Technical University Berlin
Electrical Engineering and Computer Science
Institute of Software Engineering and Theoretical Computer Science
Algorithmics and Computational Complexity (AKT)

Coalitional Manipulation for Multiwinner Elections: Algorithms and Experiments

Lydia Kalkbrenner

Thesis submitted in fulfillment of the requirements for the degree
"Bachelor of Science" (B. Sc.) in the field of Computer Science

December 2019

Supervisor and first reviewer:	Prof. Dr. Rolf Niedermeier
Second reviewer:	Prof. Dr. Markus Brill
Co-Supervisors:	Dr. Robert Bredereck and Andrzej Kaczmarczyk

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

Die selbstständige und eigenhändige Ausfertigung versichert an Eides statt:

Berlin, den	____________________	____________________________
	Datum	Unterschrift

Zusammenfassung

In dieser Arbeit befassen wir uns mit der Manipulation von zwei Wahlregeln, k-*Borda* und ℓ-*Bloc*, die für exzellenzbasierte Wahlen verwendet werden, in der mehrere zur Kandidatur Stehende gewinnen. In einer Manipulation gibt eine Gruppe von Wählenden nach einer Strategie Stimmen ab, die nicht ihrer wahren Präferenz entsprechen. Damit soll ein besseres Ergebnis erzielt werden. Für beide Wahlregeln betrachten wir sowohl die allgemeine Variante, als auch die konsistente Variante, in der alle Manipulierenden geschlossen abstimmen. Wir schlagen einen Algorithmus vor, der die konsistente Variante der k-*Borda*-Manipulation in Polynomzeit löst. Weiterhin stellen wir die Behauptung auf, dass bestimmte Varianten von nahezu konsistenter Manipulation polynomzeitlösbar bleiben und entwickeln weitere Algorithmen für die nahezu konsistente Variante der k-*Borda*-Manipulation.

Wir evaluieren existierende Algorithmen von Bredereck u. a. [BKN17] für die Manipulation von ℓ-*Bloc* nach Laufzeit und Effektivität. Dazu verwenden wir Wahldaten aus der realen Welt und generieren Testdaten nach verschiedenen Verteilungen. Hierbei bestätigen wir die theoretischen Laufzeiten und können sie sogar verbessern. Obwohl die allgemeine Variante deutlich schwieriger in der Berechnung ist, stellen wir fest, dass in beiden Varianten ähnlich viele zur Kandidatur Stehende durch eine Manipulation ausgetauscht werden.

Abstract

In our work we consider coalitional manipulation of two different voting rules, k-*Borda* and ℓ-*Bloc*, that are used in excellence-based multiwinner elections. In coalitional manipulation a group of manipulators cast untruthful votes to be better off in the election. For both voting rules we consider the consistent variant, where all manipulators cast exactly the same vote, and the general variant, where consistency is not required. We propose an algorithm to solve the manipulation problem for k-*Borda* in polynomial time for the consistent variant. Moreover we conjecture that almost-consistent variants stay polynomial-time solvable and develop algorithms to solve almost-consistent variants of k-*Borda* manipulation.

We evaluate running times and effectiveness of ℓ-*Bloc* manipulation by implementing algorithms from the work of Bredereck et al. [BKN17] using both real-world data and generated test data from different distributions. We are able to confirm and even improve the running times. Although the inconsistent variant is more computationally demanding, we observe that the average number of candidates exchanged in a winning group is roughly the same for both variants.

Contents

1 Introduction

In social choice theory, the question whether an election can be efficiently manipulated comes naturally. In the *coalitional manipulation* setting, a group of manipulative voters casts untruthful votes to be better off as a group. This setting is also called *strategic voting*.

In this work, we study algorithms for the coalitional manipulation of *ℓ-Bloc* and *k-Borda* multiwinner elections in shortlisting applications. The work of Bredereck et al. [BKN17] on *ℓ-Bloc* serves as an inspiration for new algorithms for manipulation of *k-Borda*. We provide an implementation of the algorithms proposed by Bredereck et al. [BKN17] on GitHub.

Delivering algorithms for *k-Borda* is relevant, because *k-Borda* fulfills the properties of *unanimity* and *committee monotonicity* which are desirable properties for shortlisting applications. We describe these properties more detailed in Section 1.1. Voting rules similar to *k-Borda* are also used in real-world applications such as Formula 1 racing or in the Eurovision Song Contest.

In our model of coalitional manipulation we assume that the manipulative voters have knowledge about all other voters' preferences. Although this is rarely the case in real-world applications, we made this choice because our goal is to study the complexity of manipulation itself and not the complexity of guessing other voters' votes. For missing knowledge about the other votes we refer to Endriss et al. [End+16], Conitzer and Sandholm [CS02], and Scheuerman et al. [Sch+19].

In our work, we consider a variant of manipulation where all manipulators cast exactly the same vote. In real-world applications this is often the case for example if members in a political party want to show their common agreement. Assuming consistency among the manipulators also makes computation easier.

We now give a small overview on the aspects and difficulties of multiwinner elections and manipulation by using an example.

Example 1. A group of five people wants to give a collective gift to their friend. They collected a list of five ideas that could be possible candidates for a gift. Since they only want to get three things in total, they have to make a selection from the candidates on the list. For choosing the three candidates, they want to use a multiwinner voting rule, which takes all of their different preferences into account, which are shown in Figure 1.1.

Each ordering of candidates in Figure 1.1 is a *linear order*. For example, Anh's preferences are a linear order $\succ_{\text{Anh}}$ and can be written as

$$P \succ_{\text{Anh}} C \succ_{\text{Anh}} A \succ_{\text{Anh}} D \succ_{\text{Anh}} T.$$

Figure 1.1: Possible candidates for a gift: chocolate cake C, drawing utensils D, theater tickets T, flowerpot P, photo album A. The preferences are noted in a descending order from the most preferred to the least preferred candidate. The two first positions in every voters' preference are printed in bold for the use in 2-*Bloc.*

Anh	Bob	Charlie	Dimi	Esme
P	**C**	**T**	**T**	**D**
C	**P**	**A**	**D**	**T**
A	A	C	A	P
D	T	D	P	A
T	D	P	C	C

Figure 1.2: The scores according to the election using 2-*Bloc*. The candidates whose scores are printed in bold are the winning candidates.

	2-*Bloc* scores
theater tickets T	**3**
chocolate cake C	**2**
drawing utensils D	**2**
flowerpot P	2
photo album A	1

An easy variant of considering all voters' preferences would be to let every voter choose a fixed number of their favorite candidates. In this example we fix two approvals per voter to be distributed to the candidates. To determine the winners, we compute the score for every candidate by summing up how many voters approved the candidate and choose three candidates with the highest score. This voting rule is called 2-*Bloc*. We chose 2-*Bloc* in this example because it makes the outcome computed interesting with regards to ties and breaking ties. For 2-*Bloc*, we computed the candidates' scores in Figure 1.2. We can see that chocolate cake, drawing utensils and flowerpot all have two approvals each. To select the winning candidates, we have to use a tie-breaking method.

Bredereck et al. [BKN17, pp. 15–21] showed that breaking ties is an essential part of coalitional manipulation and in some cases already NP-hard. In Chapter 4 we evaluate our implementation of their algorithm.

To resolve the tie in our example, we use *lexicographic tie-breaking*, which chooses the candidates that are first in alphabetical order. Note that in lexicographic tie-breaking it is important to fix a predefined order. We fixed alphabetical order in this example, but there is no great meaning behind which candidate comes first according to the alphabet. Thus, the chocolate cake and drawing utensils are chosen from the tied candidates. This results in chocolate cake, drawing utensils and theater tickets winning the election. In Chapter 2 we introduce more tie-

Figure 1.3: Anh and Bob's modified votes (left) and the new 2-*Bloc* scores (winning candidates' scores in bold). The crossed out votes are the truthful preferences from Figure 1.1.

Anh		Bob	
~~P~~	C	~~C~~	C
~~C~~	A	~~P~~	A
~~A~~	P	~~A~~	P
~~D~~	D	~~T~~	T
~~T~~	T	~~D~~	D

	2-*Bloc* scores
photo album A	**3**
theater tickets T	**3**
chocolate cake C	**2**
drawing utensils D	2
flowerpot P	0

breaking rules and explain how we model multiwinner elections and manipulation formally. We now focus on how multiwinner elections can be manipulated and how to evaluate the outcome of a manipulation for the manipulators.

Let us come back to our example, where two people from the group try to manipulate the election. Anh and Bob are unhappy with the decision. They are short of money and think that drawing utensils are too expensive a gift. They would prefer to get the flowerpot or the photo album instead of the drawing utensils and want to find a way to manipulate the election by voting insincerely. From discussions among their friends, Anh and Bob know the preferences of the other voters and how they will vote. They take their friends' votes into account to compute how they can manipulate. In Figure 1.3 we can see their new votes and the resulting scores.

According to the new scores, the theater tickets and the photo album win the election together with another of the tied candidates. Due to *lexicographic tie-breaking* between the chocolate cake and the drawing utensils, the new winning group is:

$$\{\text{theater tickets, photo album, chocolate cake}\}.$$

Since Anh and Bob's main goal is to push drawing utensils out of the winning group, they strategically vote for a candidate that outperforms all tied candidates. Their vote would be considered as insincere because they actually prefer the flowerpot and the chocolate cake. By ranking the photo album first, they achieve a better outcome than if voting sincerely.

An even better outcome for Anh and Bob would be to convince another voter, for example Dimi, to join their manipulative group. If all three of them cast votes as shown in Figure 1.4, then they can push the flowerpot into the winning group, which they prefer over the photo album. In this case, Anh and Bob would not even have to vote insincerely.

As we can see in Figure 1.1, Dimi prefers drawing utensils and the photo album over the flowerpot. Anh and Bob can only convince her to change her vote in return for a small chocolate cake that she receives from them. Anh and Bob think that baking another chocolate cake to bribe Dimi is still less expensive than buying

Figure 1.4: Manipulators' votes (left) and the new 2-*Bloc* scores (winning candidates' scores in bold). The crossed out preference is Dimi's truthful preference from Figure 1.1.

Anh	Bob	Dimi
P	C	~~T~~ T
C	P	~~D~~ P
A	A	~~A~~ D
D	T	~~P~~ A
T	D	~~C~~ C

	2-*Bloc* scores
flowerpot P	**3**
theater tickets T	**3**
chocolate cake C	**2**
drawing utensils D	1
photo album A	1

the drawing utensils. The overall outcome for all three of the manipulative voters, Anh, Bob, and Dimi is very good because they can internally arrange themselves.

In Chapter 2 we introduce a measure to express how strongly a voter wants a candidate to win. To make this measure more fine-grained than the mere position in a voter's preference, we also allow values that differ from the positions. We call this measure the *utility* of a candidate. The *utility* can express both Anh and Bob's strong preference of pushing drawing utensils out of the winning group as well as Dimi's slight preference of drawing utensils over the flowerpot.

1.1 Motivation

For the singlewinner setting, where only one candidate is selected as winning candidate, there has been a lot of research in the past years [Dav+11] [BNW11] [CS02] [BR16] [CW16], whereas research in the multiwinner setting is still sparse. In this thesis we focus on the multiwinner setting.

In coalitional manipulation of *singlewinner voting rules*, one can always assume that the manipulative voters agree on one distinguished candidate, which they support to win the election. For *multiwinner elections*, where a group of candidates is winning, this is not the case. In this variant, already determining the possible goal of a manipulation is non-trivial.

For *multiwinner elections*, Faliszewski et al. [Fal+17, p. 28] propose excellence, diversity, and proportional representation as different goals of voting rules. In this work we focus on *excellence*-based voting rules, which refers to expert evaluation by a particular group of voters. The goal is to select a candidate group of finalists, a so called *shortlist*.

In *shortlisting*, *SNTV* and ℓ-*Bloc* are natural voting rules. *SNTV*, single non-transferable vote (1-*Bloc*) is a voting rule where each voter gives one approval to her most preferred candidate, whereas in ℓ-*Bloc* each voter gives one approval to each of her ℓ most preferred candidates. In k-*Borda*, scores are given in descending order according to the positions in each voter's preference. Both voting rules ℓ-*Bloc* and k-*Borda* are interesting because their scores can be trivially computed.

Let us compare the two voting rules. One desirable property is whether our voting rules extend the selected winning group when increasing the size of a winning group without removing anyone from it. A voting rule that fulfills this property is called *committee monotone*, which is the main normative principle for *shortlisting* [Fal+17, p. 36].

While k-*Borda* fulfills *committee monotonicity*, for ℓ-*Bloc*, we have to consider the choice of ℓ. A straightforward choice of ℓ would be to use the same number of approvals as candidates we want to choose, so $\ell = k$. This variant is called *Bloc*. Unfortunately, *Bloc* is not *committee monotone*, whereas k-*Borda* and ℓ-*Bloc* (with a fixed ℓ) fulfill *committee monotonicity*.

Another desirable property of scoring rules is *unanimity*. Scoring rules that select a candidate group as the only winning group if all voters select this group as their most preferred candidates are *unanimous*. Rules with this property use the whole preference profile of all voters, which is a desirable property when using preference-based elections. This is not the case for ℓ-*Bloc*, because this rule only uses the top ℓ preferences from each vote. Clearly, k-*Borda* uses the all information from a preference profile and is *unanimous*.

Consequently, k-*Borda*, which fulfills both *unanimity* and *committee monotonicity* [Elk+17, p. 31], is a good choice in shortlisting applications. Therefore this thesis focuses on k-*Borda* manipulation and considers ℓ-*Bloc* manipulation only in Chapter 4 for some experiments.

In Chapter 2 we discuss how to evaluate manipulators' utilities in a group of manipulative voters, as there are different ways to take the different utilities of the manipulators into account. In Chapter 3 we propose new algorithms for the consistent and inconsistent variant of k-*Borda* and in Chapter 4 we implement, compare, and evaluate different manipulation algorithms for ℓ-*Bloc*.

1.2 Related Work

Manipulation of voting rules has been studied extensively in literature, but research on coalitional manipulation of multiwinner elections is still sparse. For basics on manipulation in social choice theory we refer to Baumeister and Rothe [BR16] and Conitzer and Walsh [CW16].

Research on manipulation is often motivated by the implications of the Gibbard-Satterthwaite Theorem, which states that under a non-dictatorial voting rule and more than three candidates, there always exist elections in which a voter can be better off by lying about her true preference. An algorithmic approach was firstly studied by Bartholdi et al. [BTT89] and Bartholdi and Orlin [BO91]. Since then, computational aspects on coalitional manipulation of different singlewinner voting rules have been studied a lot, and were firstly initiated by Conitzer et al. [CLS03]. For the k-*Borda* voting rule we refer to Davies et al. [Dav+11], and Betzler et al. [BNW11], who proved that Singlewinner Borda (1-Borda) is already NP-hard for two or more manipulators and Conitzer and Sandholm [CS02] who characterized the exact numbers for which manipulation becomes hard.

For ℓ-*Bloc*, Lin [Lin11] showed that there is a polynomial time algorithm for the singlewinner variant.

Meir et al. [Mei+08] and Procaccia et al. [PRZ07] firstly introduced (non-coalitional) manipulation for multiwinner elections and showed that manipulation of *Bloc* remains polynomial-time-solvable. Obraztsova et al. [OZE13] extended this result for different tie-breaking rules and provide efficient algorithms for special cases of multiwinner scoring rules that are tractable.

Bredereck et al. [BKN17] studied coalitional manipulation of the ℓ-*Bloc* rule and deliver algorithms for tie-breaking and for various settings, for example, where all manipulators cast the same vote. Their algorithms will serve as an inspiration for building algorithms for coalitional manipulation of the k-*Borda* rule.

2 Preliminaries

In this chapter we explain how we model multiwinner elections and manipulations. To go through the basic definitions, we recall Example 1.

Multiwinner Elections. An *Election* (C, V) consists of a set C of m candidates, and of a multiset V of n linear orders, which are called *votes* and represent the voters' preferences.

Recalling Example 1, we have a set of candidates $C = \{$chocolate cake C, drawing utensils D, theater tickets T, flowerpot P, photo album $A\}$ and a set of votes $V = \{\succ_{\text{Anh}}, \succ_{\text{Bob}}, \succ_{\text{Dimi}}, \succ_{\text{Charlie}}, \succ_{\text{Esme}}\}$.

To determine possible winning candidate groups, we use a *multiwinner voting rule*, which is a function, that, given an election $E = (C, V)$ and an integer $k \in \{1, ..., |C|\}$, outputs a family of co-winning size-k subsets of C, called *k-excellence-groups*. We use the term *k-egroup* as an abbreviation for *k-excellence-group* to stress that we are focusing on excellence-based elections.

In this work we consider *committee scoring rules*, multiwinner voting rules that assign scores to candidates based on their positions in the votes. By score(c), we denote the total number of points that a candidate $c \in C$ obtains. A *committee scoring rule* outputs co-winning k-egroups with the maximum total sum of scores.

Let the *rank* $r(c)$ of a candidate c be the position in a vote.
Formally, let C be a set of candidates, V a set of votes, $|V| = n$. Let $\succ_j \in V$ be a vote that represents a voter j with $1 \leq j \leq n$. We use the set of predecessors $\{p \in C \mid p \succ_j c\}$ to define the rank r_j of a candidate $c \in C$ as follows:

$$r_j(c) := |\{p \in C \mid p \succ_j c\}| + 1 \tag{2.1}$$

In the introductory example we use a committee scoring rule, 2-*Bloc*, where the score depends on whether a candidate is among the first two positions of a vote or not. We now define the family of *ℓ-Bloc* voting rules formally, which are a generalization of this idea, together with *k-Borda* which is the other family of voting rules we focus on in this work.

- *ℓ-Bloc* assigns one point to each of the top $\ell < |C|$ candidates for each vote.
- *k-Borda* assigns $m - i$ points to each candidate of rank $i \in \{1, ..., m\}$ for each vote.

Figure 2.1: Scores according to the election in Example 1 using 3-*Borda* and 2-*Bloc* (winning candidates' scores in bold).

	3-*Borda* scores	2-*Bloc* scores
theater tickets T	**12**	**3**
chocolate cake C	9	**2**
drawing utensils D	9	**2**
flowerpot P	**10**	2
photo album A	**10**	1

Figure 2.2: Utility values of the manipulators Anh, Bob, and Dimi.

	Anh	Bob	Dimi
C	11	11	5
D	0	0	7
T	1	1	8
P	11	11	6
A	9	8	7

Note that for both voting rules, k is the parameter for the winning group size.

Recalling Example 1, 2-*Bloc* with three winning candidates ($k = 3$) outputs multiple winning k-egroups, namely $\{\{T, P, D\}, \{T, P, C\}, \{T, C, D\}\}$. We chose one winning k-egroup by *lexicographic tie-breaking* in the introductory example. When using 3-*Borda* for the same election, there is only one winning k-egroup, $\{T, P, A\}$ as we can see in Figure 2.1.

Coalitional Manipulation. To explain how we model manipulations of multiwinner elections, we recall Example 1 of Anh and Bob voting insincerely to achieve a better outcome.

Let $E = (C, V)$ be an election and r the number of manipulators. For each manipulator $i \in \{1, ..., r\}$ we define a *utility function* $u_i \colon C \to \mathbb{N}$ that outputs a utility value for each candidate.

Recalling Example 1 and the manipulative group of Dimi, Anh, and Bob, when using utility functions we are able to express more fine-grained preferences in comparison to some position in a sincere vote of a manipulator. This is possible because of the greater range of the utility values. The candidates also do not have to be strictly ordered.

In Figure 2.2 we give an example for utility values that complement Example 1. We can see that Bob and Anh's utilities are polarized between chocolate cake, flowerpot, and photo album and the more expensive gifts, theater tickets and drawing utensils. In comparison to that, Dimi's utilities vary only slightly, which indicates that Dimi does not have such a strong preference.

Figure 2.3: Evaluation values for different k-egroups and evaluation functions (highest values in bold).

k-egroup	*utilitarian*	*egalitarian*	*candidate-wise egalitarian*
$\{T, C, D\}$	44	12	6
$\{T, C, A\}$	61	**20**	**13**
$\{T, C, P\}$	**65**	19	12

We now extend the single manipulator's utility function to an *evaluation function* that evaluates a k-egroup for a group of manipulators. The straightforward way to do so is summing up all utility values. This variant is called *utilitarian* evaluation.

We observe that pushing the flowerpot into the winning k-egroup is the best outcome for Anh, Bob and Dimi according to *utilitarian* evaluation (Figure 2.3). According to Dimi's utility function in Figure 2.2, she prefers drawing utensils and the photo album over the flowerpot, which means that although the group is better off, Dimi herself is worse off. In our example, Anh and Bob have to bribe her to balance utilities within the manipulating group.

We define two other variants of evaluating manipulator's utilities, the *egalitarian* and *candidate-wise egalitarian* variant, to avoid single manipulators being worse off.

Let C be a set of candidates, $S \subseteq C$ be a k-egroup. Let $U = \{u_1, ..., u_r\}$ be a family of manipulator's utility functions where $u_i \colon C \to \mathbb{N}$ and $i \in \{1, ..., r\}$. We define the three evaluation functions $\text{eval} \in \{\text{util}, \text{candegal}, \text{egal}\}$ as follows:

$$\text{util}_U(S) := \sum_{u \in U} \sum_{c \in S} u(c) \tag{2.2}$$

$$\text{egal}_U(S) := \min_{u \in U} \sum_{c \in S} u(c) \tag{2.3}$$

$$\text{candegal}_U(S) := \sum_{c \in S} \min_{u \in U} u(c) \tag{2.4}$$

The *egalitarian* variant considers the utility of the least satisfied candidate of a k-egroup by summing up only this candidate's utilities.

The evaluation values for the utilities from Figure 2.2 are depicted in Figure 2.3. Different from the *utilitarian* evaluation, $\{T, C, P\}$ is not the most preferred k-egroup anymore, because Dimi's utilities for this k-egroup are very low. Instead, the manipulators would decide to push the photo album into the k-egroup.

The same result holds for using the *candidate-wise egalitarian* evaluation function. In this variant, we consider the least satisfied manipulator for every candidate of a k-egroup (Figure 2.3). As observed by Bredereck et al. [BKN17, p. 10], we can assume that there is a single utility function over the candidates in the *utilitarian* and *candidate-wise egalitarian* variant.

We call $v_{\text{eval}}(c)$ the *value* of a candidate $c \in C, \text{eval} \in \{\text{util}, \text{candegal}\}$.

$$v_{\text{util}}(c) := \sum_{u \in U} u(c) \tag{2.5}$$

$$v_{\text{candegal}}(c) := \min_{u \in U} u(c) \tag{2.6}$$

The value of a candidate is defined for the *utilitarian* and *candidate-wise egalitarian* variant, which simplifies the core of these problems and can be exploited by our algorithms.

Tie-Breaking. In Example 1 we explained the necessity of tie-breaking rules and proposed *lexicographic tie-breaking* to resolve tie situations. Before we introduce two more tie-breaking methods, we formally define tie-breaking rules.

A *multiwinner tie-breaking rule* is a mapping that, given an election and a family of co-winning k-egroups, outputs a single k-egroup. Candidates that are in all winning k-egroups are called *confirmed candidates* C^+, whereas candidates that do not appear in any k-egroup are called *rejected candidates* C^-. Candidates that appear in some k-egroups are called *pending candidates* P.

We now formally define three families of tie-breaking rules, $\mathcal{F}_{\text{lex}}, \mathcal{F}^{\text{eval}}_{\text{opt}}, \mathcal{F}^{\text{eval}}_{\text{pess}}$:

A tie-breaking rule F belongs to $\mathcal{F}_{\text{lex}}$ if and only if ties are broken lexicographically with respect to some predefined order $>_{\mathcal{F}}$ of the candidates from C. That is, $\mathcal{F}$ selects all candidates from C^+ and the top $k - |C^+|$ candidates from P with respect to $>_{\mathcal{F}}$.

A tie-breaking rule F belongs to $\mathcal{F}^{\text{eval}}_{\text{opt}}$, we say F is an *optimistic tie-breaking rule*, if and only if the evaluation function of the winning k-egroup is as high as possible. Formally, a k-egroup S is selected such that $C^+ \subseteq S \subseteq (C^+ \cup P)$ and there is no other k-egroup S' with $C^+ \subseteq S' \subseteq (C^+ \cup P)$ and $\text{eval}(S') > \text{eval}(S)$.

A tie-breaking rule F belongs to $\mathcal{F}^{\text{eval}}_{\text{pess}}$, we say F is a *pessimistic tie-breaking rule*, if and only if the evaluation function of the winning k-egroup is as low as possible. Formally, a k-egroup S is selected such that $C^+ \subseteq S \subseteq (C^+ \cup P)$ and there is no other k-egroup S' with $C^+ \subseteq S' \subseteq (C^+ \cup P)$ and $\text{eval}(S') < \text{eval}(S)$.

Although *lexicographic tie-breaking* seems to be more simple than *optimistic* or *pessimistic tie-breaking*, Bredereck et al. [BKN17, p. 13] showed the following

Proposition 2.1. *For every tie-breaking rule $\mathcal{F}^{\text{eval}}_{\text{bhav}}$ with* $\text{bhav} \in \{\text{opt}, \text{pess}\}$*,* $\text{eval} \in \{\text{util}, \text{candegal}\}$*, and fixed utility functions it is possible to find a tie-breaking rule $F \in \mathcal{F}_{\text{lex}}$ in polynomial time such that F and $\mathcal{F}^{\text{eval}}_{\text{bhav}}$ have the same output for all possible elections [BKN17, p. 13].*

According to Proposition 2.1, when using $\text{bhav} \in \{\text{util}, \text{candegal}\}$ it is sufficient to only prove Theorem 3.1 and 3.2, and Conjecture 3.1 from Chapter 3 for lexicographic tie-breaking.

Having tie-breaking rules and utility functions defined, we can define the *strength* of a candidate.

Strength Order. Like Bredereck et al. [BKN17, p. 26] do in their algorithm for consistent manipulation of *ℓ-Bloc*, we also introduce a *strength order* $>_{\mathcal{S}}$. It sorts the candidates in a descending order with respect to the score they receive from the non-manipulative votes and, as a second criterion, according to their position in $>_{\mathcal{F}}$. We say that a candidate c is *stronger* than candidate c' if $c >_{\mathcal{S}} c'$ according to the non-manipulative votes.

The computational problem of Coalitional Manipulation is stated as follows. Let $\mathcal{R}$ be a multiwinner voting rule and $\mathcal{F}$ be a multiwinner tie-breaking rule.

$\mathcal{R}$-$\mathcal{F}$-Coalitional Manipulation ($\mathcal{R}$-$\mathcal{F}$-eval-CM),
eval $\in$ {util, egal, candegal}.

Input: An election (C, V), an egroup-size $k < |C|$, r manipulators represented by their utility functions $U = \{u_1, ..., u_r\}$ where $u_i \colon C \to \mathbb{N}$ and a non-negative, integral evaluation threshold q

Question: Is there a size-r multiset W of manipulative votes over C such that k-egroup $S \subset C$ wins the election $(C, V \cup W)$ under $\mathcal{R}$ and $\mathcal{F}$, with $\text{eval}(S) \geq q$?

Maximum Weight Perfect Matchings. Let $G = (V, E, w)$ be a weighted graph and $w \colon E \to \mathbb{R}$ be a weight function.

A *matching* of G is a set $M \subseteq E(G)$ of pairwise disjoint edges. We call a matching M *perfect*, if every vertex of the graph is incident to exactly one edge of M.

The Maximum Weight Matching problem is to find a matching M such that the total edge weight $w(M) = \sum_{e \in M} w(e)$ is maximized among all matchings. In Chapter 3 we use the Maximum Weight Bipartite Perfect Matching, abbreviated by MWBPM, which maximizes the edge weights and also requires every vertex of the bipartite graph G to be matched. MWBPM can be solved in $\mathcal{O}(mn)$ time [DS12, p. 2].

The Knapsack Problem. In the Knapsack Problem we are given an item set N consisting of n *items* j with *profit* p_j and *weight* w_j, and a *capacity value* c. The objective is to select a subset of N such that the total profit of the selected items is maximized and the total weight does not exceed c [KPP04, p. 2].

Whereas the general Knapsack Problem is known to be NP-complete, versions exist that are efficiently computable.

In this work we use the Exact k-Item Knapsack Problem (E-kKP), where the number of items in a feasible solution must be *exactly equal* to k. This variant can be solved in $\mathcal{O}(k^2mr)$ time [KPP04, p. 272].

3 Manipulation of the k-Borda Rule

In this chapter we focus on the *k-Borda* voting rule, which is known to be computationally hard already in the single winner case for at least two manipulators, but solvable in polynomial time for one manipulator [BNW11] [Dav+11]. This leads us to the variant of *k-Borda* manipulation where manipulators vote consistently.

In Section 3.1 we show that the consistent variant can be solved in polynomial time in the *utilitarian* and *candidate-wise egalitarian* case.

In Section 3.2 we extend our results to a special variant of inconsistent manipulation, which we call One-Swap-CM and its generalization, which we call Almost-Consistent-CM. Both variants are similar to the consistent variant. Almost-Consistent-CM allows the manipulators to swap candidates in the vote of one manipulator while the other manipulators vote consistently. In Section 3.2.2 we conjecture that Almost-Consistent-CM of *k-Borda* is solvable in polynomial time when fixing the number of positions in the vote of one manipulator that differs from the consistent vote that the other manipulators cast.

3.1 Consistent k-Borda Manipulation

In this section we propose an algorithm to solve the consistent variant of *k-Borda* manipulation using Maximum Weight Bipartite Perfect Matching (MWBPM), which can be solved in $\mathcal{O}(mn)$ time [DS12, p. 2], with m denoting the number of edges, n denoting the number of vertices in each group. We define the algorithm in Section 3.1.1 and prove the correctness of the algorithm separately in Section 3.1.2.

The consistent variant of *k-Borda* manipulation is stated as follows:

k-Borda-$\mathcal{F}$-eval-Coalitional Manipulation with consistent manipulators for eval $\in$ {util, candegal} (Consistent-CM)

Input: An election (C, V), an egroup size $k < |C|$, r manipulators represented by their utility functions $U = \{u_1, ..., u_r\}$ where $u_i \colon C \to \mathbb{N}$ and a non-negative, integral evaluation threshold q.

Question: Is there a size-r multiset W of identical manipulative votes over C such that some k-egroup $S \subset C$ wins the election $(C, V \cup W)$ under $\mathcal{F}$ with $\text{eval}(S) \geq q$?

Theorem 3.1. *Let m be the number of candidates, n the number of voters, k the size of a desired egroup, and r the number of manipulators. One can solve* CONSISTENT-CM *for k-*Borda *in time $\mathcal{O}(m(n + r + m^2(m + n)))$ for any* eval $\in \{\text{util}, \text{candegal}\}$ *and $\mathcal{F} \in \{\mathcal{F}_{\text{lex}}, \mathcal{F}^{\text{eval}}_{\text{opt}}, \mathcal{F}^{\text{eval}}_{\text{pess}}\}$.*

To solve CONSISTENT-CM, the proposed Algorithm 3.1 outputs a single linear order of candidates. In the consistent setting, a manipulation is fully described by one linear order because all manipulators cast exactly the same vote.

We prove Theorem 3.1 for a lexicographic tie-breaking rule $\mathcal{F} \in \mathcal{F}_{\text{lex}}$. This is sufficient since, using Proposition 2.1, one can generalize the proof for the cases of *utilitarian* and *candidate-wise egalitarian* variants. Before we describe how the algorithm works, we introduce some concepts which are used in the algorithm.

A Candidate's Gain and the Dropped Candidate. We define the *gain* of a candidate as a mapping g from ranks to the number of additional points that the candidate gets at that rank. We also define the *maximal gain* $g_{\max}$. Formally, the *gain* $g\colon \{1, ..., m\} \to \mathbb{N}$ of CONSISTENT-CM and $g_{\max}$ can be defined as follows:

$$g(x) := r \cdot (m - x) \tag{3.1}$$

$$g_{\max} := \max\{g(1), ..., g(m)\} \tag{3.2}$$

For CONSISTENT-CM we observe that $g_{\max} = r \cdot (m - 1) = g(1)$. We define H_i as the set of the $i \in \mathbb{N}$ ranks with the highest-valued gains as follows:

$$H_i \subseteq \{1, ..., m\} \text{ with } |H_i| = i \text{ and } \forall h \in H_i, j \in \{1, ..., m\} \backslash H_i : g(h) \geq g(j) \tag{3.3}$$

For CONSISTENT-CM, the gain is monotonically decreasing, so we have $H_i = \{1, ..., i\}$.

We also introduce the concept of the *dropped candidate*, the strongest candidate that will not be part of the winning k-egroup. When fixing a dropped candidate d one also has to consider the additional gain $g(i_d)$ that the dropped candidate gets when being ranked at rank i_d by the manipulators.

3.1.1 Algorithm for Consistent k-Borda Manipulation

Algorithm 3.1. The basic idea of our algorithm is to fix certain parameters of a solution and reduce the resulting subproblem to the MWBPM problem. The algorithm iterates through all possible value combinations of the following parameters:

- the dropped candidate d and
- the position i_d that d is ranked at by the manipulators.

Fixing a combination of these two values also induces the final score z of the dropped candidate when being ranked at i_d. We use z further in the definition of our graph by partitioning candidates as follows.

The candidates that already have more than z points, or have the same amount of points and win by tie-breaking rule $\mathcal{F}$ are called *kept candidates* C^+. We denote the number of *kept candidates* by $t := |C^+|$. Let C^- be the set of candidates that cannot outperform d even when getting $g_{\max}$ additional points from the manipulators. Let C^* be the set of candidates with less than z points, or with score exactly z that loose against d according to the tie-breaking rule $\mathcal{F}$ but can still outperform d when getting additional points from the manipulators. We formally define C^+, C^*, and C^- as follows and call C^- the *rejected candidates.*

$$\begin{aligned}
C^+ :=& \{c \in C \setminus \{d\} \mid \mathrm{score}_V(c) > z\} \cup \{c \in C \setminus \{d\} \mid \mathrm{score}_V(c) = z \text{ and } c >_{\mathcal{F}} d\} \\
C^* :=& \{c \in C \setminus (C^+ \cup \{d\}) \mid \mathrm{score}_V(c) + g_{\max} > z\} \\
& \cup \{c \in C \setminus (C^+ \cup \{d\}) \mid \mathrm{score}_V(c) + g_{\max} = z \text{ and } c >_{\mathcal{F}} d\} \\
C^- :=& C \setminus (\{d\} \cup C^+ \cup C^*)
\end{aligned} \tag{3.4}$$

In each iteration of the algorithm, we check if the choice of d and i_d is feasible. We skip an iteration if $|C^+| > k$ or $|C^*| < k - t$, meaning that there are either too many kept candidates or that there are not enough candidates that can outperform d to form a size-k winning egroup. We then find an ordering R of candidates by finding a matching between candidates $C \setminus \{d\}$ and ranks $M := \{1, ..., m\} \setminus \{i_d\}$ in the following graph.

Let $G_d^{i_d} = \{C \setminus \{d\} \cup M, E\}$ be a bipartite graph with $E = E^0 \cup E^+ \cup E^-$, where E^0, E^+ and E^- as follows:

$$\begin{aligned}
E^0 :=& \{\{c, j\} \mid c \in C^+ \cup C^-, j \in M \setminus H_{k-t}\}, \\
E^+ :=& \{\{c, j\} \mid c \in C^*, j \in M \cap H_{k-t}, c \text{ outperforms } d \text{ with } g(j) \text{ points}\}, \\
E^- :=& \{\{c, j\} \mid c \in C^*, j \in M \setminus H_{k-t}, c \text{ does not outperform } d \text{ with } g(j) \text{ points}\}.
\end{aligned} \tag{3.5}$$

Let $f\colon E \to \mathbb{R}$ be the weight function. The weight of an edge $\{c, j\} \in E$ is defined as follows:

$$f(\{c, j\}) = \begin{cases} v_{\mathrm{eval}}(c) & \text{if } \{c, j\} \in E^+, \\ 0 & \text{otherwise.} \end{cases} \tag{3.6}$$

We observe, that a solution to our MWBPM-instance uniquely defines an ordering of candidates in $C \setminus \{d\}$ by matching each candidate to some rank. After finding a solution of MWBPM and the corresponding ordering R, the manipulators achieve an outcome that maximizes their utilities for the choice of t if they vote consistently with R.

For every iteration of the algorithm, we compute the evaluation function for the resulting k-egroup $S \subseteq C$ to pick the ordering from an iteration that maximizes

Figure 3.1: The non-manipulative preference profile for Example 2 and the resulting Borda scores (winning candidates' scores in bold).

v_1	v_2	3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	
a	a	a	a	a	a	a	a	a	b	b	b	$s(a) = \mathbf{38}$
b	b	b	b	c	c	c	c	c	c	c	d	$s(b) = \mathbf{30}$
d	d	d	d	d	d	d	d	b	a	d	c	$s(c) = \mathbf{27}$
c	c	c	c	b	b	b	b	d	d	e	e	$s(d) = 23$
e	e	e	e	e	e	e	e	e	e	a	a	$s(e) = 2$

the evaluation function. Note, that it is not sufficient to compare the sum of the edge weights for every iteration, since we also have to take the value of the kept candidates into account.

To demonstrate how the construction of the graph works, we show the following example.

Example 2. Let $C = \{a, b, c, d, e\}$ be the candidates of an election and $n = 12$ be the number of non-manipulative votes. We fix $k = 3$ and $r = 2$. We consider the distribution of non-manipulative scores by voters $\{v_1, ..., v_n\}$ depicted in Figure 3.1. The graph $G_d^{i_d}$ in Figure 3.2 is constructed in the iteration $t = 1$ by Algorithm 3.1, which means that candidate a is the kept candidate and b is the dropped candidate.

3.1.2 Proof of Theorem 3.1

In every solution of Algorithm 3.1, all candidates that are pushed into the k-egroup are always ranked at positions where they have the highest-valued gains in the votes of the manipulators. We defined these positions in H_{k-t} in Equation 3.3.

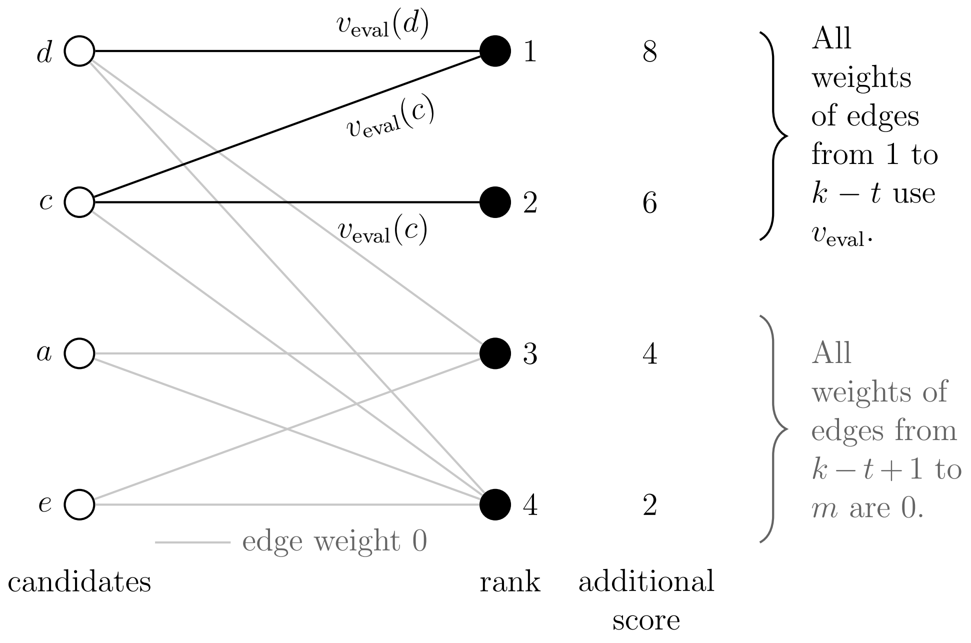
There might also exist different solutions, that cannot be found by Algorithm 3.1. For our previous Example 2, Figure 3.3 shows a different consistent vote that the manipulators could cast, which would also result in $S = \{a, c, d\}$ winning the election and cannot be found by Algorithm 3.1.

However, Algorithm 3.1 always finds a solution where the candidates that are pushed into the k-egroup get as many points as possible, even if they could also be ranked lower and would still win. We indeed show that the approach Algorithm 3.1 takes is sufficient in the following Lemma 3.1.

Lemma 3.1. *For every optimal winning k-egroup $S \subset C$ and its number t of kept candidates there exists an ordering R that can be found by Algorithm 3.1 as a matching in the graph in iteration t and that results in S winning the election.*

Proof of Lemma 3.1. Let $S \subset C$ be a k-egroup that gives the optimal outcome of the manipulation according to eval(S). Let t be the number of kept candidates in S. Let R be an ordering of candidates that results in S winning the election if the

Figure 3.2: Example graph $G_d^{i_d}$ for $t = 1$ in Algorithm 3.1 with dropped candidate b and its position $i_b = 5$, kept candidate $C^+ = \{a\}$, and sets $C^* = \{c, d\}$, and $C^- = \{e\}$.



There is no edge from d to rank 2 because d cannot outperform the dropped candidate b with 6 additional points. Candidate c can outperform b with either 8, 6 or 4 additional points. This is why there are edges $\{c, 1\}, \{c, 2\}$, but no edge from c to 3 because $3 \notin H_{k-t}$. Candidate a is the kept candidate and only has 0-weighted edges to ranks lower than $k - t$. The same holds for e because it is too weak to outperform b.

manipulators vote consistently according to R. Suppose R cannot be represented as a matching in the respective graph $G_d^{i_d}$. This is only the case if some edge $\{c_i, j\}$ is missing in $G_d^{i_d}$. By the definition of the graph, edge $\{c_i, j\}$ is only missing if one of the following holds:

1. $j \in H_{k-t}$ and candidate c_i cannot outperform the dropped candidate with $g(j)$ additional points,
2. $j \notin H_{k-t}$ and candidate c_i, that is not a kept candidate, nor a rejected candidate, $c_i \in C^*$, can outperform d with $g(j)$ additional points, or
3. $j \in H_{k-t}$ and candidate c_i is a kept or rejected candidate, $c_i \in C^+ \cup C^-$.

If the first condition holds, then giving $g(j)$ additional points to c_i does not result in pushing c_i into the winning k-egroup. In this case, there must be another

Figure 3.3: Two consistent votes that the manipulators could cast, which both result in $S = \{a, c, d\}$ winning the election for the iteration $t = 1$ and $k = 3$ (winning candidates' scores in bold). Vote 2 cannot be found by the algorithm. Note that b is the dropped candidate and a already outperforms b in the non-manipulative preference profile in Figure 3.1. Vote 1 results from Example 2 when using Algorithm 3.1.

vote 1	vote 2
d	d
c	a
a	c
e	e
b	b

$s_1(a) = \mathbf{42}$	$s_2(a) = \mathbf{44}$
$s_1(c) = \mathbf{33}$	$s_2(c) = \mathbf{31}$
$s_1(d) = \mathbf{31}$	$s_2(d) = \mathbf{31}$
$s_1(b) = 30$	$s_2(b) = 30$
$s_1(e) = 4$	$s_2(e) = 4$

candidate c_g that gets pushed into the winning k-egroup with less than $g(k-t)$ additional points, i.e., c_g is ranked at a lower position than $k-t$ in R. Otherwise, the number of kept candidates would differ from our assumption. We define a new ordering R' with the same positions of candidates, except swapping positions of c_i and c_g. Voting with R' results in the same k-egroup S because c_g still gets pushed into the k-egroup after gaining more points than before and c_i is still not part of the k-egroup because the score of c_i is lower.

If the second condition holds, then giving $g(j)$ additional points to c_i results in pushing c_i into the winning k-egroup. In this case there must be another candidate c_g that does not get pushed into the winning k-egroup with $g(k-t)$ additional points or more, i.e., c_g is ranked at $k-t$ or a greater position. With a similar argument as seen before, we can swap positions of these two candidates in a new ordering R'.

In the third case, candidate c_i is a kept or rejected candidate. So either c_i was already part of the winning group, before the manipulative votes were cast, or c_i cannot be part of the winning k-egroup. This means that there must be another candidate c_g that gets pushed into the winning egroup with less than $g(k-t)$, i.e., c_g is ranked at a lower position than $k-t$ in R. Otherwise the dropped candidate or the k-egroup-size would differ from our assumption. We define a new ordering R' with the same positions of candidates, except swapping positions of c_i and c_g. Voting with R' results in the same egroup because c_g is still part of the winning group when getting more points than before and c_i is also either part or not part of the winning group by the definition of a kept or rejected candidate.

We showed that for every ordering R we can find an R' that results in the same k-egroup winning and that can be represented as a matching in some of the graphs constructed by Algorithm 3.1. □

Since Algorithm 3.1 always considers orderings that result in the optimal k-egroup winning, we have to make sure that the algorithm also chooses the solution that results in the k-egroup with the highest value of the evaluation function winning.

Lemma 3.2. *For every optimal winning k-egroup $S \subset C$, its number t of kept candidates and an ordering R, which is the solution of Algorithm 3.1 and leads to k-egroup $S' \subset C$ winning with its number of kept candidates t', eval(S') cannot be lower than eval(S) in our optimal solution.*

Proof of Lemma 3.2. Suppose $\text{eval}(S') < \text{eval}(S)$ for the same number of kept candidates, $t = t'$ and ordering R' that is output of Algorithm 3.1 in iteration t' and leads to S' winning. According to Lemma 3.1, we can find an ordering R in the same graph that results in S winning the election. If $\text{eval}(S') < \text{eval}(S)$ then the sum of the edge weights from R' is lower than of R. We can only assume this because $t = t'$ which means that the evaluation function of the kept candidates does not change. Hence, the algorithm for MWBPM must choose R instead of R'.

Suppose $t \neq t'$. For every iteration of Algorithm 3.1, an ordering with the highest value of the evaluation function of the resulting k-egroup is found. According to Lemma 3.1, Algorithm 3.1 can find an ordering R that leads to S winning in a different iteration. Since the algorithm computes the winning k-egroup and its evaluation function for every iteration, Algorithm 3.1 must choose R instead of R'. □

Proof of Theorem 3.1. Consider an instance of Consistent-CM with an election $E = (C, V)$ where C is a candidate set and V is a set of non-manipulative votes, an egroup size k, r manipulators, and a tie-breaking rule $\mathcal{F}$.

From Lemma 3.1 and 3.2 we can conclude the correctness of Theorem 3.1.

To analyze the running time of Algorithm 3.1, several steps need to be considered. At the beginning, we have to compute the scores that the candidates get from the non-manipulative voters and then sort the candidates to obtain the strength order $>_S$. Computing $>_S$ takes $\mathcal{O}(\ell n + m \log m)$ time. We also compute the values of all candidates, which takes $\mathcal{O}(mr)$. We have m possible choices of a dropped candidate. For each such choice there are m possible values of additional scores by the manipulators because in the consistent variant the approvals are fixed to $s \in \{r \cdot (m-1), r \cdot (m-2), ..., 0\}$. This makes at most m^2 possible combinations of a dropped candidate and its score. For every of these combinations, we have to construct the graph's adjacency matrix which will be an input of the MWBPM algorithm and takes $\mathcal{O}(m^2)$ time. MWBPM can be solved in $\mathcal{O}(mn)$ time [DS12, p. 2]. Summed up, the overall running time is $\mathcal{O}(m(n + r + m^2(m + n)))$. □

Figure 3.4: Non-manipulative scores on the left, utilities for manipulators m_1 and m_2 on the right for Example 3.

$s(a) = \mathbf{17}$
$s(b) = \mathbf{14}$
$s(c) = \mathbf{11}$
$s(d) = 7$
$s(e) = 6$
$s(f) = 5$

	u_{m_1}	u_{m_2}	u_{m_3}
a	1	1	1
b	1	1	1
c	1	1	1
d	9	10	10
e	11	10	10
f	10	12	10

3.2 Inconsistent *k*-Borda Manipulation

In consistent manipulation, the manipulators are limited to certain values of additional points that they can give to a candidate. Possible scores are in $s \in \{0, ..., r \cdot (m-1)\}$ for ranks $i \in \{0, ..., m-1\}$, where m denotes the number of candidates and r the number of manipulators. For example, for two manipulators and four candidates, only values of additional points $s \in \{0, 2, 4, 6\}$ are possible. The limited possibilities of a manipulation in the consistent setting simplifies computation, which was exploited in Section 3.1.

In this section we focus on a variant of inconsistent manipulation, where all manipulators vote consistently except one manipulator, which is allowed to modify her vote by swapping candidates. The number of positions that one manipulator is allowed to change from the consistent vote that the other manipulators cast, serves as a measure on how similar a manipulation is to the consistent variant.

Hence at first we consider a variant of inconsistent manipulation that allows one swap and how to computationally solve this variant in Section 3.2.1. Afterwards, we generalize this to a variant that allows multiple swaps in Section 3.2.2.

3.2.1 *k*-Borda Manipulation with One Swap

To explain the variant of *k-Borda* manipulation that allows one swap we start with a small example.

Example 3. In Figure 3.4 we fix non-manipulative scores and utility functions for three manipulators m_1, m_2 and m_3, candidates $C = \{a, b, c, d, e, f\}$, $k = 3$, and lexicographic tie-breaking.

Although d, e, f are the most favored candidates by the manipulators, not all three of them can win in the consistent variant. Candidates e and f, that are more favored than d can still win. Casting votes consistently, e and f get pushed into the k-egroup, as we can see in Figure 3.5. In the consistent variant it is not possible to give more than 9 points to all three of, d, e, and f because, as explained

Figure 3.5: Consistent and inconsistent manipulations for Example 3 and the preference profile in Figure 3.4.

(a) Scores by the manipulators in the consistent case

m_1	m_2	m_3
f	f	f
e	e	e
d	d	d
c	c	c
b	b	b
a	a	a

	old score	manipulative score	total score
a	17	0	**17**
b	14	3	17
c	11	6	17
d	7	9	16
e	6	12	**18**
f	5	15	**20**

(b) Scores by the manipulators in the inconsistent case

m_1	m_2	m_3
f	f	**d**
e	e	e
d	d	**f**
c	c	c
b	b	b
a	a	a

	old score	manipulative score	total score
a	17	0	17
b	14	3	17
c	11	6	17
d	7	11	**18**
e	6	12	**18**
f	5	13	**18**

before, the score s has to be in $s \in \{0, 3, 6, 9, 12, 15\}$. In this case, the outcomes for the manipulators are $\{a, e, f\}$, $\{a, d, f\}$ or $\{a, d, e\}$ winning the election. Their utilities for the possible k-egroups are $\text{util}(\{a, e, f\}) = 66$, $\text{util}(\{a, d, f\}) = 64$ and $\text{util}(\{a, d, e\}) = 63$. Note that a can be exchanged by b or c, because they have the same utility values.

In the inconsistent scenario, the manipulators can support d, e and f and push all three of them into the winning k-egroup. They can do this by swapping the first and third position in one manipulator's vote, so the possible scores are in $s \in \{0, 3, 6, 11, 12, 13\}$. The evaluation value of the winning k-egroup in that case would be $\text{util}(\{d, e, f\}) = 92$. To achieve this, they can cast their votes as in Figure 3.5.

Coalitional Manipulation with One Swap. In this variant, all manipulators cast exactly the same vote except one manipulator. This manipulator is allowed to choose two ranks i, j, such that $1 \leq i \leq j \leq m$, and to swap the candidates on these ranks. The vote she casts has the same candidate on rank i as the other manipulators on rank j and vice versa. We now formally define the variant that allows one swap.

k-Borda-$\mathcal{F}$-eval-Coalitional Manipulation with One Swap for eval $\in$ {util, candegal} (One-Swap-CM)

Input: An election (C, V), an egroup size $k < |C|$, r manipulators represented by their utility functions $U = \{u_1, ..., u_r\}$ where $u_i \colon C \to \mathbb{N}$ and a non-negative, integral evaluation threshold q.

Question: Is there a size-r multiset W of manipulative votes over C such that $(r-1)$ manipulators vote according to $\succ$, a linear order on C, and one manipulator votes according to $\succ_{i,j}$, that is emerging from swapping positions i, j in $\succ$ for two fixed ranks i, j, such that $1 \leq i \leq j \leq m$, and some k-egroup $S \subset C$ wins the election $(C, V \cup W)$ under $\mathcal{F}$ with $\text{eval}(S) \geq q$?

Let $r, r_{i,j} \colon C \to \{1, ..., m\}$ be two rank functions defined in Equation 2.1 of $\succ$ and $\succ_{i,j}$ respectively. For two candidates $c, d \in C$ and any other candidate $e \in C \setminus \{c, d\}$ the following holds:

$$r(c) = i = r_{i,j}(d),$$
$$r(d) = j = r_{i,j}(c),$$
$$r(e) = r_{i,j}(e).$$

To solve One-Swap-CM, we can reuse the idea of finding a matching on a bipartite graph with candidates and positions from Algorithm 3.1.

Theorem 3.2. *Let m be the number of candidates, n the number of voters, k the size of a desired egroup, and r the number of manipulators. One can solve* One-Swap-CM *in time $\mathcal{O}(m(n + r + m^4(m + n)))$ for any* eval $\in$ {util, candegal} *and* $\mathcal{F} \in \{\mathcal{F}_{\text{lex}}, \mathcal{F}^{\text{eval}}_{\text{opt}}, \mathcal{F}^{\text{eval}}_{\text{pess}}\}$.

Proof. We propose an algorithm for One-Swap-CM that reuses the idea from Algorithm 3.1.

Algorithm 3.2. The algorithm iterates through all possible value combinations of the following parameters:

- two ranks i, j, such that $1 \leq i \leq j \leq m$, that are swapped by one manipulator,
- the dropped candidate d, and
- the position i_d that d is ranked at by $(r - 1)$ manipulators.

We now define a *gain* mapping $g_{i,j}$. For each rank it assigns the number of additional points that a candidate gets at that rank.

Let $g_{i,j} \colon \{1, ..., m\} \to \mathbb{N}$ be as follows:

$$g_{i,j}(x) := \begin{cases} r(m - x) - (j - i) & \text{if } x = i, \\ r(m - x) + (j - i) & \text{if } x = j, \\ r(m - x) & \text{otherwise.} \end{cases} \tag{3.7}$$

We define H_{k-t} as the set of the $k-t$ ranks with the highest-valued gains from $g_{i,j}$ and $g_{\max}$ as in Equation 3.2. Note that $g_{i,j}$ might not be monotonically decreasing, so we do not necessarily have $H_{k-t} = \{1, ..., k-t\}$.

Fixing these values allows us to compute the final score z of the dropped candidate and the sets C^+, C^-, and C^* as defined in Equations 3.4. Note that the computation of z might also depend on the ranks i and j, so $z := g_{i,j}(i_d)$.

We skip an iteration if $|C^+| > k$ or $|C^*| < k-t$. We denote the number of *kept candidates* by $t = |C^+|$.

Let $G_d^{i_d} = (V \setminus \{d\} \cup M, E)$ be a graph with $M = \{1, ..., m\} \setminus \{i_d\}$, edges $E = E^0 \cup E^+ \cup E^-$ defined in Equation 3.5, and edge weights defined in Equation 3.6 in the definition of Algorithm 3.1. We now solve the MWBPM problem on $G_d^{i_d}$ to obtain the order $\succ$. By the choice of i and j that are fixed in every iteration, we also obtain $\succ_{i,j}$.

For every iteration we find orderings $\succ$ and $\succ_{i,j}$ and compute the winning k-egroup and its evaluation function. The algorithm chooses the orderings such the evaluation function is maximized.

Correctness. We first ensure that every optimal k-egroup $S \subset C$ is considered by Algorithm 3.2. To show this we make sure that for every optimal S, there are $\succ, \succ_{i,j}$ that can be found by our algorithm, such that voting with $\succ, \succ_{i,j}$ results in S winning. This means that $\succ$ can be respresented in the graph $G_d^{i_d}$ for an iteration of Algorithm 3.2 using the gain function $g_{i,j}$. The second ordering $\succ_{i,j}$ can be obtained by swapping i and j in $\succ$ which are fixed in the iteration.

In Algorithm 3.2 we ensure by the definition of the graph $G_d^{i_d}$ that the candidates that get pushed into the k-egroup always get ranked at the ranks with the highest gain. Although the new mapping $g_{i,j}$ might not be monotonically decreasing, H_{k-t} contains the ranks with the highest gain.

Let $S \subset C$ be a k-egroup with an optimal outcome of One-Swap-CM according to eval(S). Let $\succ, \succ_{i,j}$ be orderings that result in S winning if $(r-1)$ manipulators vote with $\succ$ and one manipulator votes with $\succ_{i,j}$. Suppose that $\succ$ cannot be represented as a matching in our graph, which only happens if some edge $\{c_x, y\}$ is missing in the graph $G_d^{i_d}$. This is only the case if one of the following holds:

1. $y \in H_{k-t}$ and candidate c_x cannot outperform the dropped candidate with $g_{i,j}(y)$ additional points,

2. $y \notin H_{k-t}$ and candidate c_x is neither kept nor rejected candidate, $c_x \in C^*$ and can outperform d with $g_{i,j}(y)$ additional points, or

3. $y \in H_{k-t}$ and candidate c_x is a kept or rejected candidate, $c_x \in C^+ \cup C^-$.

For the first and the third case, when giving $g_{i,j}(y)$ additional points to c_x does either not result in pushing c_x into the winning k-egroup, or c_x is already part of the winning group before casting manipulative votes. We swap the position of c_x

with some c_g that gets pushed into the winning k-egroup with less than $g_{i,j}(k-t)$ additional points and obtain orderings $\succ'$ and $\succ'_{i,j}$. Voting with $\succ'$ and $\succ'_{i,j}$ results in S winning. A candidate c_g exists because of the choice of t in our iteration.

If the second condition holds, then giving $g_{i,j}(y)$ additional points to c_x results in pushing c_i into the winning k-egroup. In this case there must be another candidate c_g that does not get pushed into the winning k-egroup with $g_{i,j}(k-t)$ additional points or more. We swap candidates c_x and c_g to obtain new orderings $\succ'$ and $\succ'_{i,j}$.

Let $S \in C$ be the optimal winning k-egroup for an election $E = (C, V)$. Suppose Algorithm 3.2 with the mapping in Equation 3.7 outputs orderings $\succ, \succ_{i,j}$ that lead to k-egroup S' winning the election such that $\mathrm{eval}(S') < \mathrm{eval}(S)$.

If S has the same amount of kept candidates t, as S', by the first part of our proof there must be a matching that represents S in Algorithm 3.2 for the iteration of t. So S must be chosen in the same iteration of the algorithm.

If S has a different amount of kept candidates t, then our algorithm also must take t into account because it iterates over all possible values of t. Then S must be found in a different iteration of the algorithm.

Running Time. Algorithm 3.2 only differs in running time from Algorithm 3.1 by iterating over the positions to be swapped. This makes a factor or m^2 for the number of iterations. Thus, the overall running time is $\mathcal{O}(m(n + r + m^4(m + n)))$. $\square$

3.2.2 *k*-Borda Manipulation with Multiple Swaps

To extend manipulation with one swap to more swaps, we use a metric d that counts the number of disagreements between two ranking lists $\succ_1, \succ_2$ on the candidate set C which we define as follows:

$$d(\succ_1, \succ_2) = |\{c \in C \mid r_{\succ_1}(c) \neq r_{\succ_2}(c)\}|.$$

We use our metric d to define the problem statement of the almost-consistent variant of *k-Borda* manipulation:

ALMOST-CONSISTENT-k-Borda-$\mathcal{F}$-eval-COALITIONAL MANIPULATION for $\mathrm{eval} \in \{\mathrm{util}, \mathrm{candegal}\}$ (ALMOST-CONSISTENT-CM)

Input: An election (C, V), an egroup size $k < |C|$, the number ℓ of positions that may differ, r manipulators represented by their utility functions $U = \{u_1, ..., u_r\}$ where $u_i \colon C \to \mathbb{N}$ and a non-negative, integral evaluation threshold q.

Question: Is there a size-r multiset W of manipulative votes over C and two linear orders $\succ_1, \succ_2$ with $d(\succ_1, \succ_2) \leq \ell$, such that there is a manipulator m_{swap} that votes with $\succ_2$ and all other manipulators vote with $\succ_1$ and some k-egroup $S \subset C$ wins the election $(C, V \cup W)$ under $\mathcal{F}$ with $\mathrm{eval}(S) \geq q$?

Conjecture 3.1. *Let m be the number of candidates, n the number of voters, k the size of a searched egroup, ℓ the number of positions that may differ and r the number of manipulators. One can solve* Almost-Consistent-CM *in polynomial time for any* $\text{eval} \in \{\text{util}, \text{candegal}\}$ *and* $\mathcal{F} \in \{\mathcal{F}_{\text{lex}}, \mathcal{F}^{\text{eval}}_{\text{opt}}, \mathcal{F}^{\text{eval}}_{\text{pess}}\}$ *with ℓ being constant.*

Algorithm 3.3. To solve Almost-Consistent-CM we can use the MWBPM. The algorithm for Almost-Consistent-CM iterates through all possible value combinations of the following parameters:

- a set of ℓ distinct ranks $L = \{r_1, ..., r_\ell\} \subseteq \{1, ..., m\}$ that may differ in $\succ_1$ and $\succ_2$,
- a permutation $h\colon \{r_1, ..., r_\ell\} \to \{r_1, ..., r_\ell\}$,
- the dropped candidate d, and
- the position i_d that d is ranked at by $(r-1)$ manipulators.

The algorithm iterates over all possible permutations $h\colon \{r_1, ..., r_\ell\} \to \{r_1, ..., r_\ell\}$ of the chosen ℓ ranks, including the identity, where no swaps are performed, because we want to cover all possible manipulations with up to ℓ positions that differ from the consistent vote.

Similar to the gain function defined in Equation 3.7 we can now compute a new gain function $g_h\colon \{1, ..., m\} \to \mathbb{N}$ for at most ℓ positions that differ from the consistent vote using the permutation h that we fixed.

$$g_h(x) := \begin{cases} r(m-x) & \text{if } x \notin L, \\ (r-1)(m-x) + m - h(x) & \text{otherwise.} \end{cases} \tag{3.8}$$

As in Equation 3.3, we define H_{k-t} as the set of the $k-t$ ranks with the highest-valued gains from g_h and the maximal gain $g_{\max}$ as in Equation 3.2. Let $G_d^{i_d} = (V \setminus \{d\} \cup M, E)$ a graph with edges $E = E^0 \cup E^+ \cup E^-$ defined in Equations 3.5, and edge weights defined in Equation 3.6. We solve the MWBPM problem to obtain the order $\succ$ on $G_d^{i_d}$ and $\succ_h$ by applying h on $\succ$. Then $d(\succ, \succ_h) \leq \ell$ holds by the choice of L and h.

Running Time. Since we have to guess ℓ distinct ranks and all possible permutations, we have a factor of $m^{\underline{\ell}} := \binom{m}{\ell}\ell!$ in our running time. This then makes a running time of $\mathcal{O}(m(n + r + m^2 m^{\underline{\ell}}(m + n)))$. We leave the proof of correctness subject of future research, but remark that it is similar to the two proofs of Theorems 3.1 and 3.2 that we have seen above.

According to our conjecture, we can solve Almost-Consistent-CM in polynomial time for a fixed ℓ, which is the number of positions that one manipulator is allowed to modify from the consistent vote that the other manipulators cast. We leave the question open, whether it is also solvable in fixed parameter tractable when parameterized by ℓ.

4 Experiments

In this chapter we evaluate four different algorithms that where proposed by Bredereck et al. [BKN17]. Our goal is to confirm tractablility by experiments and to compare how effective the different variants of manipulation are. We provide two different measures of effectiveness, which we introduce in Section 4.5 as well as an implementation and statistics about the running time of tie-breaking and ℓ-*Bloc* manipulation.

4.1 Experimental Setup and Test Data

The following tests where conducted on a server running a 64bit version of Ubuntu 18.04.4 LTS with 256GB RAM and an Intel(R) Xeon(R) W-2125 4.00GHz processor with 4 cores. We decided to use Python as programming language, as well as the PrefLib Library [MW13] and the Gurobi ILP solver [Gur19]. The implementation is hosted on GitHub.

Data Sets. We generated most of the test data by using the PrefLib Data Generation tool. For the distribution of preferences, PrefLib offers five different models to generate data from which we use the following two:

- The *Impartial Culture* model assumes that the probability of observing preference orders is equally likely for each voter.
- *Mallows Mixture Model* assumes there is one or multiple true rankings and that individuals deviate from the ground truth with exponentially decreasing probability as the ranking moves away from the reference. In our test we used one variant with only one reference and one with five references.

Apart from the data generation tool, we used real-world election datasets for testing, the election for the city council of San Francisco. The datasets are available on the PrefLib Webpage provided by O'Neill [ONe13] and there are election data for several years.

Generating Utilities. For both, using generated data and real-world data, we need manipulators and their utility functions as an input for the manipulation algorithms. Hence, we chose voters from the dataset randomly and used their votes to compute utility functions by Borda ranks. We then removed the manipulative votes from the election to obtain the non-manipulative votes and utilities of the manipulators as input for the algorithms.

Note that in future work one can also compare truthful and untruthful votes by not removing the manipulators from the dataset.

Unfortunately there is no real-world data on manipulators' utilities and no other, more sophisticated tool to generate manipulators' utilities. This is why in this work we decided to consider, how randomly selected groups of voters would manipulate together. In future research one could also select groups of voters whose votes are more similar to each other. One way to do this is for example to choose voters from one side of a single peaked preference profile. Another way can be to measure the distance between voters' ranking lists, for example using Kendall Tau distance and pick groups that are close to each other.

4.2 Tie-Breaking

In Chapter 2 we already described how ties can be solved by *lexicographic*, *pessimistic*, and *optimistic* tie-breaking rules. Since manipulation also has to take tie-breaking rules into account, we have to consider the complexity of tie-breaking when we talk about manipulation. In some cases, tie-breaking is already NP-hard, which then also makes coalitional manipulation hard.

Bredereck et al. [BKN17, p. 16] showed that optimistic and pessimistic tie-breaking can be solved in polynomial time for the *utilitarian* and *candidate-wise egalitarian* variant.

For the *egalitarian* variant Bredereck et al. [BKN17, p. 17] showed that *pessimistic* tie-breaking can be solved in polynomial time and *optimistic* tie-breaking is NP-hard. Moreover, for the *optimistic egalitarian* variant, Bredereck et al. [BKN17, p. 20] proposed an algorithm that is fixed parameter tractable (FPT) when parameterized by $r + u_{\text{diff}}$, where u_{diff} denotes the number of different utility values.

Egalitarian Optimistic Tie-Breaking. We implemented the FPT algorithm by Bredereck et al. [BKN17, pp. 20–21] for *egalitarian optimistic tie-breaking* by using the Gurobi Optimizer for Python, a library for solving linear programs [Gur19].

The algorithm is an *integer linear program (ILP)* which creates variables for every possible type vector of utility functions. The type of a candidate $c_i \in C$ from the candidate set C is defined as vector $t = (u_1(c_i), ..., u_r(c_i))$ with $u_1, ..., u_r$ denoting the utility functions of manipulators $1, ..., r$.

To enforce different values for u_{diff} we used utility values generated by Borda-scores that are substituted by 0 for ranks above or equal u_{diff}.

To measure the running time, we use test data drawn from Impartial Culture in Figure 4.1. We can observe that r has a greater impact on the running time than u_{diff}, which made it difficult for us to find a good test range for r where

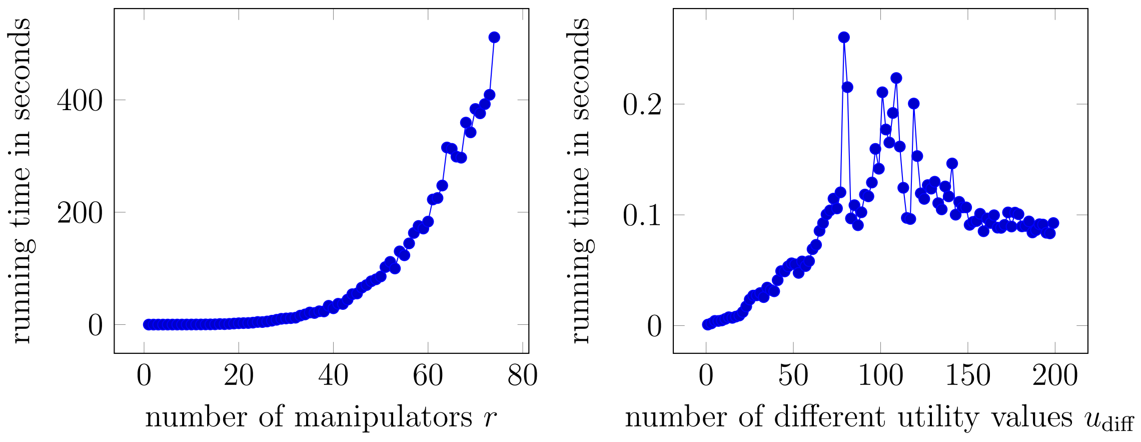


Figure 4.1: Running time for different values of r (left) and u_{diff} (right), u_{diff}. We averaged over 40 (left) and 100 (right) iterations and fixed values of $r = 6$ (right), $m = 400$ (right), $u_{\text{diff}} = m = 100$ (left), and $k = 10$.

running times are feasible in our setup. In the algorithm, u_{diff} and r are used as an upper bound u_{diff}^r for the number of possible type vectors of utilities, because the ILP creates one variable for every possible type. The use of Impartial Culture probably leads to creating many different type vectors of utilities, which does not increase anymore from values of around $u_{\text{diff}} = 200$ and when leaving the number of manipulators fixed. For a low number of u_{diff}, there are many candidates of the same type of utility vectors, which leads to less variables created. The peak around $u_{\text{diff}} = 100$ can be explained by having some candidates of the same type, and some which are the only ones for their type, which makes is hard for the ILP solver to find an underlying structure for some instances. For u_{diff} that is more than one forth of the candidates, there is probably only one candidate per type, which is why the running time does not increase anymore from that point because the number of manipulators and candidates are fixed.

Overall, we can observe that running times for elections with $m = 100$ candidates are still feasible to compute for 80 manipulators in our setup and even quickly to compute with less than 40 manipulators.

4.3 Consistent Manipulation of ℓ-Bloc

In the consistent variant of *ℓ-Bloc* manipulation, all voters cast the same vote. Bredereck et al. [BKN17, p. 26] showed that this variant can be solved in $\mathcal{O}(m(m+r+n))$ time for in the *utilitarian* and *candidate-wise egalitarian* variant and any of our introduced tie-breaking rules. Their algorithm uses the fact that for the *utilitarian* and *candidate-wise egalitarian* variants, we can assume that there exists a function that computes the utility of a candidate group for all manipulators. We showed how to compute this function in Equation 2.5 and 2.6.

The algorithm first computes a strength order and iterates over every possible choice of a *dropped candidate*, the strongest candidate that is not part of the winning k-egroup. The algorithm then compares all possible solutions and picks the one that is most liked by the manipulators, meaning the one with the highest value of the evaluation function eval $\in$ {util, candegal}.

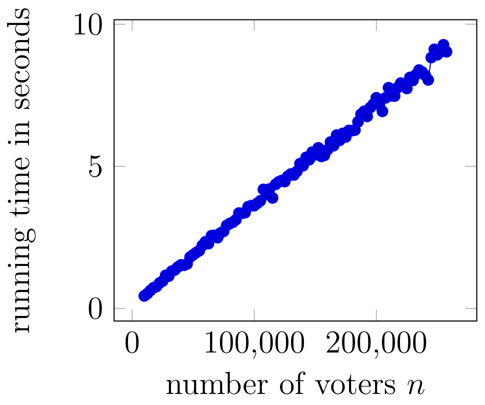
To measure the running time, we use the *utilitarian* evaluation function and test data drawn from Impartial Culture in Figure 4.2. As expected, the running time increases linearly with increasing n. We can observe that for increasing m, the running time also only increases linearly, although the theoretical running time proposes quadratic increase. This is because ℓ was approximated by m in the proof of running time [BKN17, pp. 28–29]. Hence, we added a fourth test to measure the running time for increasing ℓ. We can observe that when ℓ gets close to all or no candidates, the running time grows faster because of higher differences in the structure. For the increase of r, the running time does not significantly increase anymore from $r = 50$. In the proof of running time [BKN17, pp. 28–29], r used to compute the value of a candidate and to compute which candidates can still win when getting r additional points. In our implementation of the algorithm we do not compute all values of candidates in the beginning, but only of the candidates needed during the algorithm. We did this because we observed that in every iteration there are usually only a few candidates that need to be considered as candidates that can outperform the dropped candidate. In this way we can save computation time.

We also observe that the algorithm is well suited also for high numbers of candidates, voters and manipulators.

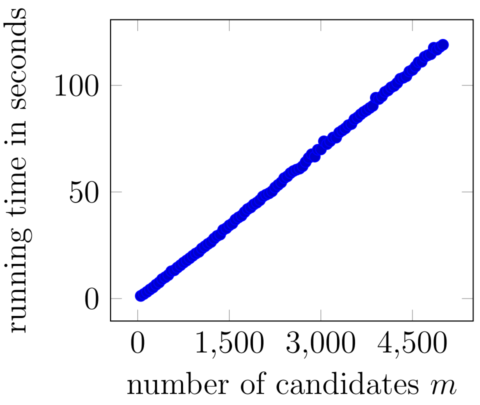
4.4 Utilitarian and Candidate-wise Egalitarian Variant of ℓ-Bloc

Inconsistent *ℓ-Bloc* manipulation is NP-hard in general, but solvabe in polynomial time for the *utilitarian* and *candidate-wise egalitarian* case as showed by Bredereck et al. [BKN17, p. 23]. Their algorithm uses the single utility functions for the *utilitarian* and *candidate-wise egalitarian* variants from Equation 2.5 and 2.6. As observed before, for the *egalitarian* variant it is not possible to compute a single utility function. We therefore consider the *egalitarian* variant separately in 4.6.

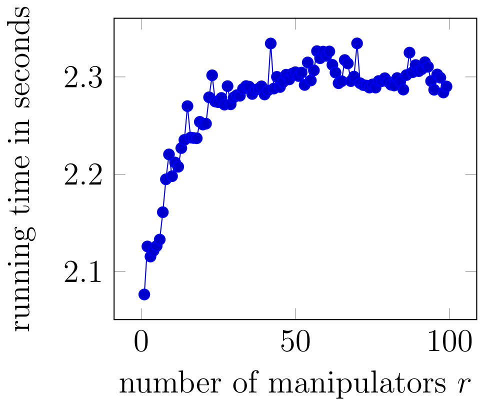
The algorithm for the *utilitarian* and *candidate-wise egalitarian* variant iterates over all possible combinations of the least preferred member of the k-egroup and its score. Each of these iteration can then be reduced to the k-item Knapsack Problem which can be solved in $\mathcal{O}(k^2mr)$ as described in Chapter 2. This makes an overall running time of $\mathcal{O}(k^2m^2(n + r))$.



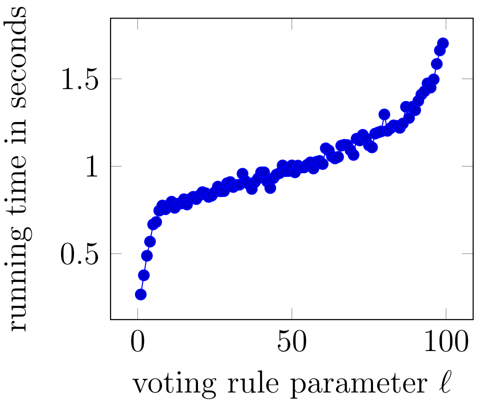
(a) Running time for different number of voters, $r = 7, m = 100$, $\ell = 8$ and $k = 7$. Average for ten iterations for each n.



(b) Running time for different number of candidates, $r = 7, n = 10000$, $\ell = 8$ and $k = 7$, not averaged.



(c) Running time for different number of manipulators, $n = 10000, m = 100$, $\ell = 8$ and $k = 7$. Average for ten iterations for each r.



(d) Running time for different parameters of ℓ-*Bloc*-rule, $n = 10000, m = 100$, $r = 7$ and $k = 7$. Average for fifty iterations for each ℓ.

Figure 4.2: Running time for different values of m, n, ℓ, and r in the consistent variant.

Implementation of the Exact-k-Item Knapsack Problem E-kKP. We implemented the algorithm for E-kKP using dynamic programming as suggested by Kellerer et al. [KPP04, pp. 273–275]. The input of our knapsack implementation are the weight and value lists which hold weight and value of each knapsack item. As knapsack items we use all candidates that can possibly join the egroup. We compute their value using the single utility function and their weight by how many approvals are needed to push this candidate into the winning k-egroup.

To store subresults of the knapsack weights and items chosen, we use two two-dimensional lists. In every iteration of the algorithm, the computation of the subproblem only relies on the iteration before, so we only have to store two matrices at a time.

The algorithm also requires to guess the final value P of the knapsack. We do that by iterating over all possible values for P, bounded by the sum of the k most valuable items.

Results of Running Time Analysis. To measure the running time, we use test data drawn from Impartial Culture in Figure 4.3.

We can observe that parameter k has the strongest impact on running time, which made it especially difficult to find a good test range. Different than in the consistent algorithm in Section 4.3, k is used heavily in the knapsack algorithm, where the dynamic algorithm uses different sizes of subproblems to compute a solution for all k knapsack items.

In the plot for the number of voters, there are some outliers for small values of n, for which the reasons are unclear. Reasons could be technical overhead or instances that have a complex structure for a small number of voters.

While for m, the running time behaves clearly quadratically, as expected, for r the running time also looks quadratic from 0 to 20, before the plot starts to flatten to linear growth. Different to the consistent variant in Section 4.3, where r is only used to compute the candidates' values and the set of candidates that can possible join the k-egroup, here r is also used to compute the E-kKP capacity. This is why we think that r has a greater impact on the inconsistent algorithm than on the consistent algorithm in Section 4.3. The algorithm also uses the knapsack capacity and the approximation of the knapsack weight as sizes to create two-dimensional lists. Hence, a reason for the quadratic growth for small instances can be that the capacity and approximation value are dominated by r for small instances.

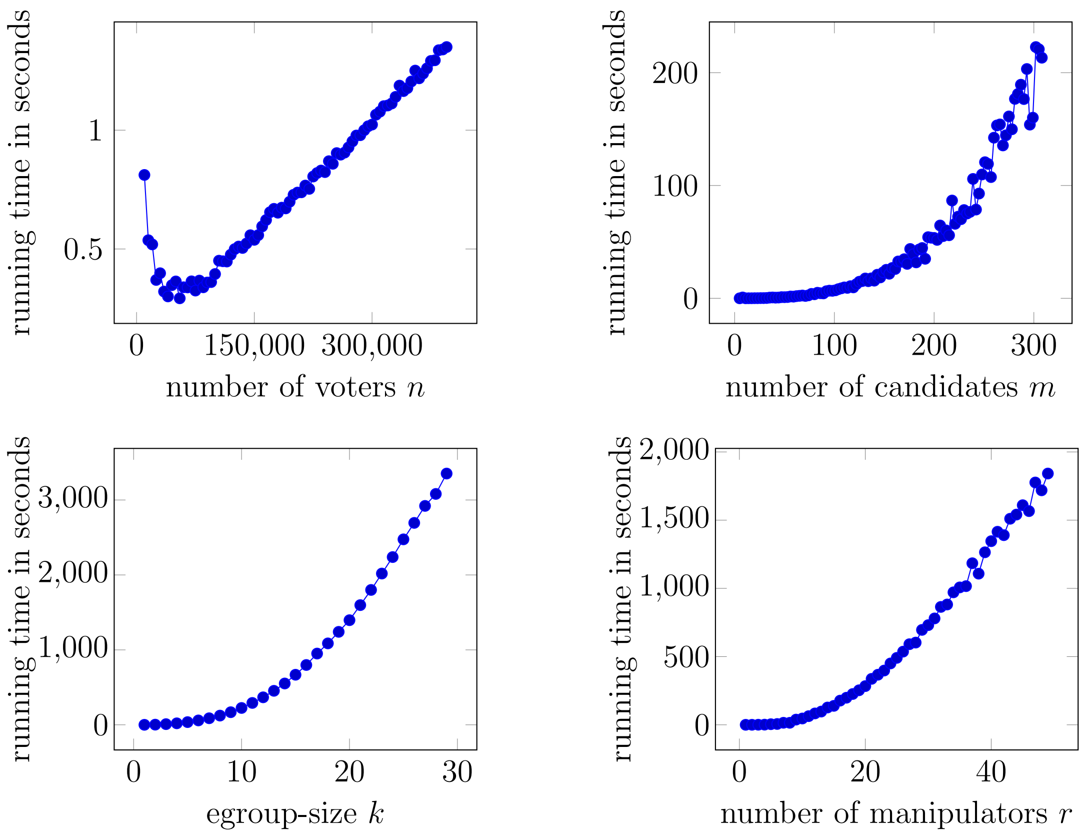


Figure 4.3: Running time for different m, n in the utilitarian variant. We fix $r = 7$, $\ell = 8$, $k = 7$. On the right we fix $m = 100$, on the left we fix $n = 10000$ and average over thirty iterations.

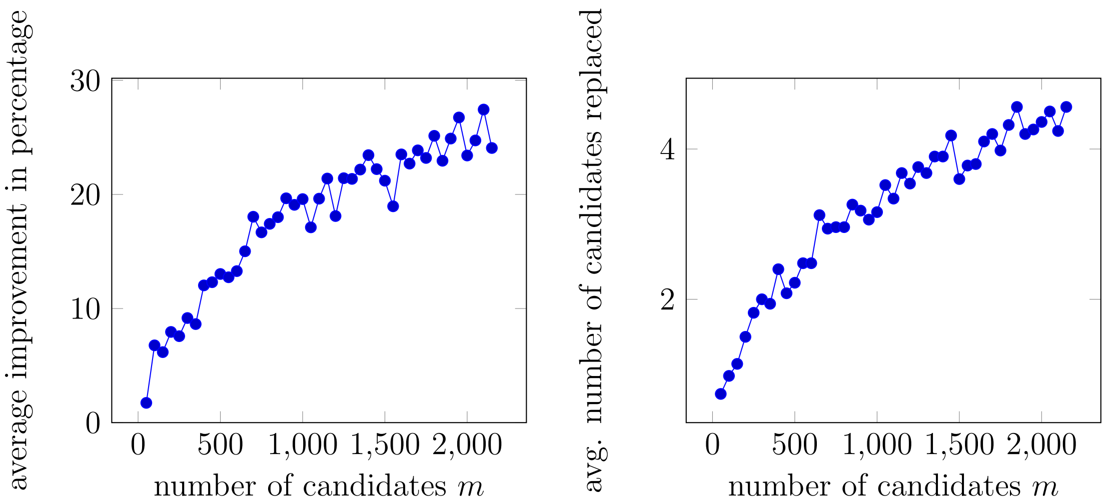


Figure 4.4: Average percentage/candidates replaced for 50 times for each number of candidates m for the consistent variant. We fix $k = 8, r = 5, \ell = 7, n = 10000$.

4.5 Likelihood of ℓ-Bloc Manipulation

In this section we want to measure how likely it is to actually change the outcome of an election, say how effective our algorithms are. We also want to compare the effectiveness of consistent ℓ-*Bloc* manipulation and inconsistent ℓ-*Bloc* manipulation.

To measure how effective a manipulation is, we propose the following two values:

- The number of candidates that differ from the winning egroups in the elections with and without manipulative voters and
- the improvement in percentage by computing the evaluation function for the outcome in the elections with and without the manipulative group.

First, we would like to study the correlation of these two values. Therefore we computed both measures for the Impartial Culture distribution when changing the number of manipulators in the consistent setting. We chose the consistent setting because the running times are much lower. In Figure 4.4 we can see that both measures of effectiveness behave in the same way for our setup.

To compare the effectiveness of consistent and inconsistent manipulation, we compare the number of exchanged candidates for different distributions of test data and the real-world dataset of the San Francisco election.

Therefore we fix $m = 10$ candidates and increase the ratio of manipulators among the voters in Figure 4.5.

For both, the consistent and inconsistent setting, we can observe a clear trend against exchanging between two and three candidates in all distributions.

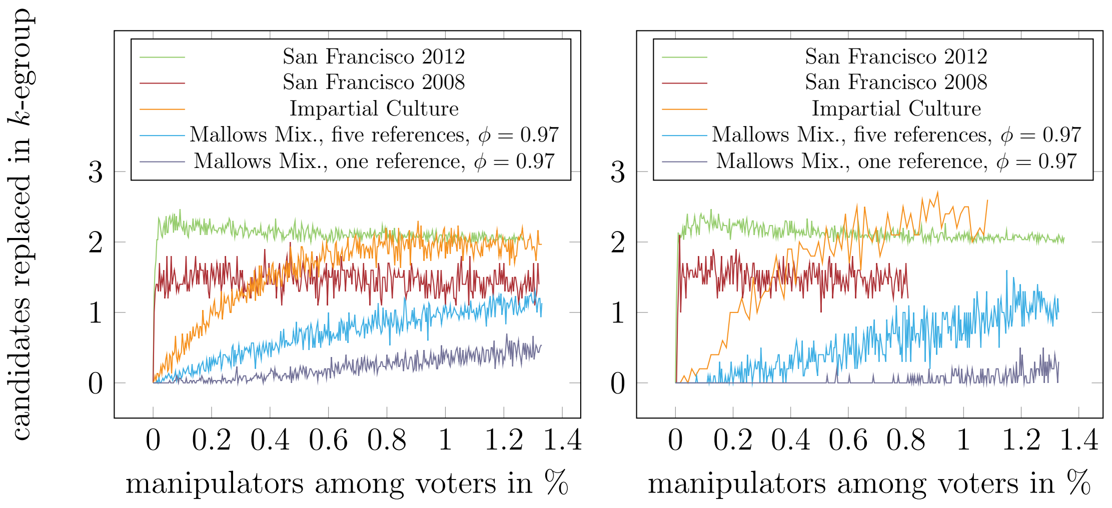


Figure 4.5: Number of candidates replaced in the consistent (left) and inconsistent (right) manipulation setting for different distributions of test data and utilitarian evaluation. We fix $n = 30000, m = 10, k = 6, \ell = 7$ and average for 30 iterations. The manipulators were chosen randomly from the set of voters.

This can be explained by most of the voters being satisfied by changing almost half of the k-egroup in the general case.

For the San Francisco election of 2008 we can observe for both the consistent and the inconsistent variant that the manipulators only exchange between one and two candidates. Since this appears in both variants of manipulation we can explain this by the choice of candidates in the election. It may happen that there are candidates that are not approved by many voters, which looks as if they are not part of the election.

We also observe that the number of candidates exchanged is roughly the same for both consistent and inconsistent ℓ-*Bloc* manipulation in all distributions. This indicates that when using the average number of candidates exchanged as measure for effectiveness, that the inconsistent variant of ℓ-*Bloc* manipulation is roughly as effective as the consistent variant.

In future work, one can also compare the effectiveness of consistent and inconsistent manipulation in terms of the average improvement of the evaluation function.

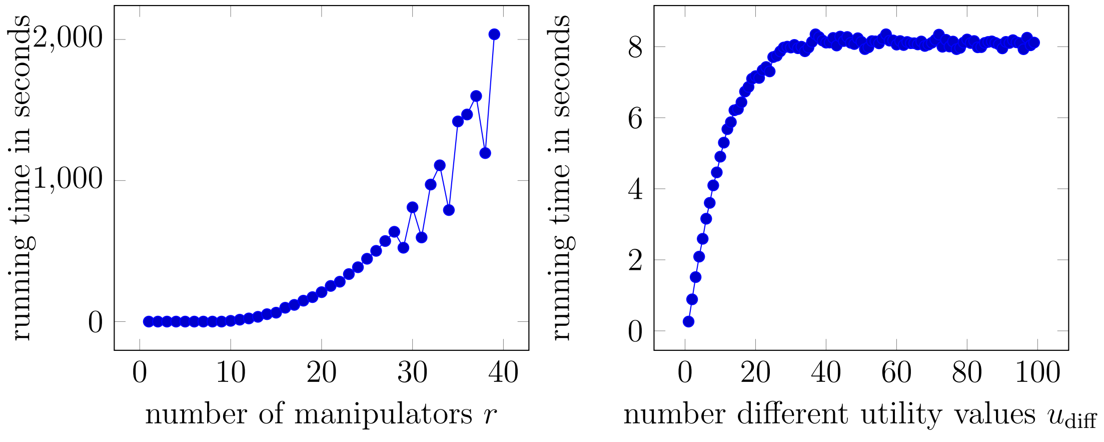


Figure 4.6: Running time for different values of r and u_{diff}. We fix $u_{\text{diff}} = m$ (left), $r = 10$ (right), $n = 200, m = 100, \ell = 6$ and $k = 10$.

4.6 Egalitarian Variant

The egalitarian variant of *ℓ-Bloc* manipulation is fixed-parameter tractable when parameterized by $r + u_{\text{diff}}$, with u_{diff} denoting the number of different utility values as showed by Bredereck et al. [BKN17, p. 34]. They suggest an *Integer Linear Program* to solve this variant.

We implemented the ILP using the Gurobi ILP solver [Gur19]. The algorithm iterates over all possible combinations of the lowest final score z of a member of the k-egroup and numbers of candidates that either get pushed into the k-egroup with a higher score than z or have score exactly z when taking the manipulative votes into account. For every combination of these values we then create an ILP. Similar to the concept of types in the consistent algorithm in Section 4.3, we create two variables for every group. Two candidates belong to the same group, if they have the same utility vector and need the same number of additional approvals to get z or more than z points.

To measure the running time, we use test data drawn from Impartial Culture in Figure 4.6. While the running time for r increases as expected, we can observe that for u_{udiff}, the curve starts to flatten around $m = 25$, which is about one forth of the candidates. We observed the same thing already in the ILP for egalitarian tie-breaking in Section 4.2 for about one forth of the candidates. Again we can explain this by the use of Impartial Culture. At a certain point, the utility vectors of the candidates are so different, that there is only one candidate per possible group or type of utility vector (for the egalitarian tie-breaking). From this point the running time does not increase anymore because the number of manipulators and candidates are fixed.

4.7 Conclusions from the Experiments

In the experiments we were able to confirm tractability results and even improve the running times from Bredereck et al. [BKN17]. We learned that, although the inconsistent variant is more computationally demanding, the consistent and inconsistent variant roughly exchange the same number of candidates from a winning k-egroup. Manipulators intuitively rather decide to vote consistently to achieve their goal, and we were able to show that this simplistic intuition gives relatively good results.

It remains to be seen in future experiments whether the number of candidates exchanged is a good measure of effectiveness or if one should rather use the average improvement of the evaluation function. We suggest to compare the two proposed measures on more real-world datasets to gain more insight. We also suggest to compare the consistent and inconsistent variant with the truthful election results for *ℓ-Bloc*.

In this thesis we analyzed, how randomly selected groups of voters would manipulate together. Hence, one can study different ways of choosing the manipulative voters. This can be done by for example using the Kendall Tau distance to select voters with similar utility vectors. In future experiments one can also use other techniques to generate manipulators' utilities or even collect real-world data.

For the *egalitarian* variant of *ℓ-Bloc* manipulation, one can extend the proposed ILP to show that the consistent egalitarian variant remains fixed parameter tractable.

We would also like to experimentally confirm the tractability results from Chapter 3 for *k-Borda* manipulation and compare the effectiveness of manipulation.

5 Conclusion

In this work we analyzed the complexity of *k-Borda* manipulation for both variants, the consistent variant, where manipulators cast exactly the same vote, and the inconsistent variant. Although voting consistently might come intuitively natural to a group of manipulators, we gave examples where manipulators are better off when voting inconsistently. We showed that the consistent variant is easier to compute and less complex than the inconsistent variant. Indeed we provide a polynomial-time algorithm for consistent *k-Borda* whereas the inconsistent variant is known to be NP-hard already in the singlewinner case [Dav+11, p. 658] [BNW11, p. 58].

For the inconsistent variant of *k-Borda* we provide an algorithm that runs in polynomial time to solve ONE SWAP-CM, which is similar to the consistent variant. We used this result to conjecture that the variant of *k-Borda* manipulation where one manipulator is allowed to modify her vote from the consistent vote that the other manipulators cast, is solvable in polynomial time when fixing the number of positions that differ. One can say that according to our conjecture, the manipulation problem gets more computationally demanding as the number of positions that differ increases, but is still polynomial-time solvable when the number of positions is fixed. Subject to further research is whether the complexity of inconsistent manipulation depends only on the number of positions without a combination with another parameter, i.e. whether it is solvable in FPT when parameterized by the number of positions that differ.

In comparison to *ℓ-Bloc*, *k-Borda* has desirable properties, such as *committee monotonicity* and *unanimity* which make *k-Borda* suitable for the use in shortlisting applications. In future work one can focus on other excellence-based voting rules as well as on voting rules that aim for diversity or proportional representation. We think that our Algorithm 3.1, which uses MAXIMUM WEIGHT BIPARTITE PERFECT MATCHING to solve consistent *k-Borda* manipulation can be extended to manipulate other scoring-based voting rules by substituting the *gain* function.

For *ℓ-Bloc* we implemented algorithms from Bredereck et al. [BKN17], verified tractability, and compared the effectiveness of different variants of manipulation to actually change the outcome of an election. Manipulators intuitively rather decide to vote consistently to achieve their goal, and we were able to show that this simplistic intuition gives relatively good results. We measured the effectiveness of a manipulation by the average number of candidates that get exchanged in a winning group by manipulation and observed that consistent and inconsistent *ℓ-Bloc* manipulation have roughly the same effectiveness in our experiments. This was an unexpected result, because the inconsistent variant of *ℓ-Bloc* is more general and more challenging to compute. It remains to be seen whether the number of candidates exchanged is a good measure of effectiveness or if one should rather use the average improvement of the evaluation function. We suggest to compare these two measures to gain more insight into these two measures of effectiveness by collecting more data from experiments on other real-world datasets and other distributions in future work. We also suggest to compare the consistent and inconsistent variants with the truthful election result.

Another difficulty in our experiments was, that there is no real-world data on manipulators' utilities. In future work, one can study different ways of generating or even collecting real-world data on manipulators' utilities, other than using Borda-ranks. We also suggest to experimentally compare the effectiveness of the consistent and inconsistent variants of *k-Borda*.

To conclude, the most important lesson that we learned from analyzing time complexity for *ℓ-Bloc* and *k-Borda* is that the consistent case is usually much easier to compute and also very effective. This gives us new perspectives on the time complexity of manipulation of other voting rules, because the consistent case can be studied separately in future work.

Literature

[BKN17] R. Bredereck, A. Kaczmarczyk, and R. Niedermeier. “On coalitional manipulation for multiwinner elections: shortlisting”. In: *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI ’17)*. Long version available as arXiv:1806.10460. AAAI Press. 2017, pp. 887–893 (cit. on pp. 4, 6, 7, 11, 14–16, 30–33, 39–41).

[BNW11] N. Betzler, R. Niedermeier, and G. J. Woeginger. “Unweighted Coalitional Manipulation under the Borda Rule Is NP-Hard”. In: *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI ’11)*. AAAI Press, 2011, pp. 55–60 (cit. on pp. 9, 10, 17, 41).

[BO91] J. J. Bartholdi and J. B. Orlin. “Single transferable vote resists strategic voting”. In: *Social Choice and Welfare* 8.4 (1991), pp. 341–354 (cit. on p. 10).

[BR16] D. Baumeister and J. Rothe. “Preference Aggregation by Voting”. In: *Economics and Computation, An Introduction to Algorithmic Game Theory, Computational Social Choice, and Fair Division*. Ed. by J. Rothe. Springer, 2016, pp. 197–325 (cit. on pp. 9, 10).

[BTT89] J. J. Bartholdi, C. A. Tovey, and M. A. Trick. “The computational difficulty of manipulating an election”. In: *Social Choice and Welfare* 6.3 (1989), pp. 227–241 (cit. on p. 10).

[CLS03] V. Conitzer, J. Lang, and T. Sandholm. “How many candidates are needed to make elections hard to manipulate?” In: *Proceedings of the 9th Conference on Theoretical Aspects of Rationality and Knowledge (TARK ’03)*. ACM, 2003, pp. 201–214 (cit. on p. 10).

[CS02] V. Conitzer and T. Sandholm. “Complexity of Manipulating Elections with Few Candidates”. In: *Proceedings of the 18th National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence (AAAI ’02)*. AAAI Press, 2002, pp. 314–319 (cit. on pp. 6, 9, 11).

[CW16] V. Conitzer and T. Walsh. “Barriers to Manipulation in Voting”. In: *Handbook of Computational Social Choice*. Ed. by F. Brandt, V. Conitzer, U. Endriss, J. Lang, and A. D. Procaccia. Cambridge University Press, 2016, pp. 127–145 (cit. on pp. 9, 10).

[Dav+11] J. Davies, G. Katsirelos, N. Narodytska, and T. Walsh. “Complexity of and Algorithms for Borda Manipulation”. In: *Proceedings of the 25th AAAI Conference on Artificial Intelligence (AAAI ’11).* AAAI Press, 2011 (cit. on pp. 9, 10, 17, 41).

[DS12] R. Duan and H.-H. Su. “A scaling algorithm for maximum weight matching in bipartite graphs”. In: *Proceedings of the 23rd annual ACM-SIAM symposium on Discrete Algorithms (SODA ’12).* Society for Industrial and Applied Mathematics. 2012, pp. 1413–1424 (cit. on pp. 16, 17, 23).

[Elk+17] E. Elkind, P. Faliszewski, P. Skowron, and A. Slinko. “Properties of multiwinner voting rules”. In: *Social Choice and Welfare* 48.3 (2017), pp. 599–632 (cit. on p. 10).

[End+16] U. Endriss, S. Obraztsova, M. Polukarov, and J. S. Rosenschein. “Strategic Voting with Incomplete Information”. In: *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI ’16).* AAAI Press, 2016, pp. 236–242 (cit. on p. 6).

[Fal+17] P. Faliszewski, P. Skowron, A. Slinko, and N. Talmon. “Multiwinner voting: A new challenge for social choice theory”. In: *Trends in Computational Social Choice* 74 (2017), pp. 27–47 (cit. on pp. 9, 10).

[Gur19] Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual.* 2019. URL: http://www.gurobi.com (cit. on pp. 30, 31, 39).

[KPP04] H. Kellerer, U. Pferschy, and D. Pisinger. “Multidimensional Knapsack Problems”. In: *Knapsack Problems.* Springer Berlin Heidelberg, 2004, pp. 235–283 (cit. on pp. 16, 35).

[Lin11] A. Lin. “The Complexity of Manipulating k-Approval Elections”. In: *Proceedings of the 3rd International Conference on Agents and Artificial Intelligence, Volume 2, (ICAART ’11).* Ed. by J. Filipe and A. L. N. Fred. SciTePress, 2011, pp. 212–218 (cit. on p. 11).

[Mei+08] R. Meir, A. D. Procaccia, J. S. Rosenschein, and A. Zohar. “Complexity of Strategic Behavior in Multi-Winner Elections”. In: *Journal of Artificial Intelligence Research* 33 (2008), pp. 149–178 (cit. on p. 11).

[MW13] N. Mattei and T. Walsh. “PrefLib: A Library of Preference Data HTTP://PREFLIB.ORG”. In: *Proceedings of the 3rd International Conference on Algorithmic Decision Theory (ADT ’13).* Springer, 2013 (cit. on p. 30).

[ONe13] J. O’Neill. *San Fransisco Election Data.* 2013. URL: http://www.preflib.org/data/election/sf/ (cit. on p. 30).

[OZE13] S. Obraztsova, Y. Zick, and E. Elkind. “On manipulation in multi-winner elections based on scoring rules”. In: *Proceedings of the 12th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS ’13)*. Ed. by M. L. Gini, O. Shehory, T. Ito, and C. M. Jonker. IFAAMAS, 2013, pp. 359–366 (cit. on p. 11).

[PRZ07] A. D. Procaccia, J. S. Rosenschein, and A. Zohar. “Multi-winner Elections: Complexity of Manipulation, Control, and Winner-determination”. In: *Proceedings of the 20th International Joint Conference on Artifical Intelligence (IJCAI ’07)*. Morgan Kaufmann Publishers Inc., 2007, pp. 1476–1481 (cit. on p. 11).

[Sch+19] J. Scheuerman, J. L. Harman, N. Mattei, and K. B. Venable. “Heuristics in Multi-Winner Approval Voting”. In: *CoRR* abs/1905.12104 (2019). arXiv: `1905.12104` (cit. on p. 6).