, \ldots, c_1$, respectively, and the last m to $c_1, \ldots, c_m$, respectively. Since every score-type has 3 copies, we have now described a relaxed manipulation matrix and therefore according to Davies et al. (2014, see Theorem 7 therein) it can be rearranged to become a valid manipulation matrix without changing the final score of each candidate. Now notice that the final score of each candidate is of the form $(m-r)+(m/3+r-1)+(m/3-r) = 5m/3 - r - 1$ for $r = 1, \ldots, m/3$. As such a score can be at most $5m/3 - 2$ (in the specific

case where $r = 1$), the difference between the maximum competitor scores according to the two strategies we described is thus $m/3 = \Omega(m)$. □

3.2 Linear Programming for UCM

We begin by providing a 'natural' way to formulate the min-max version of $\mathcal{R}_{\boldsymbol{\alpha}}$-UCM as an Integer Program (IP), which will be detailed in section 3.2.1. As solving IPs is NP-hard, we can relax it into the equivalent Linear Program (LP). However, such a natural LP is not useful in our setting. In Section 3.2.2 we will provide an intuition as to why this is the case. We note that these two sections can be safely skipped as they are not required for the understanding of our algorithms, but serve as a segue into Section 3.2.3 where we address the shortcomings of the natural LP in the form of a radically different LP formulation, called *Configuration Linear Programming* (C-LP). The number of variables in the C-LP is exponential in the size of the input. Nevertheless, we show that our C-LP can be solved in polynomial time. In the following, we let $[m]$ be a shorthand for $\{1, \ldots, m\}$.

3.2.1 The Natural LP

We define the variables $x_{i,j}$ for $(i, j) \in [m] \times [m]$, and the variable T, with the intent that $x_{i,j}$ will equal the number of times candidate c_i received a score of α_j, and T will serve as the upper-bound on each candidate's final aggregate score. The IP can then be stated as follows:

$$\min_{\mathbf{x}} T$$

subject to:

$$\sum_{i=1}^{m} x_{i,j} = k \qquad \forall j \in [m] \ , \tag{1}$$

$$\sum_{j=1}^{m} x_{i,j} = k \qquad \forall i \in [m] \ , \tag{2}$$

$$\sum_{j=1}^{m} \alpha_j x_{i,j} \leq T - \sigma_i \qquad \forall i \in [m] \ , \tag{3}$$

$$x_{i,j} \in \{0, \ldots, k\} \qquad \forall i \in [m], j \in [m] \ , \tag{4}$$

where (1) guarantees that every score was awarded k times, (2) guarantees that every candidate was given k scores, and (3) guarantees that every candidate gets at most T points.

It should be noted that when treating the problem as a min-max problem, we need to take T as a variable that we wish to minimize (this is done by the objective function). However, if we consider the original definition in which our aim is to make the preferred candidate p win, T can be set to $\sigma_0 + k\alpha_0$ (the final score of p), and the IP will not have an objective function.

3.2.2 Integrality Gap of the Natural LP

While we can relax this IP into an LP by replacing the set in the last constraint to be the continuous interval $[0, k]$, it would not be as helpful. To illustrate this, consider a 'pure' LP rounding algorithm, applied w.l.o.g. to a minimization problem. Such an algorithm works as follows: given an instance of a problem, it solves its associated relaxed LP (the problem's IP formulation where integrality constraints are replaced with their continuous counterparts) and then rounds the resulting solution in some way such that a valid (non-necessarily optimal) solution to the original IP is obtained. The approximation analysis of such algorithms is based on reasoning about how worse the objective value of the rounded solution is compared to the fractional one (i.e., the optimum of the relaxed LP). In other words, what is the increase—or the 'damage done'—to the optimum objective incurred by the rounding process. Since the fractional optimum of a relaxed LP is a lower bound to the integral optimum, i.e., the optimum of the original problem, the same factor also upper-bounds the difference between the objective value of the rounded solution and the objective value of the integral optimum. Thus this process derives an approximation guarantee. We show that in our case, the increase can be $\Omega(m)$, by showing that there are cases in which the difference between the integral and fractional optimum objective values is $\Omega(m)$, and thus an algorithm based solely on the rounding procedure cannot expect an additive $o(m)$-approximation. This kind of reasoning is known as an *integrality gap*, and is demonstrated by the following:

Lemma 2. *For Borda-UCM, an algorithm based solely on rounding the relaxed natural LP cannot obtain an additive $o(m)$-approximation.*

Proof. We show a lower bound on the approximation ratio in the form of an additive integrality gap. In other words, we show an infinite family of instances where the integral solution to the LP (and thus, to the original problem) gives $\Omega(m)$ worse objective value when compared to the fractional solution.

Consider the simple case of m competitors, all having equal initial scores (w.l.o.g. $\sigma_i = 0$ for all $i = 1, \ldots, m$) and a single manipulator. When solving the problem, one candidate will be awarded $m-1$ and thus will have a final score of $m-1$. However, in the fractional solution, the optimum is obtained by splitting each score equally, that is, setting $x_{i,j} = 1/m$ for every $i \in [m]$ and $j \in [m]$. Now every candidate obtained a final score of $1/m \cdot \sum_{j=1}^{m} j = (m-1)/2$. Therefore notice that the gap between the objective values of the integral and fractional solutions is $(m-1)/2 = \Omega(m)$. □

3.2.3 Introducing Configuration LPs

In order to work around the limitations discussed in Section 3.2.2, we will have to resort to a radically different approach, in which variables no longer represent score-types, and instead represent a set of scores—or *configuration*—that can be awarded to a candidate.

Formally, a *configuration* C for some candidate c_i is a vector of dimension m in which C_j represents a number of scores of type j that c_i has received, and for which $\sum_{j=1}^{m} C_j = k$, that is, the overall number of scores awarded is k. For a candidate c_i and a bound T, let $\mathcal{C}_i(T)$ be the set of configurations that do not cause the candidate's overall score to surpass T, i.e., the set of configurations C for which $\sum_{j=1}^{m} C_j \alpha_j \leq T - \sigma_i$.

We formulate the configuration LP as follows:

$$\sum_{C \in \mathcal{C}_i(T)} x_{i,C} \leq 1 \qquad \forall i \in [m] \ , \tag{5}$$

$$\sum_{\substack{i,C \\ C \in \mathcal{C}_i(T)}} C_j x_{i,C} \geq k \qquad \forall j \in [m] \ , \tag{6}$$

$$x_{i,C} \geq 0 \qquad \forall i \in [m], C \in \mathcal{C}_i(T) \ . \tag{7}$$

where we want the $x_{i,C}$ variables to serve as indicator variables indicating whether or not c_i was awarded with configuration C (so that $x_{i,C} \in \{0,1\}$), but due to the need to avoid an IP we relax this constraint to be $x_{i,C} \geq 0$. In addition, eq. (5) guarantees that every candidate was given at most one configuration and eq. (6) guarantees that every score was awarded at least k times. The choice of inequalities over equalities will be explained in Section 3.4.

Example 5. Continuing with our running example, recall that $k = 2$, $m = 5$, $\boldsymbol{\alpha} = (5,4,3,2,1,0)$ (i.e., Borda) and $\boldsymbol{\sigma} = (0,5,6,6,6,7)$. Now assume that $T = 10$. Our choice of T is not arbitrary; 10 is p's final score (that is, $\alpha_0+\alpha_0$), and—as seen in Example 2—is indeed the optimum. Let us focus on the last candidate c_5. $\mathcal{C}_5(T)$ should contain all configurations which award c_5 at most $T - \sigma_5 = 3$ points. These configurations are $(0,0,0,0,2)$ (0 points), $(0,0,0,1,1)$ (1 point), $(0,0,0,2,0)$, $(0,0,1,0,1)$ (2 points), $(0,1,0,0,1)$ and $(0,0,1,1,0)$ (3 points).

When solving the C-LP, only two of her configurations will be given a non-zero value: $x_{5,(0,1,0,0,1)} \approxeq 0.7$ and $x_{5,(0,0,1,1,0)} \approxeq 0.3$. We omit the variables corresponding to the rest of the candidates.

After solving the LP, we will execute additional procedures that will transform the fractional LP solution into a valid solution for the original problem. This procedure can increase the score of some of the candidates, and thus it makes sense to start with the smallest possible T; hopefully, even after the increase the final score will be bounded by $\sigma_0 + k\alpha_0$.

To find the smallest possible T, we perform a one-sided binary search on the value of T. For this purpose, for each possible value of T that we test during the binary search, we redefine the LP, solve the new LP from scratch, and check whether it has a solution. The reason we do not add T as a variable in an objective function (instead of the binary search) is that the number of summands in eqs. (5) and (6) depends on T.

This formulation has the obvious drawback that the number of variables is exponential in k. However, in Section 3.4, following the approach of Bansal and Sviridenko (2006), we will show that if we find a polynomially-computable separation oracle we can solve the LP by referring to the LP dual and using the ellipsoid method. Such an oracle will require a solution to the following seemingly unrelated problem as a subroutine: a variant of the classic Knapsack problem.

3.3 k-Multiset Knapsack

Let $\{1,\ldots,m\}$ be a set of distinct items, where each item j has an associated value v_j and a weight w_j. These items are given as input as well as a weight upper-bound W and a value

lower-bound V. In addition—as opposed to ordinary knapsack—the input also includes an integer k. We are required to find a multiset S of exactly k items (i.e., we can repeat items from the item-set), such that S's overall weight is at most W and S's overall value is greater than V (or return that no such multiset exists). We notice that this problem is very similar to the *exact k-item knapsack* problem mentioned by Caprara, Kellerer, Pferschy, and Pisinger (2000), though in our variant, an item can be repeated in the solution more than once. Another variant, this time with a quadratic objective function (but again without the option to repeat items) is studied by Létocart and Wiegele (2016).

Lemma 3. *The k-multiset knapsack can be solved in time polynomial in k, m, and W (which is pseudo-polynomial due to the dependency on W).*

Proof. We fill out a table $Q[w, \ell]$, for $w = 0, \ldots, W$ and $\ell = 0, \ldots, k$, in which $Q[w, \ell]$ is the highest value obtainable with a size-ℓ multiset of items of an aggregate-weight at most w. Notice that Q can be filled using the following recurrence relation:

$$Q[w, \ell] = \begin{cases} 0 & \text{if } \ell = 0, \\ \max_j v_j + Q'(w - w_j, \ell - 1) & \text{otherwise,} \end{cases} \tag{8}$$

where $Q'(w, \ell) = Q[w, \ell]$ if it is defined, i.e., $w \geq 0$ and $\ell \geq 0$, and otherwise it is $-\infty$.

Q can be filled-out using dynamic programming. Finally, the entry $Q[W, k]$ contains the highest value obtainable with an overall weight of at most W. Therefore, if $Q[W, k] > V$, we have found a required multiset; otherwise such does not exist. The resulting multiset itself can be recovered using backtracking on the table Q. Notice that the amount of work done is $O(Wkm)$. □

3.4 Solving the UCM C-LP

We return to our problem. The choice of inequalities over equalities in the C-LP will be motivated by our use of the LP dual in Theorem 5. However, they have the same effect as equalities, as shown by the following lemma:

Lemma 4. *In a solution to the above C-LP, eqs.* (5) *and* (6) *will actually be equalities.*

Proof. Notice that by eq. (6):

$$km \leq \sum_{j=1}^{m} \sum_{i=1}^{m} \sum_{C \in \mathcal{C}_i(T)} C_j x_{i,C} \tag{9}$$

$$= \sum_{i=1}^{m} \sum_{C \in \mathcal{C}_i(T)} x_{i,C} \sum_{j=1}^{m} C_j \tag{10}$$

$$= k \sum_{i=1}^{m} \sum_{C \in \mathcal{C}_i(T)} x_{i,C} \tag{11}$$

$$\leq km \tag{12}$$

where (12) holds by plugging (5) into (11). We therefore obtain that

$$\sum_{j=1}^{m}\sum_{i=1}^{m}\sum_{C\in\mathcal{C}_i(T)} C_j x_{i,C} = km$$

which forces both non-trivial C-LP inequalities to be equalities. □

Theorem 5. *Given a value T, the UCM C-LP can be solved in polynomial time.*

Proof. In order to solve the C-LP, we will turn to its LP dual; we briefly repeat some LP duality concepts. Please refer to the textbook by Schrijver (1998) for complete definitions and an in-depth discussion.

The dual of a maximization problem is a minimization problem. In order to define it we can treat our primal program as a maximization problem having all coefficients 0 in its objective function. In the dual there is a variable for every constraint of the primal, and a constraint for every variable of the primal. Therefore, we define a variable y_i for each candidate c_i and a variable z_j for each score-type j (since the primal has a constraint for each candidate c_i and for each score-type j). However, since our primal has an exponential number of variables, the dual will have an exponential number of constraints. We will show how to address this.

In short, the non-trivial constraints are then obtained by transposing the constraint-coefficient matrix of the primal, using the primal objective function coefficients as the right-hand side of the dual constraints, and using the right-hand side of the primal constraints as the coefficients of the dual objective function.

The process yields the following dual:

$$\min_{\mathbf{y},\mathbf{z}} \sum_{i=1}^{m} y_i - k\sum_{j=1}^{m} z_j$$

subject to:

$$\begin{aligned}
\sum_{j=1}^{m} C_j z_j &\le y_i && \forall i \in [m], C \in \mathcal{C}_i(T)\\
y_i &\ge 0 && \forall i = 1,\ldots,m\\
z_j &\ge 0 && \forall j = 1,\ldots,m
\end{aligned}$$

As mentioned, the dual has an exponential number of constraints. However, it is solvable; the *ellipsoid method* (Khachiyan, 1980) is a method for solving an LP which iteratively tries to find a point inside the feasible region described by the constraints. However, we do not need to provide all the constraints in advance. Instead, the algorithm can be provided with a subroutine, called a *separation oracle*, to which it calls with a proposed point, and the subroutine then either confirms that the point is inside the feasible region or that it returns a violated constraint (Grötschel et al., 1981). The ellipsoid method algorithm performs a polynomial number of iterations, therefore if the separation oracle runs in polynomial time as well, the LP is solved in overall polynomial time. Notice that the polynomial number of iterations performed by the ellipsoid method implies that the number of constraints that

played a part in finding the optimum (known as *active* constraints) was polynomial as well. In other words, we could effectively discard all except a polynomial number of constraints.

As discussed, a separation oracle for the dual, given a proposed solution $(\mathbf{y}; \mathbf{z})$, needs to find a violated constraint in polynomial time, if one exists. It remains to show that such a separation oracle is polynomial-time computable.

Observe that a violated constraint in this program is a pair (i, C) for which $C \in \mathcal{C}_i(T)$ (and therefore $\sum_{j=1}^{m} C_j \alpha_j \leq T - \sigma_i$) and at the same time $\sum_{j=1}^{m} C_j z_j > y_i$. Fortunately, for a specified i, finding a configuration C that induces a violated constraint can be seen as finding a k-multiset (since $\sum_{j=1}^{m} C_j = k$) given by a solution to our knapsack variant: $[m]$ is the item set (over which j ranges), the value for item j is z_j, while its weight is α_j. The given value lower bound is y_i, and $T - \sigma_i$ is the given upper bound on the weight. Effectively, we use the possibly-tighter weight bound $\min\{k\alpha_1, T - \sigma_i\}$ instead, as $k\alpha_1$ bounds the overall weight obtainable with a size-k multiset. Due to the fact that the weight bound is now polynomial in m and k, the solution to our knapsack variant becomes polynomial.

We repeat this knapsack-solving step for each i until we find a violated constraint, or conclude that no constraint is violated. Once we have solved the dual using the ellipsoid method with the separation oracle, we can discard all variables in the primal that do not correspond to the active constraints of the dual, since the inclusion of those constraints (resp. their corresponding variables) did not have any effect on the dual optimum (resp. the primal optimum).[4] The primal now contains only a polynomial number of variables and can be solved directly using the ellipsoid method or any other known polynomial solvers for LP, such as Karmarkar's (1984) method. □

3.5 Algorithm for UCM

We solve the configuration LP as described in Section 3.4. As mentioned, while both constraints are inequalities, in any solution they will actually be equalities. We perform *randomized rounding* as follows: for each candidate c_i, observe the variables $x_{i,C}$ for $C \in \mathcal{C}_i(T)$. Since $\sum_{C \in \mathcal{C}_i(T)} x_{i,C} = 1$, we treat the $x_{i,C}$'s as a distribution over the configurations for c_i and randomly choose one according to that distribution. For the time being, we give c_i this configuration.[5]

While every candidate now has a valid configuration (and her score does not exceed T), it is possible that the number of scores of a certain type is above or below k. Formally, if candidate c_i received a configuration C^i, let the array H such that $H[j] = \sum_{i=1}^{m} C^i_j$ be the *histogram* of the scores. It is then possible that $H[j] \neq k$. If we would translate the configuration given to each candidate to the list of the scores awarded within it, and write this list as the column of a matrix, this matrix might not be a relaxed manipulation matrix. In order to solve this, we need to replace some of the scores in this matrix with others such that the number of scores of each type will be exactly k. On the other hand, we need to make sure that this process does not add much to the score of each candidate.

4. In other words, the dual of the dual without the discarded constraints is the primal without their corresponding variables. Another way to explain this is that this is exactly the complementary slackness condition of the Karush-Kuhn-Tucker conditions (Karush, 1939; Kuhn & Tucker, 2014), a necessary condition for obtaining the optimum.
5. This is known as *randomized rounding* since choosing an $x_{i,C}$ variable according to the distribution has the effect of rounding it to 1, while rounding all other $x_{i,\hat{C}}$ for $\hat{C} \neq C$ to 0.

Algorithm 1: UCM Approximation algorithm.

Solve the C-LP as described in Section 3.4
foreach i **do** define distribution q s.t. $q(C) = x_{i,C}$ for all $C \in \mathcal{C}_i(T)$ and randomly choose $C^i \sim q$.
$L \leftarrow \langle\rangle$ `/* L is the empty list */`
foreach $i \in [m]$, $j \in [m]$ **do**
  Append C^i_j copies of (i,j) to L `/* j represents the score` α_j `*/`
Sort L in an ascending order by ScoreType$(\cdot)$ `/*` ScoreType$(t) = j$ `if` $t = (i,j)$ `*/`
Re-index L such that $L = \langle t_0, \ldots, t_{km-1} \rangle$
for $r = 0, \ldots, km-1$ **do**
  Observe tuple $t_r = (i,j)$
  `/* Assign the score` $\alpha_{\lfloor r/k \rfloor + 1}$ `to` c_i`, instead of the previous` α_j`: */`
  $C^i_j \leftarrow C^i_j - 1$
  $C^i_{\lfloor r/k \rfloor + 1} \leftarrow C^i_{\lfloor r/k \rfloor + 1} + 1$
return *the relaxed manipulation matrix corresponding to* $C^1, \ldots, C^m$

Let $t = (i,j)$ be a tuple representing the event that candidate c_i received a score of α_j in her configuration. We place all such tuples in a single multiset (if α_j is awarded to c_i more than once, repeat (i,j) as needed). We then sort this multiset according to the j values in a non-decreasing manner (break ties between candidates arbitrarily) thus creating the event-sequence $\langle t_0, \ldots, t_{km-1} \rangle$, i.e., the tuples are now indexed by their rank in this sequence. We now start fixing the scores given as follows; for each tuple $t_r = (i,j)$ having rank r in the sorted sequence, we change the score awarded to c_i (as described by the tuple) from α_j to $\alpha_{\lfloor r/k \rfloor + 1}$. To perform this change in the algorithm, it is enough to set $C^i_j \leftarrow C^i_j - 1$ followed by setting $C^i_{\lfloor r/k \rfloor + 1} \leftarrow C^i_{\lfloor r/k \rfloor + 1} + 1$. This is correct as $C^i_{j'}$, for any j', represents the number of $\alpha_{j'}$ scores awarded to c_i.

Notice that now every score is repeated k times (there are only k possible r values mapping to the same $\lfloor r/k \rfloor$ value). Finally, the corrected configurations represent the final strategy. This can be easily represented as a relaxed configuration matrix by referring to the matrix $[\mathrm{MS}(C^1); \cdots ; \mathrm{MS}(C^m)]$, where $\mathrm{MS}(C^i)$ is a column constructed by taking the configuration C^i, represented as an ordered-multiset of scores (each j is repeated C^i_j times), in some arbitrary order. The entire process is summarized as Algorithm 1.

Let $\beta = d\sqrt{m \ln m}$ for some constant d to be defined later. We also define $g(\boldsymbol{\alpha}) = \max_{j=1,\ldots,m-\beta} \alpha_j - \alpha_{j+\beta}$. In words, $g(\boldsymbol{\alpha})$ is the maximal difference between a score in $\boldsymbol{\alpha}$ and another score β entries away from it.

Lemma 6. *Let $C \in \{C^1, \ldots, C^m\}$ be a configuration obtained for some candidate by the rounding process, and let C' be its corrected version given by the process described above. Then with an arbitrarily-chosen polynomially-small failure probability,*

$$\sum_{j=1}^{m} \alpha_j C'_j \leq \sum_{j=1}^{m} \alpha_j C_j + k \cdot g(\boldsymbol{\alpha}) \ .$$

Proof. Let H be the histogram of the original configurations $C^1, \ldots, C^m$, and let the array G be the array of histogram partial sums, i.e., $G[j] = \sum_{j'=1}^{j-1} H[j']$. In words, $G[j]$ is the number of scores $\alpha_{j'}$ awarded to c_i for which $j' < j$. In a similar manner, define $D^i[j] = \sum_{j'=1}^{j-1} C^i_{j'}$ to be the partial sums array w.r.t. each candidate c_i. We will show that with a polynomially-small failure-probability that depends on d, it holds that $G[j] \geq (j-1)k - dk\sqrt{m \ln m}$.

Fix a specific j. Note that

$$\mathbb{E}[G[j]] = \sum_{j'=1}^{j-1} \mathbb{E}[H[j']] = \sum_{j'=1}^{j-1} \sum_{\substack{i,C \\ C \in \mathcal{C}_i(T)}} C_{j'} x_{i,C} = (j-1)k$$

according to the LP constraints, and that $G[j] = \sum_{i=1}^{m} D^i[j]$, that is, $G[j]$ is a random variable which is the sum of the independent random variables $D^i[j]$ for $i = 1, \ldots, m$. In addition, for every candidate c_i, it holds that $D^i[j] \in [0, k]$, due to the fact that a configuration contains k scores. Therefore, using Hoeffding's (1963) generalized inequality:

$$\begin{aligned} \Pr\left[\mathbb{E}[G[j]] - G[j] \geq \lambda\right] &\leq \exp\left(-\frac{2\lambda^2}{\sum_{i=1}^{m} k^2}\right) \\ &= \exp\left(-\frac{2\lambda^2}{mk^2}\right) . \end{aligned}$$

Setting $\lambda = dk\sqrt{m \ln m}$, for some arbitrary constant $d \geq 2$, we obtain that $\Pr[\mathbb{E}[G[j]] - G[j] \geq dk\sqrt{m \ln m}] \leq 1/m^{2d^2}$, that is, the probability that we will deviate from $\mathbb{E}[G[j]]$ by more than $\widetilde{O}(k\sqrt{m})$ can be made arbitrarily polynomially small. Using the union bound, the same can be made to hold for all $j = 1, \ldots, m$ simultaneously, with a failure probability of at most $m/m^{2d^2} = 1/m^{2d^2-1}$.

Now observe a tuple $t_r = (i, j)$ before being possibly corrected by the algorithm. The number of tuples appearing before t_r in the sorted sequence must be greater than the number of scores awarded with a score-type that is strictly less than j (as all such scores appear in tuples $t_{r'}$ where $r' < r$), which is by definition $G[j]$. Therefore, it holds that $r > G[j] \geq (j-1)k - dk\sqrt{m \ln m}$, where the second inequality holds with a high probability. Therefore by the algorithm changing the score α_j to $\alpha_{\lfloor r/k \rfloor + 1}$, the score increases by at most $\alpha_{\lfloor r/k \rfloor + 1} - \alpha_j \leq \alpha_{(j-1) - d\sqrt{m \ln m} + 1} - \alpha_j \leq g(\boldsymbol{\alpha})$.

Now observe some candidate c_i with a given configuration C^i corrected by the algorithm to become a configuration C'. Since in the worst case, all of c_i's k scores were affected as such, her overall score has increased by at most $kg(\boldsymbol{\alpha})$. □

Corollary 7. *The above algorithm provides an additive $kg(\boldsymbol{\alpha})$-approximation with a high probability. By repeating the randomized rounding procedure a linear number of times, the failure probability becomes exponentially-small. The overall running time is polynomial.*

Proof. Let T^{OPT} be the optimal value for the original problem, and let $T^\star$ be the best bound obtainable via the above C-LP combined with the binary search on T. Notice that $T^\star \leq T^{\mathsf{OPT}}$, as the optimal solution is also a valid solution for the C-LP. Now observe the highest-scoring candidate in the C-LP. When the algorithm terminates, with a high-probability her score will be at most $T^\star + kg(\boldsymbol{\alpha}) \leq T^{\mathsf{OPT}} + kg(\boldsymbol{\alpha})$.

As the increase in the score of any other candidate resulting from the algorithm is also bounded by $kg(\boldsymbol{\alpha})$, the bound $T^{\mathsf{OPT}} + kg(\boldsymbol{\alpha})$ holds for all candidates. We conclude that this is indeed an additive $kg(\boldsymbol{\alpha})$-approximation.

As discussed, solving the C-LP is done in polynomial time (by the polynomial number of iterations of the ellipsoid method and the polynomial runtime of the k-multiset knapsack separation oracle). The rounding and fixing procedures are dominated by going over a polynomial number of non-zero variables of the C-LP and are therefore polynomial as well. If we repeat the randomized rounding procedure a linear number of times and pick the iteration yielding the minimum addition to $T^{\star}$, the probability of not obtaining a $kg(\boldsymbol{\alpha})$-approximation becomes exponentially-small while the runtime remains polynomial. □

The above corollary directly yields the following theorem:

Theorem 8. *There exists a randomized Monte Carlo algorithm for $\mathcal{R}_{\boldsymbol{\alpha}}$-UCM which provides an additive $kg(\boldsymbol{\alpha})$-approximation to T^{OPT} with an exponentially-small failure probability.*

Proof. By Corollary 7. □

At this point we directly obtain the next corollary, focusing on the specific case of Borda:

Corollary 9. *There exists a randomized Monte Carlo algorithm for Borda-UCM which provides an additive $O(k\sqrt{m \log m})$-approximation to T^{OPT} with an exponentially-small failure probability.*

Proof. By noticing that for Borda, $g(\boldsymbol{\alpha}) = O(\sqrt{m \log m})$. □

4. Weighted Coalitional Manipulation

In this section we show how to generalize the previous algorithm to support the weighted setting. Specifically, supporting weighted manipulators introduces the notion that the manipulators are no longer identical, and thus a new definition of a *configuration* is needed: one that also maintains their identity. In turn, this also requires us to define and employ a different knapsack variant, namely that of the k-sequence knapsack. In the following, we explain the new algorithm in a similar fashion to the previous one, focusing on the required changes. First we begin with the case of polynomial weights, that is we assume that the weight w_ℓ for $\ell = 1, \ldots, k$ is an integer bounded by a polynomial in the input size $\mathcal{N}$ (or any of the equivalent conditions mentioned in Section 2.1); later, we will generalize our results for any weight vector.

4.1 Linear Programming for WCM

When turning to the WCM problem, the 'natural' LP still suffers from the deficiencies described in Section 3.2. We again resort to using configurations. As mentioned, configurations will now be defined in a different manner, since now, when each voter has an associated weight, voters are no longer identical and consequently our configurations need to capture the identity of the manipulators.

A configuration C for some candidate c_i is now defined as a length-k sequence in which $C_\ell = j$ if the manipulator u_ℓ awarded α_j to c_i. For a candidate c_i and a bound T, $\mathcal{C}_i(T)$ is again the set of configurations that do not cause the candidate's overall score to surpass T, which this time is formally $\sum_{\ell=1}^{k} w_\ell \alpha_{C_\ell} \leq T - \sigma_i$.

The configuration LP is now formulated as follows:

$$\sum_{C \in \mathcal{C}_i(T)} x_{i,C} \leq 1 \qquad \forall i \in [m] \ , \tag{13}$$

$$\sum_{\substack{i, C \in \mathcal{C}_i(T) \\ C_\ell = j}} x_{i,C} \geq 1 \qquad \forall j \in [m], \forall \ell \in [k] \ , \tag{14}$$

$$x_{i,C} \geq 0 \qquad \forall i \in [m], C \in \mathcal{C}_i(T) \ . \tag{15}$$

Again, we want the $x_{i,C}$'s to serve as indicator variables indicating whether or not c_i was awarded with configuration C (but again, are required to relax their corresponding integrality constraints). In addition, eq. (13) guarantees that every candidate was given at most one configuration and eq. (14) guarantees that every score was awarded by every manipulator at least once. The choice of inequalities over equalities will be explained in Section 4.3.

We present another—more complex—Knapsack variant, which will be used later by the separation oracle required for solving the C-LP.

4.2 k-Sequence Knapsack

Let $\{1, \ldots, m\}$ be a set of distinct items. In the *k-sequence knapsack* problem we are required to construct a length-k sequence $S = s_1, \ldots, s_k$ of items; we can repeat items from the item-set, however, we are subject to some additional constraints as will be specified immediately. The input in this problem is the following:

- A value $v(j, \ell)$, for every $j \in [m], \ell \in [k]$ where $v(j, \ell)$ is the value obtained by placing item j at location ℓ in the sequence.
- A cost $b(j)$ for each item j, and a penalty p_ℓ for each location $\ell \in [k]$. Placing an item j at location ℓ in the sequence has a penalized cost $p_\ell b(j)$, i.e., it depends on both the item's weight and the penalty for location ℓ, but on each independently.
- A value lower-bound V.
- A penalized cost upper-bound B.

The resulting sequence S should adhere to the following constraints:

- S's overall value $\sum_{\ell=1}^{k} v(s_\ell, \ell)$ is greater than V.
- S's overall penalized cost $\sum_{\ell=1}^{k} p_\ell b(s_\ell)$ is at most B.

If such sequence S exists, we should return it; otherwise we return that no such sequence exists.

Lemma 10. *The k-sequence knapsack can be solved in time polynomial in k, m, and B (which is pseudo-polynomial due to the dependency on B).*

Proof. Similar to the proof of Lemma 3, we fill out a table $Q[b', \ell]$, for $b' = 0, \ldots, B$ and $\ell = 0, \ldots, k$, in which $Q[b', \ell]$ is the highest value obtainable with a length-ℓ sequence of items of a penalized cost of at most b'. This time Q is filled using a different recurrence relation:

$$Q[b', \ell] = \begin{cases} 0 & \text{if } \ell = 0, \\ \max_j v(j, \ell) + Q'(b' - p_\ell b(j), \ell - 1) & \text{otherwise,} \end{cases} \tag{16}$$

where $Q'(b', \ell) = Q[b', \ell]$ if it is defined, i.e., $b' \geq 0$ and $\ell \geq 0$, and otherwise it is $-\infty$.

Q can be filled-out using dynamic programming. Finally, the entry $Q[B, k]$ contains the highest value obtainable with an overall penalized cost of at most B. If $Q[B, k] > V$, we have found a required sequence; otherwise such does not exist. The resulting sequence itself can be recovered using backtracking on the table Q. Notice that the amount of work done is $O(Bkm)$. □

4.3 Solving the WCM C-LP

We return to our problem. Again, the choice of inequalities over equalities in the C-LP will be motivated by our use of the LP dual in Theorem 12. However, they have the same effect as equalities, as shown by the following lemma:

Lemma 11. *In a solution to the above C-LP, eqs.* (13) *and* (14) *will actually be equalities.*

Proof. Note that by eq. (14):

$$km \leq \sum_{j,\ell} \sum_{i=1}^{m} \sum_{\substack{C \in \mathcal{C}_i(T) \\ C_\ell = j}} x_{i,C} \tag{17}$$

$$= \sum_{i=1}^{m} \sum_{C \in \mathcal{C}_i(T)} \sum_{\substack{\ell, j \\ C_\ell = j}} x_{i,C} \tag{18}$$

$$= \sum_{i=1}^{m} \sum_{C \in \mathcal{C}_i(T)} x_{i,C} \sum_{\substack{\ell, j \\ C_\ell = j}} 1 \tag{19}$$

$$= k \sum_{i=1}^{m} \sum_{C \in \mathcal{C}_i(T)} x_{i,C} \tag{20}$$

$$\leq km \tag{21}$$

where (21) holds by plugging (13) into (20). We therefore obtain that

$$\sum_{j,\ell} \sum_{i=1}^{m} \sum_{\substack{C \in \mathcal{C}_i(T) \\ C_\ell = j}} x_{i,C} = km$$

which forces both above non-trivial LP inequalities to be equalities. □

Theorem 12. *Given a value T, the WCM C-LP can be solved in polynomial time.*

Proof. We again turn to the LP dual, which this time is:

$$\min_{\mathbf{y},\mathbf{z}} \sum_{i=1}^{m} y_i - \sum_{j,\ell} z_{j,\ell}$$

subject to:

$$\sum_{\substack{j,\ell \\ C_\ell = j}} z_{j,\ell} \leq y_i \qquad \forall i \in [m], C \in \mathcal{C}_i(T)$$

$$y_i \geq 0 \qquad \forall i = 1, \ldots, m$$

$$z_{j,\ell} \geq 0 \qquad \forall (j,\ell) \in [m] \times [k]$$

Furthermore, the above single non-trivial constraint can be more conveniently re-written as

$$\sum_{\ell=1}^{k} z_{C_\ell,\ell} \leq y_i \qquad \forall i \in [m], C \in \mathcal{C}_i(T) \ .$$

We again turn to the ellipsoid method with a separation oracle; this time, a violated constraint w.r.t. this program is a pair (i, C) for which $C \in \mathcal{C}_i(T)$ (and therefore $\sum_{\ell=1}^{k} w_\ell \alpha_{C_\ell} \leq T - \sigma_i$) and at the same time $\sum_{\ell=1}^{k} z_{C_\ell,\ell} > y_i$. For a specified i, finding a configuration C that induces a violated constraint can be seen as finding a k-sequence given by a solution to our knapsack variant: $[m]$ is the item set (over which j ranges), the value for placing item j at location ℓ is $z_{j,\ell}$, item j's cost is α_j, and the penalty for location ℓ is w_ℓ. The given value lower bound is y_i, and $T - \sigma_i$ is the given upper bound on the penalized cost. Effectively, we use the possibly-tighter penalized cost bound $\min\{W\alpha_1, T - \sigma_i\}$ instead, as $W\alpha_1$ bounds the overall penalized cost obtainable with a length-k sequence. As now the weight bound is polynomial in α_1 and W—and by our assumption that the weights (and thus W) are polynomial in the input size $\mathcal{N}$—the solution to our knapsack variant becomes polynomial. (As mentioned, we will later address the compromise needed to be made when W is not polynomial in the input size.)

We repeat this knapsack-solving step for each i until we find a violated constraint, or conclude that no constraint is violated. Once we have solved the dual using the ellipsoid method with the separation oracle, we continue in a similar fashion to the proof of Theorem 5. □

4.4 Algorithm for WCM

We solve the configuration LP as described in Section 4.3. As mentioned, both constraints will actually be equalities. For each candidate c_i, since $\sum_{C \in \mathcal{C}_i(T)} x_{i,C} = 1$, we treat the $x_{i,C}$'s as a distribution over the configurations for c_i and randomly choose one according to that distribution. For the time being, we give c_i this configuration.

As for UCM, every candidate now has a valid configuration but constraints may still be violated; it is possible that the number of scores of a certain type j given by a specific manipulator u_ℓ is not exactly 1. Formally, fix a specific manipulator u_ℓ; we let the array H such that $H[j] = |\{i \mid C_\ell^i = j\}|$ be the histogram of the scores with respect to u_ℓ. It is then possible that $H[j] \neq 1$. Our goal, as before, is to fix this without introducing too much of

Algorithm 2: WCM Approximation algorithm.

```
1 Solve the C-LP as described in Section 4.1
2 foreach i do define distribution q s.t. q(C) = x_{i,C} for all C ∈ C_i(T) and randomly
    choose C^i ∼ q.
3 for ℓ ← 1 to k do
4 |   Let A = {(i, ℓ, C^i_ℓ) | i = 1, ..., m}      /* tuple (i, ℓ, j) represents the event
  |    that u_ℓ awarded α_j to c_i */
5 |   Sort A in an ascending order by ScoreType(·) and let L = ⟨t_0, ..., t_{m−1}⟩ be the
  |    resulting list.                              /* ScoreType(t) = j if t = (i, ℓ, j) */
6 |   for r ← 0 to m − 1 do
7 |   |   Observe tuple t_r = (i, ℓ, j).
8 |   |   C^i_ℓ ← r + 1      /* this assigns the score α_{r+1} to c_i, instead of the
  |   |    previous α_j */
9 return the resulting manipulation matrix [C^1; ··· ; C^m]  /* place C^i as the i-th
    column vector of the resulting matrix */
```

an addition to the candidates' overall scores. However, there is now some added complexity due to the necessity to preserve the identity of the voter when fixing a specific score that is awarded by him.

Let $t = (i, \ell, j)$ be a tuple representing the event that candidate c_i received a score of α_j from manipulator u_ℓ in her configuration. Fix a manipulator u_ℓ, and place all tuples having u_ℓ as their respective manipulator in a set A, that is $A = \{(i, \ell, C^i_\ell) \mid i = 1, \ldots, m\}$. Sort A according to each tuple (i, ℓ, C^i_ℓ)'s score-type C^i_ℓ, and let $L = \langle t_0, \ldots, t_{m-1} \rangle$ be the resulting list. Notice that any tuple $t_r = (i, \ell, j)$ in L represents the event that currently $C^i_\ell = j$. Now change C^i_ℓ by setting $C^i_\ell \leftarrow r + 1$. In words, C^i_ℓ gets the rank of its respective tuple in the sorted list L, plus one. In effect, the score awarded to c_i by u_ℓ changes from α_j to α_{r+1}.

We repeat the above process for each manipulator. Notice that now every score is repeated k times, one by each manipulator. The entire process is summarized as Algorithm 2.

In a similar fashion to the UCM algorithm, we let $\beta = d\sqrt{m \ln \mathcal{N}}$ for some constant d to be defined later, and let $g(\boldsymbol{\alpha}) = \max_{j=1,\ldots,m-\beta} \alpha_j - \alpha_{j+\beta}$. Notice that there is a slight change to the logarithm appearing in β.

Lemma 13. *Let $C \in \{C^1, \ldots, C^m\}$ be a configuration obtained for some candidate by the rounding process, and let C' be its corrected version given by the process described above. Then with an arbitrarily-chosen polynomially-small failure probability,*

$$\sum_{\ell=1}^{k} w_\ell \alpha_{C'_\ell} \leq \sum_{\ell=1}^{k} w_\ell \alpha_{C_\ell} + W \cdot g(\boldsymbol{\alpha}) \ .$$

Proof. Fix a specific ℓ. Let H be the histogram with respect to manipulator u_ℓ over the original configurations $C^1, \ldots, C^m$, so that $H[j] = |\{i \mid C^i_\ell = j\}|$. Let the array G be the array of histogram partial sums, i.e., $G[j] = \sum_{j'=1}^{j-1} H[j'] = |\{i \mid C^i_\ell < j\}|$. We also define $D^i[j]$ to be a Bernoulli variable which is equal to 1 if $C^i_\ell < j$, and 0 otherwise. We will

show that with a polynomially-small failure-probability which depends on d, we have that $G[j] \geq (j-1) - d\sqrt{m \ln \mathcal{N}}$.

Now also fix a specific j. Note that

$$\mathbb{E}[G[j]] = \sum_{j'=1}^{j-1} \mathbb{E}[H[j']] = \sum_{j'=1}^{j-1} \sum_{\substack{i,C \\ C \in \mathcal{C}_i(T) \\ C_\ell = j'}} x_{i,C} = j-1$$

according to the LP constraints, and that $G[j] = \sum_{i=1}^{m} D^i[j]$, that is, $G[j]$ is a random variable which is the sum of the independent random variables $D^i[j] \in \{0,1\}$ for $i = 1, \ldots, m$. Therefore, using the 'classic' Hoeffding inequality:

$$\Pr\left[\mathbb{E}[G[j]] - G[j] \geq \lambda\right] \leq \exp\left(-\frac{2\lambda^2}{m}\right) .$$

Setting $\lambda = d\sqrt{m \ln \mathcal{N}}$, for some arbitrary constant $d \geq 2$, we have that $\Pr[\mathbb{E}[G[j]] - G[j] \geq d\sqrt{m \ln \mathcal{N}}] \leq 1/\mathcal{N}^{2d^2}$, that is, the probability that we will deviate from $\mathbb{E}[G[j]]$ by more than $\widetilde{O}(\sqrt{m})$ can be made arbitrarily polynomially small. Using the union bound, the same can be made to hold for all $j = 1, \ldots, m$ and $\ell = 1, \ldots, k$ simultaneously, with a failure probability of at most $km/\mathcal{N}^{2d^2} \leq 1/\mathcal{N}^{2d^2-2}$.

Now observe a tuple $t_r = (i, \ell, j)$ before being possibly corrected by the algorithm. Since (as a result of the sorting) its rank r in the sorted sequence must be strictly greater than the number of scores $\alpha_{j'}$ given by u_ℓ for which $j' < j$—which is by definition $G[j]$—we obtain that $r > G[j] \geq (j-1) - d\sqrt{m \ln \mathcal{N}}$, where the second inequality holds with a high probability. Therefore by the algorithm changing the score α_j to α_{r+1}, the score increases by at most $\alpha_{r+1} - \alpha_j \leq \alpha_{(j-1)-d\sqrt{m \ln \mathcal{N}}+1} - \alpha_j \leq g(\boldsymbol{\alpha})$.

Now observe some candidate c_i with a given configuration C^i corrected by the algorithm to become configuration C'. Since in the worst case, all of c_i's k scores were affected as such, her overall score has increased by at most $\sum_{\ell=1}^{k}(w_\ell g(\boldsymbol{\alpha})) = W g(\boldsymbol{\alpha})$. □

Corollary 14. *For integer weights bounded by a polynomial in $\mathcal{N}$, the above algorithm provides an additive $W g(\boldsymbol{\alpha})$-approximation with a high probability. By repeating the randomized rounding procedure a linear number of times, the failure probability becomes exponentially-small. The overall running time is polynomial.*

Proof. Identical to the proof of Corollary 7, where the only difference is that W is used instead of k. □

This directly yields the main theorem of this paper:

Theorem 15. *For integer weights bounded by a polynomial in $\mathcal{N}$, there exists a randomized Monte Carlo algorithm for $\mathcal{R}_{\boldsymbol{\alpha}}$-WCM which provides an additive $W g(\boldsymbol{\alpha})$-approximation to T^{OPT} with an exponentially-small failure probability, where W is the sum of voter weights.*

Proof. By Corollary 14. □

The above theorem immediately paves the way to the following corollary, focusing on Borda:

Corollary 16. *For weights bounded by a polynomial in $\mathcal{N}$, there exists a randomized Monte Carlo algorithm for Borda-WCM which provides an additive $O(W\sqrt{m\log\mathcal{N}})$-approximation to T^{OPT} with an exponentially-small failure probability.*

Proof. By noticing that for Borda, $g(\boldsymbol{\alpha}) = O(\sqrt{m\log\mathcal{N}})$. □

4.5 Supporting General Weights

While we cannot achieve the same additive approximation factor when the weights in $\mathbf{w}$ (and thus W) are not guaranteed to be polynomial in the input size $\mathcal{N}$, we can still obtain a very similar result: instead of yielding a solution with a bound $T \leq T^{\mathsf{OPT}} + Wg(\boldsymbol{\alpha})$, our solution will provide a bound $T \leq (1+\epsilon)T^{\mathsf{OPT}} + Wg(\boldsymbol{\alpha})$ for any constant $\epsilon > 0$. Specifically for Borda-WCM, we will show that under reasonable assumptions it holds that $Wg(\boldsymbol{\alpha}) = o(T^{\mathsf{OPT}})$ and thus in this case our solution will yield an FPTAS.

The above can be achieved by some relatively simple changes to the WCM algorithm detailed hitherto. Let us first focus on the problematic aspect of the current algorithm w.r.t. general weight vectors: the bound $B = \min\{W\alpha_1, T - \sigma_i\} \leq W\alpha_1$ is used as the penalized cost bound when we run the k-sequence knapsack algorithm (and is therefore a factor in its runtime), but W is no longer guaranteed to be polynomial in $\mathcal{N}$. We solve this issue in the proof of the following theorem.

Theorem 17. *There exists a randomized Monte Carlo algorithm for $\mathcal{R}_{\boldsymbol{\alpha}}$-WCM which finds a strategy that will obtain a bound $T \leq (1+\epsilon)T^{\mathsf{OPT}} + Wg(\boldsymbol{\alpha})$, for any constant $\epsilon > 0$, with an exponentially-small failure probability, where W is the sum of voter weights.*

Proof. We shall use the well-known scaling trick that is the heart of many FPTAS algorithms. However, it will not be performed on the weights, but rather on the penalized costs. Recall that while searching for the C-LP optimum $T^\star$, we try different T values in the course of a one-sided binary search. Each time we redefine the C-LP and solve it. Focus on some trial with a value T, and observe the penalized cost $p_\ell b(j)$ for $\ell = 1,\ldots,k$ and $j = 1,\ldots,m$. Replace each such $p_\ell b(j)$ with $\overline{p_\ell b(j)}$, where $\overline{p_\ell b(j)}$ is the value $p_\ell b(j)$ rounded *down* to the nearest multiple of $\epsilon T/k$. Formally:

$$\overline{p_\ell b(j)} = \left\lfloor \frac{p_\ell b(j)}{\epsilon T/k} \right\rfloor \epsilon T/k \ .$$

As a result, notice that $\overline{p_\ell b(j)} \leq p_\ell b(j) \leq \overline{p_\ell b(j)} + \epsilon T/k$. In addition, in the k-sequence knapsack algorithm, we fill only the entries of the table Q referring to penalized cost values that are multiples of $\epsilon T/k$. That is, we only consider entries which are of the form $Q[b', \ell]$ for $\ell = 1,\ldots,k$ and $b' = 0, \epsilon T/k, 2\epsilon T/k, 3\epsilon T/k, \ldots, \overline{B}$, where—as before—$\overline{B}$ is $B = \min\{W\alpha_1, T - \sigma_i\}$ rounded down to the nearest multiple of $\epsilon T/k$. Notice that the numbers of cells to fill is at most $k \cdot B/(\epsilon T/k) = O(k^2\epsilon^{-1})$ as $B \leq T$, and that there is no need to maintain the cells we do not fill in memory.

Let C be a configuration that appears in the result of the C-LP (i.e., it is a valid configuration for which there is a variable $x_{i,C} > 0$, and furthermore, this configuration was the result of some invocation of k-sequence knapsack). Since the penalized costs were scaled down, the fact a configuration C was deemed as valid only implies that $\sum_{\ell=1}^{k} \overline{w_\ell \alpha_{C_\ell}} \leq T - \sigma_i$

and thus $\sum_{\ell=1}^{k} w_\ell \alpha_{C_\ell} \leq \sum_{\ell=1}^{k} (\overline{w_\ell \alpha_{C_\ell}} + \epsilon T/k) \leq (1+\epsilon)T - \sigma_i$. In particular, this holds for $T^\star$: a valid configuration used in the iteration when $T = T^\star$ is always in $\mathcal{C}_i((1+\epsilon)T^\star)$ (but not necessarily in $\mathcal{C}_i(T^\star)$ anymore). When also accounting for the increases to $T^\star$ resulting from the rounding phase and the fixing phase (Lemma 13), we arrive at a solution $T \leq (1+\epsilon)T^\star + Wg(\boldsymbol{\alpha})$. □

Brelsford et al. (2008) showed that if m is bounded, $\mathcal{R}_{\boldsymbol{\alpha}}$-WCM has an FPTAS w.r.t. the score margin. For the specific case of Borda-WCM, our algorithm can be considered an FPTAS as well – avoiding any limitation on m, but under the reasonable assumption that if m is unbounded, then $k = 2^{o(m)}$.

Corollary 18. *If either m is bounded, or $k = 2^{o(m)}$, then there exists a randomized Monte Carlo algorithm for Borda-WCM which will find a strategy that will obtain a bound $T \leq (1+\epsilon)T^{\mathsf{OPT}}$, for any constant $\epsilon > 0$, with an exponentially-small failure probability.*

Proof. If m is bounded, Borda-WCM has an FPTAS using Lemma 3 in Brelsford et al.'s (2008) result. We are left with the case where m is an unbounded parameter and $k = 2^{o(m)}$.

In this scenario, for the specific case of Borda-WCM, $g(\boldsymbol{\alpha}) = d\sqrt{m \ln \mathcal{N}}$ for some constant d, and thus $g(\boldsymbol{\alpha}) = o(m)$. We will show that $Wg(\boldsymbol{\alpha}) = o(T^{\mathsf{OPT}})$ w.r.t. the unbounded parameter m. To see that, notice that for Borda-WCM the overall 'voting mass' given by the manipulators is $\Omega(Wm^2)$, and thus the highest-scoring candidate has a score of at least $T^{\mathsf{OPT}} = \Omega(Wm)$. On the other hand, the additive factor $dW\sqrt{m \ln \mathcal{N}} = W \cdot o(m)$ by the argument above. Therefore, $dW\sqrt{m \ln \mathcal{N}}$ is a lower order term compared to T^{OPT}. As such, for a large enough m, we have that $dW\sqrt{m \ln \mathcal{N}} \leq \epsilon T^{\mathsf{OPT}}$ and thus, the approximation guarantee becomes $T \leq (1+\epsilon)T^\star + Wg(\boldsymbol{\alpha}) \leq (1+2\epsilon)T^{\mathsf{OPT}}$. Recalibrating the algorithm to use $\epsilon/2$ instead of ϵ will yield the desired result. □

5. Empirical Analysis

While we believe that the main message of this work is the research of $\mathcal{R}_{\boldsymbol{\alpha}}$-UCM and $\mathcal{R}_{\boldsymbol{\alpha}}$-WCM from a theoretical perspective, i.e., establishing positive results regarding the approximability of $\mathcal{R}_{\boldsymbol{\alpha}}$-UCM and $\mathcal{R}_{\boldsymbol{\alpha}}$-WCM, we also wanted to check how they compare with the known methods, with both being of a greedy type.

5.1 Implementation

We implemented our algorithm for the case of $\mathcal{R}_{\boldsymbol{\alpha}}$-UCM and $\mathcal{R}_{\boldsymbol{\alpha}}$-WCM and uploaded the code to a public repository.[6] The main subroutine in our implementation is tasked with solving the LP dual that we defined. However, the dependency on an LP solver using the ellipsoid method with a separation oracle proved difficult as to the best of our knowledge there is no library which supports solving an LP this way. Instead we simulated this by using general LP-solving libraries (Andersen, Dahl, & Vandenberghe, 2016; Makhorin, 2017) and running the separation oracle externally as described in Algorithm 3.

6. `https://github.com/okeller/BordaManipulation`

Algorithm 3: Simulating the ellipsoid method with a separation oracle.

Input: A linear program $P = (f, S)$ with an objective function f and a separation oracle S (instead of an explicit list of constraints)

```
1 Let R ← ∅                          /* R will be a list of active constraints */
2 Let P' ← (f, R)                    /* P' is P without any constraints */
3 x ← LP-solve(P')
4 while S(x) returns a violated constraint r do
5 |   R ← R ∪ {r}
6 |   P' ← (f, R)
7 |   x ← LP-solve(P')
8 return x
```

5.1.1 Practical Improvements

We used the following practical improvements in our implementation:

- We obtained a running-time speedup by modifying the separation oracle to return a set of violated constraints, one for each i (if such exists), instead of a single violated constraint, and adding all of them to the list of constraints.
- When sorting by the score-type after the rounding phase, we broke ties between tuples having the same score-type j in favor of the candidate with the lower current aggregate score. This way, a candidate with a higher current aggregate score would be less likely to have this specific score increased by the fixing procedure.

5.2 Experiments

To evaluate our algorithms for both UCM and WCM in a well-studied setting, we ran experiments for Borda-UCM and Borda-WCM on sets of values for n, k and m (the choices of values will be explained shortly). For the unweighted case, our algorithm was compared to Average Fit that was shown to empirically outperform Reverse (Davies et al., 2014). For the weighted case, as we are not aware of a generalization of Average Fit to a weighted setting, we compared against Reverse. In both cases, and in order to be able to compare performance across different values of k and m, we plotted the ratio $T/T^\star$, where T is the value obtained by the algorithm (either ours, or its competitor, depending on the case), and $T^\star$ is the bound obtained by the fractional C-LP in conjunction with the one-sided binary search. In other words, $T^\star$ is used as a baseline, as it is polynomially computable and serves as a lower bound to the optimal solution, thus the ratio $T/T^\star$ is an upper bound to the multiplicative approximation of the respective method.

It is worth noting that runtime-wise our algorithm is inferior to both greedy heuristics: as the greedy heuristics are extremely simple combinatorial algorithms, they run very quickly, while ours is based on solving an LP. However, the main issue that affected our *practical* running time is *not* the time required for solving an LP per se, but the fact that in our implementation we had to solve an LP multiple times each time our algorithm solves a single LP instance. As mentioned, this is due to the lack of an LP solver with a separation oracle, forcing us to simulate it. As depicted in Algorithm 3, this simulation incurs many

repetitions on the subprocedure of solving an LP. As a result, we were limited in the number of repeated experiments for each set of n, k, m values. We conducted 100 for the UCM experiments and 90 for the WCM experiments. Naturally, future existence of a separation oracle LP solver would remove this limitation. We chose $n = 2k$, because having one manipulator for every two truthful voters represents a sweet-spot where manipulators have enough power to change outcomes, but not too much.

5.2.1 Borda-UCM

We compared our results to those obtained by Average Fit, and to the fractional solution, $T^\star$. We performed two sets of experiments, where we drew the non-manipulator votes from either a uniform distribution, or from a Pólya-Eggenberger urn model, similar to the experiments performed by Davies et al. (2014). This urn model can be described as follows. At first, we hold all possible $m!$ permutations of order m in an urn. Then we iterate over the voters where each voter picks a permutation from the urn as his ballot, and then returns it, alongside $b = m!$ copies of it, to the urn.

In our experiments, we chose $k \approx \sqrt{m}$ or smaller, because our algorithms are suitable for low k values. As noted, our algorithm for Borda-UCM is theoretically competitive when $k = o(\sqrt{m/\log m})$, and we wanted to verify this also in an empirical setting. Lower k values are also the cases which are more difficult for the Average Fit heuristic of Davies et al. (2014). The results are depicted in Figure 1 (uniform model) and Figure 2 (urn model).

As can be seen, our algorithm performs comparably to Average Fit. In the uniform model our algorithm seems to have a slight advantage for low k values, while the situation is opposite for higher k values. In the urn model, both performances seem to be very similar.

For a good example in which our algorithm outperforms Average Fit, consider our running example. Recall that $k = 2$, $m = 5$ and $(\sigma_0, \ldots, \sigma_5) = (0, 5, 6, 6, 6, 7)$. Obviously both methods will award p with $5 + 5 = 10$. However, in our algorithm the top score of a candidate who is not p will be 10, and in Average Fit (as seen before in Table 4) it will be 12. Therefore, the choice of the algorithm will determine if p wins or not by a difference of 2.

5.2.2 Borda-WCM

For the weighted case, since Average Fit has no weighted generalization, our competitor is the weighted variant of Reverse. To model what we see as a relatively realistic setting, we randomly chose weights according to the Zeta distribution (or equivalently, the unbounded Zipf law), which is the discrete equivalent of the Pareto distribution. These types of power law distributions are commonly used to model the distribution of wealth, and of income, in the population, and were most notably used by Pareto (1964). In principle, such distributions model the number of people whose wealth is w as proportional to $1/w^a$ for some constant $a > 1$. The results are depicted in Figure 3.

These results are similar to the unweighted case. Our algorithm performs comparably to Reverse, and as before, seems to have a slight advantage for low k values while Reverse possesses a slight advantage for higher k values.

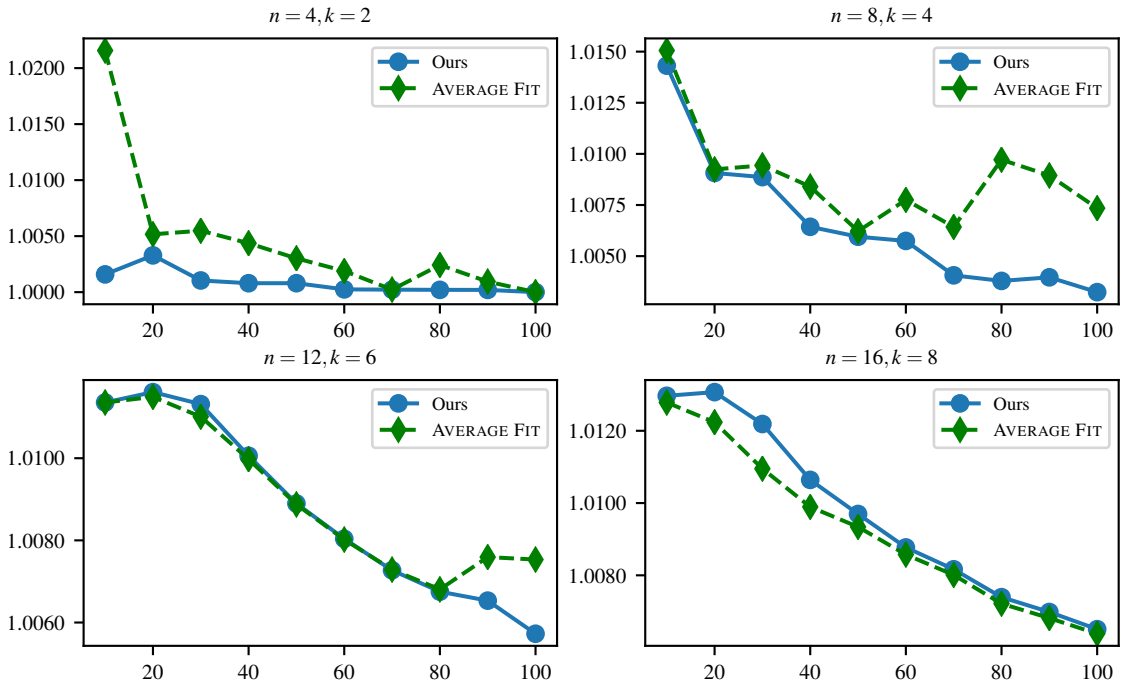

Figure 1: Experiments with voters whose preferences are drawn uniformly. Different subplots depict different values of n and k. The x-axis corresponds to different values of m. For each set of n, k, m, 100 experiments were conducted. The solid line represents the mean of the ratio between the value T obtained by our algorithm, and the C-LP fractional optimum $T^\star$, averaging over the 100 experiments. The dashed line represents the same mean w.r.t. Average Fit.

6. Related Work

We detail some of the previous work to help present our new results in context.

Tractability Results. The computational complexity of coalitional manipulation problems was studied extensively. For any scoring rule $\mathcal{R}_{\boldsymbol{\alpha}}$, much of the earlier work considered the case where the number of candidates is bounded. Conitzer, Sandholm, and Lang (2007, see Proposition 1) show that when m is bounded, $\mathcal{R}_{\boldsymbol{\alpha}}$-UCM is solvable in polynomial time.

Even when m is unbounded, Plurality-UCM and Veto-UCM are still easy using the greedy algorithm of Zuckerman et al. (2009). This also holds for t-approval-UCM, which generalizes both (Lin, 2012).

Recently Hemaspaandra and Schnoor (2016) showed that every scoring rule where $\boldsymbol{\alpha}$ consists of a constant number of unique coefficients is easy as well.

NP-Hardness Results. In the weighted case, the situation is different. For all positional scoring rules $\mathcal{R}_{\boldsymbol{\alpha}}$, except Plurality-like rules, $\mathcal{R}_{\boldsymbol{\alpha}}$-WCM is NP-hard when $m \geq 3$ (Conitzer et al., 2007; Hemaspaandra & Hemaspaandra, 2007; Procaccia & Rosenschein, 2007b). These results are based on a reduction from the well-known *partition* problem, which has a pseudo-polynomial algorithm; therefore, they do not immediately extend to the case where

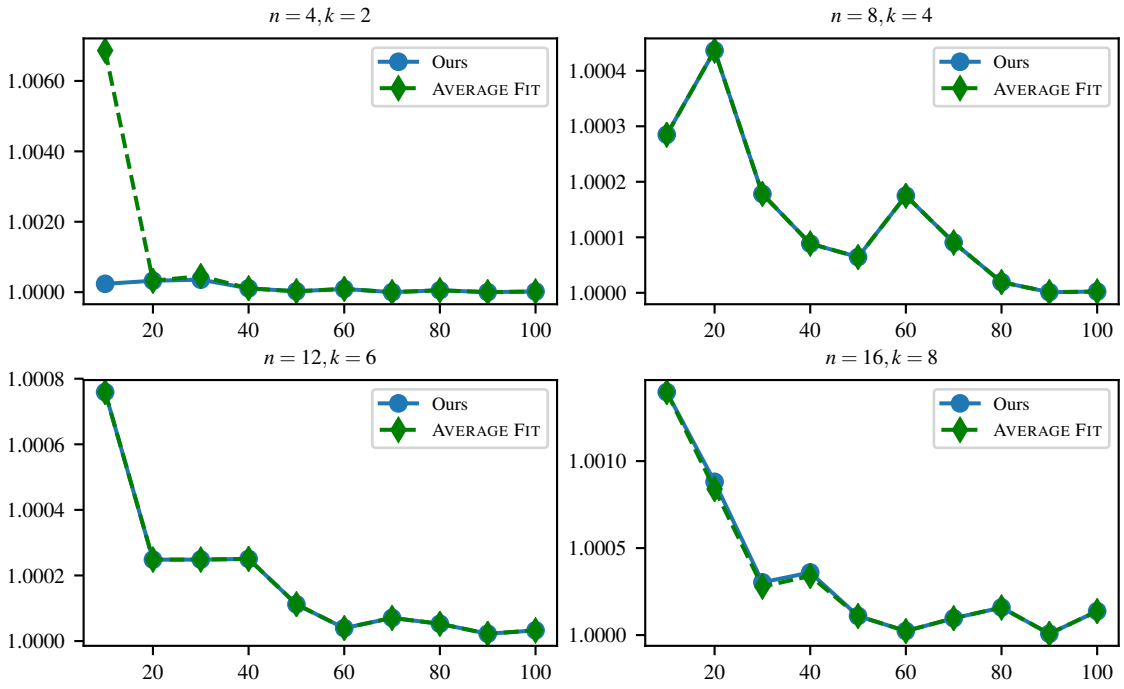

Figure 2: Experiments with voters whose preferences are drawn from an urn model. Different subplots depict different values of n and k. The x-axis corresponds to different values of m. For each set of n, k, m, 100 experiments were conducted. The solid line represents the mean of the ratio between the value T obtained by our algorithm, and the C-LP fractional optimum $T^\star$, averaging over the 100 experiments. The dashed line represents the same mean w.r.t. Average Fit.

the weights are relatively small, e.g., are integers bounded by a polynomial in the input size. When this is indeed the case, or the weights are encoded in a unary encoding, Veto-WCM, and t-approval-WCM for $t \geq 2$ remain NP-hard when the number of candidates is not fixed, by a reduction from *unary-3-partition* (Brelsford et al., 2008).

The computational hardness of Borda-UCM still remained open for quite some time, until it was finally shown to be NP-hard as well (Davies et al., 2011; Betzler et al., 2011), even for the case of $n = 3$ and adding $k = 2$ manipulators.

Approximating the number of manipulators. Zuckerman et al. (2009) presented a greedy algorithm later referred to as Reverse. As mentioned, for Borda-UCM, Reverse can be seen as an additive +1-approximation for the objective of finding the minimum number of manipulators needed.

For Borda-WCM, their approximation can be described as follows. Let $\mathbf{w} = (w_1, \ldots, w_k)$ be the weights of the k given weighted manipulators. If a p-winning strategy using these k manipulators exists, a p-winning strategy using additional manipulators will be found, if the sum of weights of the additional manipulators equals $\max_{\ell=1,\ldots,k} w_\ell$.

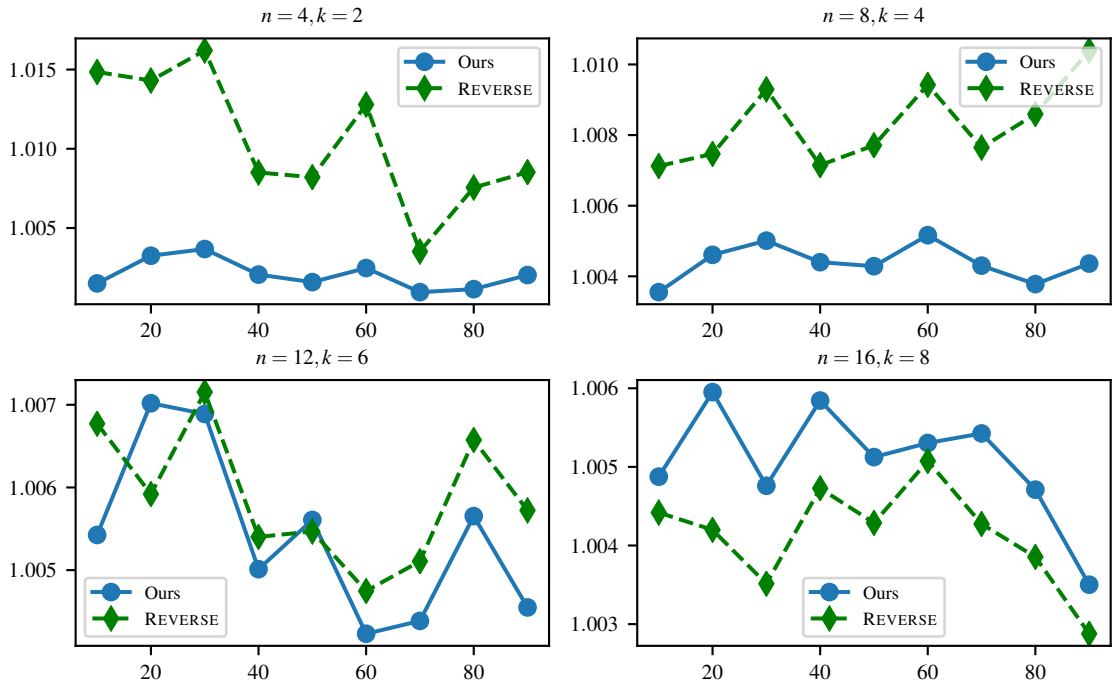

Figure 3: Experiments for weighted voters whose weights are drawn from the Zeta distribution, and their preferences are drawn uniformly. Different subplots depict different values of n and k. The x-axis corresponds to different values of m. For each set of n, k, m, 90 experiments were conducted. The solid line represents the mean of the ratio between the value T obtained by our algorithm, and the C-LP fractional optimum $T^\star$, averaging over the 90 experiments. The dashed line represents the same mean w.r.t. Reverse.

For more general results, Xia et al. (2010) provide an additive $(m-2)$-approximation for $\mathcal{R}_{\boldsymbol{\alpha}}$-UCM. In the case of $\mathcal{R}_{\boldsymbol{\alpha}}$-WCM, each of the extra manipulators will have a weight of at most $\max_{\ell=1,\ldots,k} w_\ell/2$.

Approximating the Maximum Score of a Competitor. For $\mathcal{R}_{\boldsymbol{\alpha}}$-WCM, when m is bounded, Brelsford et al. (2008, see Lemma 3 therein) provide an FPTAS with respect to the maximum score of a competitor. In their work, this FPTAS paves the way for an FPTAS for another objective, namely the difference between the score margin when not including the manipulator votes and the optimal score margin when including them.

Heuristics for Borda. Davies et al. (2014) present two heuristics that iteratively assign the largest un-allocated score either to the candidate with the largest gap (Largest Fit), or to the candidate with the largest ratio of gap divided by the number of scores yet-to-be-allocated to this candidate (Average Fit). To the best of our knowledge, these algorithms do not have a counterpart for the weighted case.

Configuration Linear Programs. As discussed, configuration linear programs were used for scheduling problems, e.g., for the following two problems which were extensively studied in the past:

- In the so-called *Santa Claus problem* (Bansal & Sviridenko, 2006), Santa Claus has t presents that he wishes to distribute between m kids, and $p_{i,j}$ is the value that kid i has to present j. The goal is to *maximize* the happiness of the least happy kid: $\min_i \sum_{j \in S_i} p_{i,j}$, where S_i is the presents allocated to kid i.
- In the problem of *scheduling on unrelated machines* (Svensson, 2012). We need to assign t jobs to m machines, and $p_{i,j}$ is the time required for machine i to execute job j. The goal is to *minimize* the makespan $\max_i \sum_{j \in S_i} p_{i,j}$, where S_i is the jobs assigned to machine i.

Both papers researched a natural and well-studied 'restricted assignment' variant of their respective problems where each present (resp. job) can be allocated only to a subset of the children (resp. machines), and that for this specific subset, it has the same value (resp. required time) p_j. Formally, this means that $p_{i,j} \in \{p_j, 0\}$ (in the Santa Claus problem) or that $p_{i,j} \in \{p_j, \infty\}$ (in scheduling on unrelated machines). Bansal and Sviridenko (2006) obtained a multiplicative $O(\log \log m / \log \log \log m)$-approximation to the first problem and Svensson (2012) obtained a multiplicative $(33/17 + \epsilon)$-approximation to the second.

7. Conclusions

In this work, we studied both the weighted and unweighted variants of the coalitional manipulation problem, for the case in which the voting rule can be any scoring rule. We believe the innovation in this paper is twofold, one conceptual and the other technical.

Conceptually, we have somewhat rekindled the line of research which focuses on objective functions that are not necessarily the number of manipulators required in order to make p win – in our case, the score margin.

We justified the return to this line of research by arguing that sometimes, "one extra manipulator" is too coarse of a resolution when studying the approximation factors provided by algorithms for the problem. Consequently, focusing instead on the margin yields a more fine-grained approximation which enables us to find a winning strategy for p in cases where other methods would not have found one. In another paper published by the current authors (Keller et al., 2018), we also showed that approximating the score margin is a key to obtaining an approximation on the number of manipulators, but this time for the related Bribery problem. Our previous work is thus providing an additional argument for the significance of this view.

On the technical side, we showed that the natural linear program formulations of our problems have an inherent limitation w.r.t. the approximation that they can provide, and thus we presented and advocated the use of *configuration linear programs*, which—to the best of our knowledge—is the first use of such LPs in this field.

There are several interesting directions for future work which we provide below. Regarding open problems:

1. Can we aim at tighter approximations, or prove the NP-hardness of finding a p-winning strategy when p can win by only a small margin, for example $O(\sqrt{m})$ for Borda (in other words, when the optimal score margin is negative, but has an absolute value in $O(\sqrt{m})$)?

2. Find other approximation factor trade-offs w.r.t. m and k. For instance, in another work (Keller et al., 2018), we showed that the natural LP formulation for $\mathcal{R}_{\boldsymbol{\alpha}}$-UCM enables—using an elaborate rounding scheme—an additive $\widetilde{O}(\alpha_1\sqrt{k})$-approximation. However, given the $\Omega(m)$ integrality gap for Borda we have shown in the current paper, this raises the immediate question: are approximation guarantees in the $\Omega(\alpha_1)$, $\widetilde{O}(\alpha_1\sqrt{k})$ range obtainable?

3. For scoring rules $\mathcal{R}_{\boldsymbol{\alpha}}$, there is a clear dichotomy as to when $\mathcal{R}_{\boldsymbol{\alpha}}$-WCM is NP-hard. Can a similar dichotomy be defined for $\mathcal{R}_{\boldsymbol{\alpha}}$-UCM and for $\mathcal{R}_{\boldsymbol{\alpha}}$-WCM when weights are known to be polynomial?

4. Define a taxonomy of instances according to which different manipulation strategies outperform others.

As for more general directions, we believe that configuration LPs are an important tool, with more applications yet to be revealed. Specifically, it would be interesting to investigate whether they can be applied to other voting methods, or even more generally, to other problems in computational social choice, such as fair division of multiple indivisible goods.

Acknowledgments

We thank Sarit Kraus and Ariel Procaccia for insightful discussions.

This work was supported by the Israel Science Foundation, under Grant No. 1488/14 and Grant No. 1394/16.

References

Andersen, M. S., Dahl, J., & Vandenberghe, L. (2016). CVXOPT: A Python package for convex optimization, version 1.1.9. cvxopt.org.

Bansal, N., & Sviridenko, M. (2006). The Santa Claus problem. In *STOC*, pp. 31–40. ACM.

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

Betzler, N., Niedermeier, R., & Woeginger, G. J. (2011). Unweighted coalitional manipulation under the borda rule is NP-hard. In *IJCAI*, pp. 55–60. IJCAI/AAAI.

Brelsford, E., Faliszewski, P., Hemaspaandra, E., Schnoor, H., & Schnoor, I. (2008). Approximability of manipulating elections. In *AAAI*, pp. 44–49. AAAI Press.

Caprara, A., Kellerer, H., Pferschy, U., & Pisinger, D. (2000). Approximation algorithms for knapsack problems with cardinality constraints. *European Journal of Operational Research*, *123*(2), 333–345.

Cary, D. (2011). Estimating the margin of victory for instant-runoff voting. In *EVT/WOTE*. USENIX Association.

Chevaleyre, Y., Lang, J., Maudet, N., & Ravilly-Abadie, G. (2009). Compiling the votes of a subelectorate. In *IJCAI*, pp. 97–102.

Conitzer, V., Sandholm, T., & Lang, J. (2007). When are elections with few candidates hard to manipulate?. *J. ACM*, *54*(3), 14.

Conitzer, V., & Walsh, T. (2016). Barriers to manipulation in voting. In *Handbook of Computational Social Choice*, pp. 127–145. Cambridge University Press.

Davies, J., Katsirelos, G., Narodytska, N., & Walsh, T. (2011). Complexity of and algorithms for Borda manipulation. In *AAAI*. AAAI Press.

Davies, J., Katsirelos, G., Narodytska, N., Walsh, T., & Xia, L. (2014). Complexity of and algorithms for the manipulation of Borda, Nanson's and Baldwin's voting rules. *Artificial Intelligence*, *217*, 20–42.

Elkind, E., & Faliszewski, P. (2010). Approximation algorithms for campaign management. In *WINE*, Vol. 6484 of *Lecture Notes in Computer Science*, pp. 473–482. Springer.

Elkind, E., Faliszewski, P., & Slinko, A. M. (2009). Swap bribery. In *SAGT*, Vol. 5814 of *Lecture Notes in Computer Science*, pp. 299–310. Springer.

Ephrati, E., & Rosenschein, J. S. (1993). Multi-agent planning as a dynamic search for social consensus. In *IJCAI*, pp. 423–431. Morgan Kaufmann.

Faliszewski, P. (2008). Nonuniform bribery. In *AAMAS (3)*, pp. 1569–1572. IFAAMAS.

Faliszewski, P., Hemaspaandra, E., & Hemaspaandra, L. A. (2009). How hard is bribery in elections?. *J. Artif. Intell. Res.*, *35*, 485–532.

Faliszewski, P., & Procaccia, A. D. (2010). AI's war on manipulation: Are we winning?. *AI Magazine*, *31*(4), 53–64.

Faliszewski, P., & Rothe, J. (2016). Control and bribery in voting. In *Handbook of Computational Social Choice*, pp. 146–168. Cambridge University Press.

Gibbard, A. (1973). Manipulation of voting schemes: a general result. *Econometrica: journal of the Econometric Society*, 587–601.

Grötschel, M., Lovász, L., & Schrijver, A. (1981). The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, *1*(2), 169–197.

Hemaspaandra, E., & Hemaspaandra, L. A. (2007). Dichotomy for voting systems. *J. Comput. Syst. Sci.*, *73*(1), 73–83.

Hemaspaandra, E., & Schnoor, H. (2016). Dichotomy for pure scoring rules under manipulative electoral actions. In *ECAI*, Vol. 285 of *Frontiers in Artificial Intelligence and Applications*, pp. 1071–1079. IOS Press.

Hoeffding, W. (1963). Probability inequalities for sums of bounded random variables. *Journal of the American statistical association*, *58*(301), 13–30.

Karmarkar, N. (1984). A new polynomial-time algorithm for linear programming. *Combinatorica*, *4*(4), 373–396.

Karush, W. (1939). Minima of functions of several variables with inequalities as side conditions. Master's thesis, University of Chicago.

Keller, O., Hassidim, A., & Hazon, N. (2017). New approximation for Borda coalitional manipulation. In *AAMAS*, pp. 606–614. ACM.

Keller, O., Hassidim, A., & Hazon, N. (2018). Approximating bribery in scoring rules. In *AAAI*. AAAI Press.

Khachiyan, L. G. (1980). Polynomial algorithms in linear programming. *USSR Computational Mathematics and Mathematical Physics*, *20*(1), 53–72.

Kuhn, H. W., & Tucker, A. W. (2014). Nonlinear programming. In *Traces and emergence of nonlinear programming*, pp. 247–258. Springer.

Létocart, L., & Wiegele, A. (2016). Exact solution methods for the k-item quadratic knapsack problem. In *ISCO*, Vol. 9849 of *Lecture Notes in Computer Science*, pp. 166–176. Springer.

Lin, A. P. (2012). *Solving hard problems in election systems*. Ph.D. thesis, Rochester Institute of Technology.

Magrino, T. R., Rivest, R. L., & Shen, E. (2011). Computing the margin of victory in IRV elections. In *EVT/WOTE*. USENIX Association.

Makhorin, A. (2017). GLPK: GNU linear programming kit, version 4.61. www.gnu.org/software/glpk/glpk.html.

Pareto, V. (1964). *Cours d'économie politique*, Vol. 1. Librairie Droz.

Procaccia, A. D., & Rosenschein, J. S. (2007a). Average-case tractability of manipulation in voting via the fraction of manipulators. In *AAMAS*, p. 105. IFAAMAS.

Procaccia, A. D., & Rosenschein, J. S. (2007b). Junta distributions and the average-case complexity of manipulating elections. *J. Artif. Intell. Res.*, *28*, 157–181.

Satterthwaite, M. A. (1975). Strategy-proofness and Arrow's conditions: Existence and correspondence theorems for voting procedures and social welfare functions. *Journal of economic theory*, *10*(2), 187–217.

Schrijver, A. (1998). *Theory of linear and integer programming*. John Wiley & Sons.

Svensson, O. (2012). Santa Claus schedules jobs on unrelated machines. *SIAM J. Comput.*, *41*(5), 1318–1341.

Xia, L. (2012). Computing the margin of victory for various voting rules. In *EC*, pp. 982–999. ACM.

Xia, L., & Conitzer, V. (2010). Compilation complexity of common voting rules. In *AAAI*. AAAI Press.

Xia, L., Conitzer, V., & Procaccia, A. D. (2010). A scheduling approach to coalitional manipulation. In *EC*, pp. 275–284. ACM.

Zuckerman, M., Procaccia, A. D., & Rosenschein, J. S. (2009). Algorithms for the coalitional manipulation problem. *Artificial Intelligence*, *173*(2), 392–412.